开放可编程
V100R021C10
开发指南

文档版本 01

发布日期 2023-04-24

华为技术有限公司

版权所有 © 华为技术有限公司 2023。 保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明

和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 预备知识 1
1.1 了解编程语言和协议 1
1.2 了解软件包结构 2
1.3 了解日志记录 2
1.3.1 查看日志 2
1.3.2 设置日志级别 3
2 开发流程 4
3 开发准备 5
4 开发环境部署 6
4.1 安装 Python 解析器 6
4.2 安装开放可编程 SDK 7
4.3 安装集成开发环境 7
4.4 安装 Gpg4Win 及生成签名用密钥 8
4.5 安装开放可编程 mini 软件包 9
5 开发 SND 11
5.1 开发 NETCONF 协议设备原子驱动包 11
5.1.1 创建原子驱动包模板 11
5.1.2 开发原子驱动包 12
5.1.2.1 编辑软件包配置文件 12
5.1.2.2 编写设备 YANG 模型 15
5.1.2.3 开发代码 18
5.1.2.3.1 引入必要的头文件 18
5.1.2.3.2 编写设备纳管定制 18
5.1.2.3.3 编写设备类型能力定制 22
5.1.2.3.3.1 配置设备驱动数据 22
5.1.2.3.3.2 设置业务自定义处理（可选） 28
5.1.2.3.3.3 设置数据前置处理（可选） 29
5.1.2.3.3.4 设置数据后置处理（可选） 29
5.1.2.3.3.5 设置数据联动（可选） 29
5.1.2.3.3.6 获取 syncId（可选） 30
5.1.2.3.4 编写设备特性能力定制（可选） 31
5.1.3 验证原子驱动包 32

 版权所有 © 华为技术有限公司

5.1.3.1 验证 YANG 文件 32
5.1.4 编译原子驱动包 33
5.1.5 开发 NETCONF SND 示例 34
5.1.5.1 开发 NE8000 M8 NETCONF SND 示例 34
5.1.5.2 开发 S6720 NETCONF SND 示例 36
5.2 开发 CLI 协议设备原子驱动包 38
5.2.1 创建原子驱动包模板 38
5.2.2 开发原子驱动包 39
5.2.2.1 编辑软件包配置文件 39
5.2.2.2 编写设备 YANG 模型 42
5.2.2.2.1 节点定义 42
5.2.2.2.2 优化模型和使用扩展项 44
5.2.2.2.3 验证模型 45
5.2.2.2.4 YANG 模型自定义功能 45
5.2.2.2.5 实例开发 YANG 模型 48
5.2.2.3 开发代码 50
5.2.2.3.1 引入必要的头文件 50
5.2.2.3.2 编写设备纳管定制 50
5.2.2.3.3 编写设备类型能力定制 53
5.2.2.3.3.1 配置设备驱动数据 53
5.2.2.3.3.2 设置业务自定义处理（可选） 65
5.2.2.3.3.3 设置数据前置处理（可选） 66
5.2.2.3.3.4 设置数据后置处理（可选） 67
5.2.2.3.3.5 设置数据联动（可选） 67
5.2.2.3.3.6 获取 syncId（可选） 68
5.2.2.3.4 编写设备特性能力定制（可选） 68
5.2.3 验证原子驱动包 69
5.2.3.1 验证 YANG 文件 70
5.2.4 编译原子驱动包 70
5.2.5 开发 CLI SND 示例 71
5.2.5.1 开发 CX600 CLI SND 示例 71
5.2.5.2 开发 NE5000E CLI SND 示例 78
5.2.5.3 开发 SG6000 CLI SND 示例 89
5.3 开发 NETCONF&CLI 双协议设备原子驱动包 98
5.3.1 创建原子驱动包模板 98
5.3.2 开发原子驱动包 99
5.3.2.1 编辑软件包配置文件 99
5.3.2.2 编写设备 YANG 模型 102
5.3.2.3 开发代码 104
5.3.2.3.1 引入必要的头文件 104
5.3.2.3.2 编写设备纳管定制 105
5.3.2.3.3 编写设备类型能力定制 109

 版权所有 © 华为技术有限公司

5.3.2.3.3.1 配置设备驱动数据 109
5.3.2.3.3.2 设置业务自定义处理（可选） 123
5.3.2.3.3.3 设置数据前置处理（可选） 123
5.3.2.3.3.4 设置数据后置处理（可选） 123
5.3.2.3.3.5 设置数据联动（可选） 124
5.3.2.3.3.6 获取 syncId（可选） 124
5.3.2.3.4 编写设备特性能力定制（可选） 125
5.3.3 验证原子驱动包 127
5.3.3.1 验证 YANG 文件 127
5.3.4 编译原子驱动包 127
5.3.5 开发 NETCONF&CLI SND 示例 128
5.3.5.1 开发 CE5855 NETCONF&CLI SND 示例 128
6 开发 GND 133
6.1 开发 GND 业务包 133
6.1.1 创建 GND 模板 133
6.1.2 开发 GND 134
6.1.2.1 编辑软件包配置文件 134
6.1.2.2 开发业务 YANG 模型 137
6.1.2.3 开发业务包代码 143
6.1.2.4 开发 Jinja2 模板 146
6.1.3 验证业务包 147
6.1.3.1 验证 YANG 文件 147
6.1.3.2 执行单元测试 148
6.1.4 编译业务包 150
6.2 开发 GND 示例 150
6.2.1 开发 AAAmini GND 示例 150
6.2.2 开发 Hillstone FireWall GND 示例 157
6.2.3 开发 Hillstone FireWall 业务还原 GND 示例 164
7 开发 SSP 173
7.1 开发网络业务插件包 173
7.1.1 创建业务包模板 173
7.1.2 开发业务包 174
7.1.2.1 编辑软件包配置文件 174
7.1.2.2 开发业务 YANG 模型 177
7.1.2.3 开发业务包代码 182
7.1.2.4 开发 Jinja2 模板 187
7.1.3 验证业务包 191
7.1.3.1 验证 YANG 文件 191
7.1.3.2 执行单元测试 191
7.1.4 编译业务包 193
7.2 开发 RPC 业务插件包 193
7.2.1 创建业务包模板 194

 版权所有 © 华为技术有限公司

7.2.2 开发业务包 194
7.2.2.1 编辑软件包配置文件 194
7.2.2.2 开发业务 YANG 模型 197
7.2.2.3 开发业务包代码 200
7.2.2.4 开发 Jinja2 模板 202
7.2.3 验证业务包 202
7.2.3.1 验证 YANG 文件 202
7.2.3.2 执行单元测试 203
7.2.4 编译业务包 203
7.3 开发 SSP 示例 204
7.3.1 开发 L3VPN SSP 示例 204
7.3.2 开发 L2EVPN SSP 示例 209
7.3.3 开发 L3EVPN SSP 示例 218
7.3.4 开发业务还原 SSP 示例 234
7.3.5 开发查询设备信息 RPC SSP 示例 240
8 开发告警插件包 244
8.1 创建业务包模板 244
8.2 开发业务包 245
8.2.1 编辑软件包配置文件 245
8.2.2 开发业务包代码 248
8.3 编译业务包 252
8.4 验证告警插件包 253
9 开发迁移插件包 254
9.1 创建迁移包模板 254
9.2 开发业务包 255
9.2.1 编辑软件包配置文件 255
9.2.2 开发业务包代码 258
9.2.2.1 判断不兼容类型和处理规则 258
9.2.2.2 编写不兼容配置规则和代码 262
9.3 编译业务包 265
10 导入并安装软件包 267
10.1 登录系统 267
10.2 导入并激活软件包 270
11 软件包升级说明 274
12 开放可编程开发规范 317
12.1 YANG 开发规范 317
12.1.1 YANG 文件的基本内容和布局 317
12.1.2 YANG 模型文件命名/模块命名统一规定 318
12.1.2.1 规则：YANG 模型文件名称的合法字集为：小写字母（a-z）、数字（0-9）和连字符（-），且最大长度不能超过 63 个字符（不含版本号和.yang） 318

 版权所有 © 华为技术有限公司

12.1.2.2 规则：YANG 模块名必须与 YANG 文件名称一致（不含版本号和.yang），YANG 模块名要求全系统唯一，最大长度不能超过 63 个字符.....319
12.1.3 YANG 模型文件 namespace 统一规定.....319
12.1.3.1 规则：YANG 模型文件的 namespace 规则为 urn:XXX:yang:<module-name>.....319
12.1.4 YANG 模型文件的元素名称定义统一规定.....319
12.1.4.1 规则：YANG 模型文件的元素命名合法字符集：小写字母（a-z）、数字（0-9）和连字符（-），且最大长度不能超过 63 个字符。其中首字符必须以小写字母开头.....319
12.1.4.2 规则：YANG 模型文件中的枚举项 enum/identity 标识符合法字符集：字母（优先使用小写字母 a-z，如有专有术语、缩略语等不适合用小写字母的可使用大写字母 A-Z）、数字（0-9）、小数点（.）和连字符（-），且最大长度不能超过 63 个字符。其中 enum 的首字符必须以字母或数字开头，identity 的首字符必须以字母开头.....320
12.1.4.3 规则：Deviations YANG 文件中如果需要使用 typedef 对数据模型中的类型做重定义，名称必须为原类型名称加”-deviations”后缀.....321
12.1.4.4 规则：各元素命名风格要统一，同一系统中的相同概念的命名保持相同的名称.....321
12.1.5 YANG 文件排版和注释的统一规定.....321
12.1.5.1 规则： YANG 文件排版和注释要风格统一，保证良好的视觉分级效果.....321
12.1.6 模型文件划分原则.....322
12.1.6.1 原则：模型的划分根据系统功能特性为基础，根据实际业务逻辑必需的数据来定义模型的要素.....322
12.1.6.2 原则：在内置类型，IETF 公共类型无法满足要求的前提下，可以在全系统定义一个公共的数据类型文件，其它 YANG 文件按需引用.....322
12.1.6.3 原则：按照业界 YANG 模型的定义习惯，类型与操作可以与数据模型定义到同一个 YANG 文件中.322
12.1.6.4 原则：YANG 模块划分要高内聚、低耦合，符合业务聚合特征，且必须符合架构分层依赖原则。不能出现循环依赖的场景.....322
12.1.7 模型文件内部各部分内容的规定及设计规则.....322
12.1.7.1 规则：list 的记录对象，必须按照 YANG 语法来定义 key.....323
12.1.7.2 规则：list 对象外要单独定义一层 container，container 的名字为 list 的名字后加 s 或 es.....323
12.1.7.3 规则：除 key 值节点以外，在创建父节点时必须携带的字段，要通过 mandatory 来指定.....324
12.1.7.4 规则：在所有类型的 YANG 文件中，必须用 config 为只读状态的 element 定义访问权限.....324
12.1.7.5 规则：如果一个或者多个 element 的组合在所有记录中取值是唯一的，则在 YANG 文件中，必须用 unique 定义其唯一性属性.....324
12.1.7.6 规则：一个对象属性 element 的取值，如果有明确的度量单位，则要使用 units 指定单位信息.....326
12.1.7.7 规则：对于有明确默认值的配置对象属性，在 YANG 文件中必须要使用 default 属性为对应的 element 指定默认值.....326
12.1.7.8 规则：对于字符串类型，必须对长度做限制，可以使用 length 显式指定长度范围，也可以使用 pattern 隐式约束长度；有格式或字符输入限制时，必须使用 pattern 约束.....327
12.1.7.9 规则：对于有明确取值范围的数值类型 element，在 YANG 文件中必须使用 range 属性指定对应的 element 的取值范围，且最大最小值要简洁，可以使用 min 或 max 代替该数值类型的最小或最大值.....327
12.1.7.10 规则：grouping 中的节点引用 grouping 外的节点要避免使用相对路径.....328
12.1.7.11 规则：定义 rpc 时，不必在 output 中定义返回错误码和错误提示.....328
12.1.7.12 规则：对于有确定最大/最小条目数限制的 list 或 leaf-list，在 YANG 文件中必须要使用 max-elements/min-elements 方式指定条目数量规格.....328
12.1.7.13 规则：废弃节点必须用 status 属性明确标识为 obsolete，不建议使用的节点必须用 status 属性明确标识为 deprecated，废弃节点或不建议使用的节点如果有替代节点，必须在描述中说明替代节点是什么.....328
12.1.7.14 规则：YANG 文件中 import 和 include 语句中，禁止包含 revision-date 语句.....329
12.1.7.15 规则：文件中全部内容（包括标点符号）必须使用英文，不允许使用中文或其他语言。.....329
12.1.7.16 规则：版权信息必须按照统一要求定义。.....329

12.1.7.17 规则：organization、contact、revision 必须按照统一要求定义。.....330
12.1.7.18 规则：YANG 文件及其交互信息中禁止出现公司、部门、项目的标识及缩写，禁止出现芯片名称、型号。（文件名/模块名出现 huawei 字样除外）.....331
12.1.7.19 规则：YANG 文件要通过 pyang 工具的规范检查校验。.....331
12.1.7.20 规则：YANG 文件中 pattern 必须采用 XSD 类型正则表达式，隐式包含“以^开头和以$结尾”.....331
12.1.7.21 规则：如下节点不能作为 list 的 key：描述信息，密码，大小写不敏感的节点，数据可能拆分合并的复杂类型节点等.....332
12.1.7.22 规则：YANG 文件中节点的 description 要清晰、准确、完整、规范的表述该节点的具体含义.....332
12.1.7.23 规则：对于密码类型的处理原则.....333
12.1.7.24 规则：presence container 的使用原则.....333
12.2 Python 开发规范.....334
12.2.1 排版.....334
12.2.1.1 规则：程序块采用 4 个空格缩进风格编写.....334
12.2.1.2 规则：Python 文件必须使用 UTF-8 编码.....334
12.2.1.3 规则：一行只写一条语句.....334
12.2.1.4 规则：相对独立的程序块之间、变量说明之后必须加空行.....334
12.2.1.5 规则：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符前后要加空格.....335
12.2.1.6 规则：加载模块必须分开每个模块占一行.....335
12.2.1.7 规则：导入部分(imports)置于模块注释和文档字符串 之后，模块全局变量和常量声明之前.....335
12.2.2 注释.....335
12.2.2.1 规则：类和接口的文档字符串写在类声明(class ClassName:)所在行的下一行，并向后缩进 4 个空格.....336
12.2.2.2 规则：公共函数的文档字符串写在函数声明(def FunctionName(self):)所在行的下一行，并向后缩进 4 个空格.....336
12.2.2.3 规则：公共属性的注释写在属性声明的上方，与声明保 持同样的缩进。行内注释应以#和一个空格作为开始，与后面的文字注释以一个空格隔开.....336
12.2.2.4 规则：模块文档字符串写在文件的顶部，导入(import) 部分之前的位置，不需要缩进.....336
12.2.2.5 规则：文档字符串多于一行时，末尾的"""要自成一行.....336
12.2.2.6 规则：注释必须与其描述的代码保持同样的缩进，并放在其上方相邻位置.....337
12.2.3 命名.....337
12.2.3.1 规则：包（Package）、模块（Module）名使用意义完 整的英文描述，采用小写加下划线（lower_with_under）的风格命名.....337
12.2.3.2 规则：类（Class）名使用意义完整的英文描述，采用大 写字母开头的单词（CapWords）风格命名.....337
12.2.3.3 规则：函数（Function）、方法（Method）、函数参数（Function Parameters）名使用意义完整的英文描述，采用小写加下划线（lower_with_under）的风格命名.....337
12.2.3.4 规则：变量（variable）采用小写加下划线 （lower_with_under）的风格命名。常量（constant） 采用大写加下划线（CAPS_WITH_UNDER）的风格命名.....337
12.2.3.5 规则： 类或对象的私有成员一般用单下划线_开头；对于 需要被继承的基类成员，如果想要防止与派生类成员重名，可用双下划线__开头.....338
12.2.4 编码.....338
12.2.4.1 规则：避免在无关的变量或无关的概念之间重用名字， 避免因重名而导致的意外赋值和错误引用.338
12.2.4.2 规则：类的方法不需访问实例时，根据具体场景选择使 用@staticmethod 或者@classmethod 进行装饰.....339
12.2.5 异常处理.....339

 版权所有 © 华为技术有限公司

12.2.5.1 规则：使用 try…except…结构对代码作保护时，需要在异常后使用 finally…结构保证操作对象的释放 339
12.2.5.2 规则：不要使用“except:”语句来捕获所有异常 339
12.2.5.3 规则：不在 except 分支里面的 raise 都必须带异常 339
12.2.6 并发与并行 339
12.2.6.1 规则：多线程适用于阻塞式 IO 场景，不适用于并行计算场景 339
12.3 其他注意事项 339
12.3.1 devicemgr.query_neid(nename)不推荐使用，推荐使用 devicemgr.query_neids_by_operatename(operatename) 339
12.3.2 devicemgr.query_status(neid='', nename='', nemanageip='')不推荐使用，推荐使用 devicemgr.query_status_by_neid(neid) 340
13 附录 341
13.1 业务安全 341
13.2 CLI 扩展项 342
13.2.1 基础扩展项简表 342
13.2.2 基础扩展项详细介绍 350
13.2.2.1 insert-before(after) "xxx" leaf 350
13.2.2.1.1 使用场景举例 351
13.2.2.2 render-view true|false container/list 355
13.2.2.2.1 使用场景举例 356
13.2.2.3 render-name true|false container/leaf/leaf-list 358
13.2.2.3.1 使用场景举例 358
13.2.2.4 optional-space list/leaf 360
13.2.2.4.1 使用场景举例 360
13.2.2.5 render-undo-value true|false leaf 362
13.2.2.5.1 使用场景举例 362
13.2.2.6 render-undo leaf 363
13.2.2.6.1 使用场景举例 363
13.2.2.7 render-undo-before container/list/leaf/leaf-list 364
13.2.2.7.1 使用场景举例 364
13.2.2.8 render-nested-delete true|false container/list 365
13.2.2.8.1 使用场景举例 366
13.2.2.9 quit-view "xxx" container/list 367
13.2.2.9.1 使用场景举例 368
13.2.2.10 allow-multi-token xx leaf 369
13.2.2.10.1 使用场景举例 369
13.2.2.11 update-by-delete leaf/leaf-list/list 371
13.2.2.11.1 使用场景举例 372
13.2.2.12 render-single-line container 373
13.2.2.12.1 使用场景举例 373
13.2.2.13 render-multi-line leaf-list/list 374
13.2.2.13.1 使用场景举例 375
13.2.2.14 render-sequential-order container 377

 版权所有 © 华为技术有限公司

13.2.2.14.1 使用场景举例 377
13.2.2.15 render-optional "xxx" container 378
13.2.2.15.1 使用场景举例 379
13.2.2.16 render-sequential-order-until "name" container 380
13.2.2.16.1 使用场景举例 381
13.2.2.17 render-suffix-on-create "xxx" list 382
13.2.2.17.1 使用场景举例 383
13.2.2.18 replace-value "xxx,yyy" leaf 383
13.2.2.18.1 使用场景举例 384
13.2.2.19 delete-siblings-on-delete container/list/leaf/leaf-list 384
13.2.2.19.1 使用场景举例 385
13.2.2.20 delete-parents-on-delete leaf 385
13.2.2.20.1 使用场景举例 385
13.2.2.21 render-undo-name true|false container/list/leaf 385
13.2.2.21.1 使用场景举例 386
13.2.2.22 delete-by-set-default-value leaf 388
13.2.2.22.1 使用场景举例 389
13.2.2.23 delete-no-support container/leaf 389
13.2.2.23.1 使用场景举例 390
13.2.2.24 leaf-dependency "xxx,xxx" container/list 390
13.2.2.24.1 使用场景举例 391
13.2.2.25 render-flat-container container 391
13.2.2.25.1 使用场景举例 392
13.2.2.26 render-container-together container 393
13.2.2.26.1 使用场景举例 393
13.2.2.27 render-identifier container/list/leaf/leaf-list 394
13.2.2.27.1 使用场景举例 394
13.2.2.28 render-undo-name-and-value true|false leaf 395
13.2.2.28.1 使用场景举例 396
13.2.2.29 render-order-dependency container/leaf 396
13.2.2.29.1 使用场景举例 397
13.2.2.30 delete-on-empty container 398
13.2.2.30.1 使用场景举例 399
13.2.2.31 reset-all-siblings container 399
13.2.2.31.1 使用场景举例 400
13.2.2.32 reset-siblings container 401
13.2.2.32.1 使用场景举例 401
13.2.2.33 cli-case-sensitive/insensitive container/list/leaf/leaf-list 402
13.2.2.33.1 使用场景举例 403
13.2.2.34 render-hide-in-view leaf 403
13.2.2.34.1 使用场景举例 404
13.2.2.35 reset-container container/list/leaf 405

 版权所有 © 华为技术有限公司

13.2.2.35.1 使用场景举例 406
13.2.2.36 render-batch-list-syntax list/leaf-list 406
13.2.2.36.1 使用场景举例 407
13.2.2.37 incomplete-command container/leaf 408
13.2.2.37.1 使用场景举例 409
13.2.2.38 incomplete-no leaf 410
13.2.2.38.1 使用场景举例 411
13.2.2.39 render-order-by-user list 412
13.2.2.39.1 使用场景举例 413
13.2.3 CLI 自定义扩展 414
13.2.3.1 背景 414
13.2.3.2 扩展 414
13.2.3.2.1 cli-custom-transform 415
13.2.3.2.2 cli-custom-read 416
13.2.3.2.3 cli-custom-processor 419
13.2.3.2.4 cli-custom-rpc 421
13.3 Jinja2 模板过滤器及方法 422
13.3.1 jinja 模板过滤器 422
13.3.2 jinja 模板支持 no_delete 标签定制删除 425
13.3.3 jinja 模板支持 create/merge/delete/replace 操作 426
13.3.4 jinja 模板支持 order 调整顺序 427

 版权所有 © 华为技术有限公司

1 预备知识

本节介绍进行开放可编程开发需要具备的基础知识，包括NETCONF、YANG、Jinja2、Python和软件包结构等。

1.1 了解编程语言和协议

1.2 了解软件包结构

1.3 了解日志记录

1.1 了解编程语言和协议

在进行开放可编程开发前，需要掌握下表所述编程语言和协议。

表 1-1 开放可编程编程语言和协议

知识项	简介	学习链接
NETCONF	NETCONF（NETwork CONFiguration）协议是一种网络设备管理协议，类似SNMP，提供一套新增、修改、删除网络设备配置，查询配置、状态和统计信息的框架机制。NETCONF是由IETF定义的用于在网元上执行安装、维护和删除配置数据的协议。所有的NETCONF操作都是通过基于XML编码的RPC层实现的。开放可编程主要使用NETCONF与网元通信。	**https://tools.ietf.org/html/rfc6241**
YANG模型	YANG（Yet Another Next Generation）是一种数据建模语言，提供了标准化的方式对网元的配置和状态数据进行建模。开放可编程使用YANG对业务和网元进行建模，并为实现从业务模型到网元模型的映射提供了相应的机制。	**https://tools.ietf.org/html/rfc7950**
Jinja2	Jinja2是一款现代化的、设计友好的Python模板引擎。开放可编程使用Jinja2快速完成业务报文的模板化处理。	**http://jinja.palletsprojects.com/en/2.10.x/**

版权所有 © 华为技术有限公司

知识项	简介	学习链接
Python	Python是一门面向对象的高阶编程语言，因其语法简便、可读性强、适用范围广、学习曲线平滑，开放可编程吸纳Python做为主要的软件包开发语言。	https://docs.python.org/3/tutorial/index.html

1.2 了解软件包结构

从系统中导出的软件包结构统一如下。

- 支持Python语言软件包。

  当Python语言的软件包激活时，包管理告知Python运行容器加载软件包；当Python语言的软件包卸载时，包管理告知Python运行容器卸载软件包。

Python开发语言

package.zip

|---package -------文件夹名字，和压缩包同名。

| |---pkg.json 【必填项】包描述文件

| |---bin 编译命令目录

| |---python Python开发的代码目录，映射类型含有Python时才有此目录

| |--- 用户开发的Python

| |---resources 资源存放目录，映射类型含有Python才有此目录

| |---template 模板存放目录，SSP包才有此目录

| |---test 单元测试文件存放目录，SSP包才有此目录

| |---yang YANG模型存放目录

1.3 了解日志记录

1.3.1 查看日志

- AOCmini运行日志路径：AOCmini\log\NCE\AOCService\AOCService-0-0\log\aocservice.log

  日志格式：时间 日志级别 [线程号][启动用户][运行类 行号]日志内容
- tomcat启动日志路径：AOCmini\log\NCE\AOCService\AOCService-0-0\tomcatlog\catalina.log

  日志格式：时间 日志级别 [线程号] 运行类 日志内容
- Python运行日志路径：AOCmini\log\NCE\ExtendedPkgRTService

  日志格式：时间(线程号) [日志级别][运行类 行号]日志内容

版权所有 © 华为技术有限公司

1.3.2 设置日志级别

支持用户自定义设置Python运行日志级别。

日志配置文件 logger.conf 介绍

logger.conf内容：

[logger_user]

level=INFO

说明

level值支持：DEBUG、INFO、WARN、ERROR，不区分大小写。支持动态设置日志级别，不需要重启微服务。

日志配置文件路径

AOCmini\envs\Product-ExtendedPkgRTService\venv-{$group_name}\python\{$package_name}\resources

说明

{$group_name}为用户开发的三方包所在的group名。

{$package_name}为用户开发的三方包名。

版权所有 © 华为技术有限公司

2 开发流程

软件包开发流程如下图所示。

图 2-1 软件包开发流程

 版权所有 © 华为技术有限公司

3 开发准备

本节介绍开发软件包前所需的配套资源。

编号	名称	用途说明	获取方式
1	aoc_api-*.*.*-py3-noneany.whl	开放可编程SDK是由华为提供的一套基于Python的编程接口，里面包含需要继承的父类NcsService以及所有可能调用到的接口。	登录数通网络开放可编程社区：**https://devzone.huawei.com/cn/enterprise/aoc/resDownload.html**，下载SDK文件。
2	yang-offline-util.zip	开放可编程YANG模型离线验证工具。	登录数通网络开放可编程社区：**https://devzone.huawei.com/cn/enterprise/aoc/resDownload.html**，下载验证工具。
3	开放可编程典型场景样例包	提供典型场景代码样例包，方便用户定制。	登录数通网络开放可编程社区：**https://devzone.huawei.com/cn/enterprise/aoc/resDownload.html**，获取对应的代码样例包。
4	开放可编程mini版本软件包	提供开放可编程轻量化的软件包，用户可以在下载安装后在本地电脑执行设备和业务相关配置操作。	登录数通网络开放可编程社区：**https://devzone.huawei.com/cn/enterprise/aoc/resDownload.html**，获取软件包。

版权所有 © 华为技术有限公司

4 开发环境部署

开发环境的部署操作以Windows 10系统为例进行介绍。

4.1 安装Python解析器

4.2 安装开放可编程SDK

4.3 安装集成开发环境

4.4 安装Gpg4Win及生成签名用密钥

4.5 安装开放可编程mini软件包

4.1 安装 Python 解析器

下载并安装

您可以访问**Python官网**，然后单击“Download Python 3.x.x”，下载并进行安装。安装时注意要勾选“Add Python 3.x to PATH”，将路径自动加入环境变量中。推荐安装Python 3.8.2，其他版本没有经过华为充分测试，使用其它版本编译的软件包可能会激活失败。

验证

在安装成功后，可以按下述方式进行安装后验证。

步骤1 单击开始菜单。并选择“Windows系统 > 命令提示符”打开“命令提示符”。

步骤2 在当前提示符下输入“python”，回显应如下图所示。

```
C:\Users\demo>python
Python 3.8.2 (tags/v3.8.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

----结束

手动配置环境变量

安装时未勾选“Add Python 3.x to PATH”，没有将路径自动加入环境变量中，则可以按照下面步骤手动配置环境变量。

 版权所有 © 华为技术有限公司

步骤1 在“此电脑”上单击右键菜单“属性”，打开“系统”界面。

步骤2 在导航栏单击“高级系统设置”，打开“系统属性”界面。

步骤3 在“高级”页签页面，单击“环境变量”，打开“环境变量”界面。

步骤4 在用户变量区域，双击“Path”变量，弹出“编辑环境变量”界面。

步骤5 单击“新建”按钮，输入python安装路径或通过单击“浏览”选择python安装路径。

步骤6 单击“新建”按钮，输入python安装路径\Scripts或通过单击“浏览”选择python安装路径下的Scripts目录。

步骤7 单击“确定”。

步骤8 按照验证步骤能正常回显，则说明环境变量配置成功。

----结束

4.2 安装开放可编程 SDK

为了能够离线验证编写的Python逻辑是否符合预期，需要安装**SDK**和一些依赖的三方包。开放可编程SDK是由华为提供的一套基于Python的编程接口，里面包含需要继承的父类NcsService以及所有可能调用到的接口。安装SDK的过程会自动下载并安装依赖的三方包，需要开发者提前配置好本地的pip镜像源，安装成功之后服务包所依赖的所有三方包都安装到了本地。

说明

1. 开放可编程SDK没有开源计划，只支持从本地运行.whl安装包的方式安装。
2. easymap算法会自动计算差异，如果老配置是关键配置，可以通过nodelete标签功能防止自动删除

步骤1 将获得的SDK安装包“aoc_api-x.y.z-py3-none-any.whl”复制到当前用户目录下。

步骤2 单击开始菜单。并选择“Windows系统 > 命令提示符”打开“命令提示符”。

步骤3 在当前提示符下运行“pip install aoc_api-x.y.z-py3-none-any.whl”，应出现类似下图的回显。此时，开放可编程SDK安装成功。

```
C:\Users\demo>pip install aoc_api-2.0.0-py3-none-any.whl
...
...
Successfully installed aoc-api-2.0.0
You are using pip version 18.1, however version 20.2.2 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

----结束

4.3 安装集成开发环境

下载并安装

您可以访问**PyCharm官网**或者**IntelliJ IDEA官网**，然后单击下载安装适用于用户操作系统的社区开发版本。

 版权所有 © 华为技术有限公司

配置 Python 解析器

在完成PyCharm或者IntelliJ IDEA应用程序的安装后，还需要配置Python解析器才能让PyCharm或者IntelliJ IDEA正常工作。如下以PyCharm程序为例，说明如何配置Python解析器。

步骤1 在开始菜单中，单击“JetBrains PyCharm Community Edition”运行PyCharm社区开发版。

步骤2 在初始设置中使用默认配置，并在“Customize PyCharm”对话框中单击“Skip Remaining and Set Defaults”。

步骤3 在“Welcome to PyCharm”窗口中，从右下角的“Configure”下拉列表中选择“Settings”。

步骤4 在“Settings for New Projects”中，单击“Project Interpreter”。然后单击“Project Interpreter”下拉菜单右侧的设置按钮并选择“Add...”。然后选择“System Interpreter”，并在“Select Python Interpreter”中选择Python 3.x的安装路径。

步骤5 单击“OK”，关闭对话框。此时“Settings for New Projects”对话框中的“Project Interpreter”列表中列出了当前安装的Python解析器版本和路径以及pip和setuptools扩展包的版本信息。

配置完成后，PyCharm即可正常使用。

----结束

4.4 安装 Gpg4Win 及生成签名用密钥

Gpg4Win（GNU Privacy Guard for Windows）是一款免费开源的GNU工具，可以对Windows操作系统下的OpenPGP签名进行检验。开放可编程使用该软件对软件包进行签名。

下载安装

您可以访问**Gpg4Win官网**，下载并安装最新版的Gpg4Win。

生成签名文件

完成安装后，使用Kleopatra工具生成OpenPGP签名文件，步骤如下：

步骤1 在开始菜单中，单击“Kleopatra”运行签名工具。

步骤2 在主菜单中选择“文件 > 新建密钥对...”，运行“密钥创建向导”。此时，单击“创建个人OpenPGP密钥对”。

步骤3 在“输入细节”中输入“名字”和“电子邮件”。

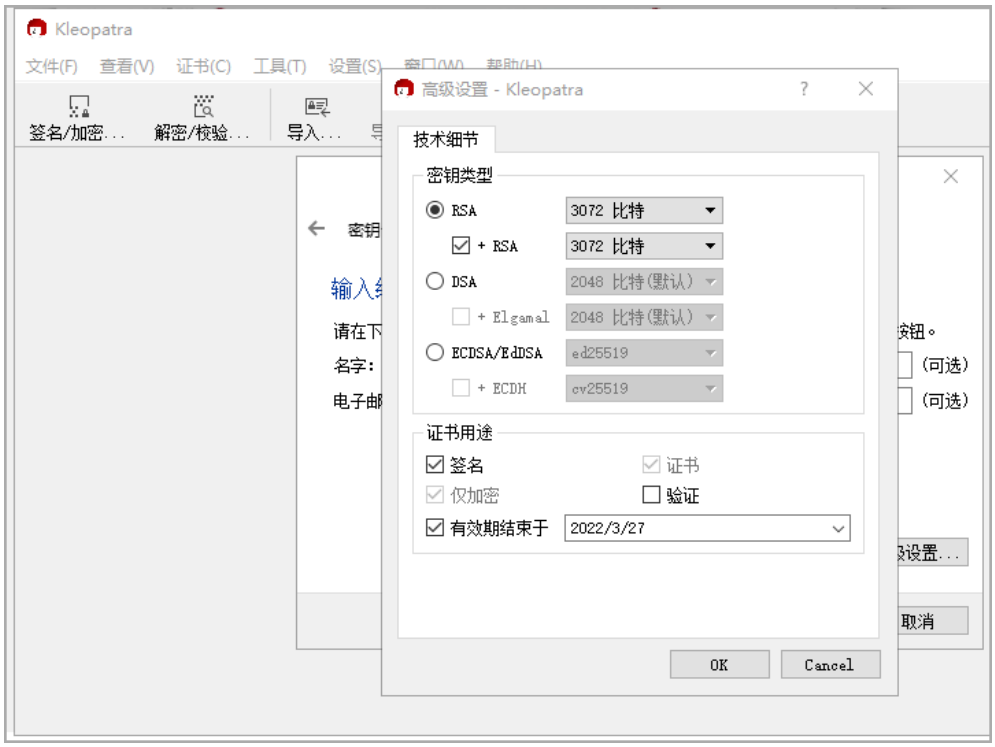
步骤4 单击“高级设置”，在“密钥类型区域”，选择“RSA”，并将密钥长度设置为“3072比特”。单击“OK”关闭当前窗口。然后单击“下一步”。

说明

在“证书用途”区域，务必不要勾选“验证”。

版权所有 © 华为技术有限公司

图 4-1 高级设置



步骤5 在“检查参数”窗口中，勾选“显示全部细节”。确认参数设置无误后，单击“新建”。

步骤6 在密钥创建过程中，需在弹出窗口中创建并记录密码。然后单击“OK”。该密码在编译软件包时会用到，请务必记住该密码。

步骤7 在“密钥对已成功创建”窗口中，单击“生成您的密钥对的副本”将私钥文件保存到安全的位置。

私钥文件建议命名为“privkey.asc”。在保存私钥的过程中需要输入上一步设置的密码。

步骤8 单击“完成”关闭“密钥创建向导”，然后在证书列表中找到已创建密钥对。单击右键，然后在快捷菜单中选择“导出”以导出公钥文件。公钥文件在导入软件包到系统中时需要上传。

----结束

4.5 安装开放可编程 mini 软件包

安装**开放可编程mini软件包**及启动服务步骤如下所述：

步骤1 将获取到的开放可编程mini软件包移动到不含中文的路径下，建议路径层级不要太深。

步骤2 解压软件包到当前路径下。

步骤3 双击start.bat即可启动开放可编程mini服务，整个启动过程约需3~5分钟。启动后不能关闭命令行窗口，当停止服务时关闭窗口。

 版权所有 © 华为技术有限公司

步骤4 在浏览器地址栏中输入“**https://127.0.0.1:32018/aocwebsite**”，按Enter进入开放可编程功能页面，即说明服务启动成功。

----结束

 版权所有 © 华为技术有限公司

5 开发 SND

用户可以开发NETCONF、CLI、RESTCONF等协议类型的SND包，为设备快速定制驱动。

支持开发的SND包的类型包括以下几种：

- NETCONF SND：提供YANG到NETCONF的通用驱动，用户可以开发NETCONF SND包快速定制驱动。
- CLI SND：提供YANG到CLI的通用驱动，用户可以开发CLI SND包快速定制驱动。
- NETCONF&CLI SND：提供NETCONF和CLI双协议驱动，用户可以开发双协议SND包快速定制双协议驱动。双协议中的CLI驱动，作为北向CLI命令行透传使用。命令行由第三方APP生成，由北向接口，利用CLI协议通道，透传命令行至设备。
- RESTCONF SND：提供YANG到RESTCONF 协议的通用驱动，用户可以开发RESTCONF协议驱动。

5.1 开发NETCONF协议设备原子驱动包

5.2 开发CLI协议设备原子驱动包

5.3 开发NETCONF&CLI双协议设备原子驱动包

5.1 开发 NETCONF 协议设备原子驱动包

本章节介绍NETCONF协议设备原子驱动包的开发过程。

用户在开发SND软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

5.1.1 创建原子驱动包模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据**表5-1**设置软件包的相关属性。

 版权所有 © 华为技术有限公司

表 5-1 软件包属性

属性名称	属性值
名称	NE_SND
版本	1.0.0
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	NetEngine
协议类型	NETCONF

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

5.1.2 开发原子驱动包

本节详细介绍开发SND包全过程。

5.1.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 请按照参数表修改对应参数的值。

表 5-2 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：ssp（Specific Service plugin）、gnd（Generic NE driver）、snd（Specific NE driver）、mp（SND Migrate plugin）。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。仅支持该字段为“huawei”（不区分大小写）的SND和GND包。	必选
augment-isolation	模型范围标识。	布尔值，只能为 true 或 false。DCN单站场景需要配置为true，其他场景为false。	可选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。 version（必选）字段：版本号三段式规则。	可选
snd-type	snd的类型。	值为“system”，表示为系统内置的snd包。系统内置的snd包，该参数值不建议修改。 该参数未设置时，默认值为“specific”，表示用户自定义包。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
snd-id	snd包才需要设置的值。id信息区分不同snd包，不同名的包不能使用相同的snd-id。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。 snd包时，必须设置，其他类型软件包时，不需要设置。	可选
devices	snd, gnd包特有值，驱动匹配的设备信息。	如果配置，则必须包含： ● 驱动设备厂商vendor（必选），不能超过 128个字符。 ● 具体设备类型device-type（必选），不能超过 128个字符。 ● 具体设备的版本device-version（必选），不能超过 128个字符。	可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)，不能超过128个字符。SND包设置为"snd"。 ● key（必选），回调key值，不能超过128个字符。SND包一般只需配置“ecs-driver”，指定在SND包中具体实现类；用户不使用系统默认连接agent时，再补充配置“ecs-connect-agent”，自定义hook中配置的连接agent，要生效需要同时设备的连接能力参数connectPolicy 配置为“COMPATIBLE_CONNECT”。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“device-version”、“python-class-name”，参数含义和修改方法请参见上述表格。编辑文件示例如下：

```
{
    "name": "NE_SND",
```

 版权所有 © 华为技术有限公司

```
  "version": "1.0.0",
  "description": "NE_SND",
  "package-type": "snd",
  "producer": "HUAWEI",
  "nce-min-versions": [
     "1.0.0"
  ],
  "snd-id": "NE_SND",
  "devices": [
   {
     "vendor": "HUAWEI",
     "device-type": "NetEngine",
     "device-version": "V800R012C00SPC300"
   }
  ],
  "hooks": [
   {
     "type": "snd",
     "key": "ecs-driver",
     "python-class-name": "com.huawei.controller.devicetype.nesnd.NE"
   }
  ]
}
```

----结束

5.1.2.2 编写设备 YANG 模型

SND包中的YANG不参与编译，用于描述设备的配置能力、设备的北向接口以及生成南向报文。用户根据SND包类型，获取或编写对应的YANG模型，替换掉软件包中默认的YANG模型文件。

本示例中，请用户将SND模板中默认YANG文件删除，将从设备商获取设备的YANG模型文件放置到yang文件夹下。

如果用户需要使用自定义数据加密处理、高危操作定义、YANG模块黑名单屏蔽、YANG模型免校验、自定义模型权限等功能，请参考下文配置。

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

表 5-3 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d

版权所有 © 华为技术有限公司

节点名称	父节点	属性	说明
-	-	type	节点类型，取值范围： ● leaf ● leaf-list ● typedef –自定义类型 默认是leaf

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
    <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
    <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
    <path>/db-security-test:security-top/security-list/security-list-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
    <rpcs>
        <rpc>huawei-aaa:reactivateUser</rpc>
        <rpc>huawei-aaa:changeMyPassword</rpc>
        <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
    </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
      <name>huawei-cfg</name>
      <revision>2020-03-18</revision>
  </module>
  <module>
      <name>huawei-dhcp</name>
      <revision>2020-04-29</revision>
```

 版权所有 © 华为技术有限公司

```
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```


版权所有 © 华为技术有限公司

5.1.2.3 开发代码

开发Python代码，实现设备的纳管、设备的配置管理、定制差异发现及同步和对账能力。

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码。用户可以重命名py文件名称和路径，将名称改为有意义的名称，需要与pkg.json文件的python-class-name中py文件名称字段保持一致。

5.1.2.3.1 引入必要的头文件

在代码最前面，首先需要引入必要的头文件中的相关接口。

下文在重写各个方法时描述了需要引用的头文件接口，请用户根据SND包类型、业务逻辑，参考示例代码导入需要的接口。

import语句应当放在文件头部，置于模块说明之后，全局变量之前。

5.1.2.3.2 编写设备纳管定制

编写设备纳管定制，包含两部分：注册设备sysoid信息和配置设备的连接能力信息。

注册设备 sysoid 信息

用户通过getSysoidInfo方法设置设备的sysoid信息，用于纳管此款型设备。

该方法不是必须要重写。纳管时期望对设备的sysoid信息校验，则需要重写方法，并准确配置sysoid字段信息。不对设备的sysoid信息校验，则可以不重写方法，通过pkg.json文件里的三元组来匹配。

注册设备sysoid信息的参数如下表所示。

表 5-4 注册设备 sysoid 参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
sysoid	否	STRING	N/A	N/A	设备唯一标识。 sysoid 可以通过以下方式获取： • 命令行查询，如华为设备一般可以执行display system information查询 • mib browser工具查询 • 通过OID网站查找，**https://oidref.com**
deviceType	否	STRING	N/A	N/A	设备类型。 请根据具体设备，匹配对应的设备类型，常用取值ROUTER、SWITCH、FIREWALL。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
deviceModel	否	STRING	N/A	N/A	设备款型。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。
deviceVendor	否	STRING	N/A	N/A	设备厂商。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。

注册设备信息代码示例如下。

```
# 保留默认代码中导入NetconfSND、SysoidInfo接口，重写getSysoidInfo方法时需要使用。
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo


# 默认定义实现类代码可全部删除，按如下示例定义继承NetconfSND父类的子类
# NetconfSND是所有Netconf协议网元驱动包的父类，默认的配置已在父类中实现
class NE(NetconfSND):

    # 重写getSysoidInfo方法，注册设备sysoid值、设备类型信息、设备型号信息、设备厂商信息
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()

    # 设置设备的sysoid
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"

    # 设置设备的设备类型
    sysoidEntity.deviceType = "ROUTER"

    # 设置设备的设备款型
    sysoidEntity.deviceModel = "NetEngine"

    # 设置设备的设备厂商
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

配置设备的连接能力信息

NETCONF&CLI SND包由于支持双协议，需要同时配置主协议和辅助协议连接能力信息，其他类型的SND包只需要配置主协议连接能力信息。

设备连接能力信息常用参数如下表所示，详细的参数说明请参考《 插件包API参考 》。继承的父类不同，默认值则可能存在差异，以下表格中不做特殊说明，都是继承NetconfSND父类后的默认值。

版权所有 © 华为技术有限公司

表 5-5 设备连接能力常用参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
connectPolicy	否	ENUM	DEFAULT_CONNECT/COMPATIBLE_CONNECT	DEFAULT_CONNECT	连接策略 DEFAULT_CONNECT，由默认agent处理 COMPATIBLE_CONNECT，由compatible agent处理，由hook指定。如果hook中未指定，则需要在methodType、agentImpl、priority参数中配置具体实现类agent。
channelInfo.writeChannel	否	ENUM	SINGLE_CHANNEL/PROTECTED_MODE	PROTECTED_MODE	写通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护
channelInfo.readChannel	否	ENUM	SINGLE_CHANNEL/PROTECTED_MODE	SINGLE_CHANNEL	读通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护
channelInfo.is_read_share_write	否	bool	True/False	False	是否读写通道共享 True：读写通道共享。只需要配置写通道类型即可，另一种通道无需配置。 False：读写通道不共享。读写通道类型都需要配置。
protocolType	否	ENUM	netconf/cli/restful/restconf	netconf	设置SND包的协议类型
helloType	否	ENUM	defaultType/standardTyp/extendType	defaultType	报文类型 defaultType，建立schema对接的NETCONF通道，能力集置为空。 standardType，建立标准yang能力集的NETCONF通道，能力集置为空。 extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
extendEntity	否	ARRAY_STRING	N/A	N/A	能力集列表 若报文类型为extendType类型时，需要携带扩展能力集。
connectionPriority	否	ENUM	PRIMARY_CONNECTION/AUXILIARY_CONNECTION	PRIMARY_CONNECTION	连接调用优先级 PRIMARY_CONNECTION：单协议SND包、双协议SND包的主协议需配置为该值。 AUXILIARY_CONNECTION：双协议SND包时，辅助协议需配置为该值。
methodType	否	STRING	Java	N/A	指定实现类agent的编码语言。
agentImpl	否	STRING	N/A	N/A	指定实现类agent。
priority	否	STRING	N/A	N/A	指定实现类agent连接调用优先级。

配置主协议设备连接能力信息代码示例如下。

```
# 导入ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity接口，重写getConnectInfo方法配置设备连接能力时使用，用于配置连接信息，如与设备的建连协议，建立几个通道，握手的hello报文等
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
# 导入SINGLE_CHANNEL接口，重写getConnectInfo方法配置设备连接通道时使用
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL


    # 重写getConnectInfo方法，配置NETCONF协议类型及连接能力信息，若不重写则按默认值配置
    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()

    # 所有类型的SND包，都需要配置主协议能力信息
    primaryConnectInfo = connectInfos.connectInfo.add()

    # 设置连接策略
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT

    """
    设置通道配置。NETCONF协议需要配置不同的通道类型，
    SINGLE_CHANNEL表示单链路无保护，
    当is_read_share_write为true表示读写通道共享，readChannel无需配置。
    """
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True

    # 设置主协议类型
```


版权所有 © 华为技术有限公司

```
primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

"""
设置报文类型：defaultType，建立schema对接的NETCONF通道，能力集置为空；
standardType，建立标准yang能力集的NETCONF通道，能力集置为空；
extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。
"""
primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType

# 设置连接调用优先级
primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
self.logger.info('getConnectInfo end.')
return connectInfos
```

5.1.2.3.3 编写设备类型能力定制

编写设备类型能力主要包括：配置设备驱动数据、业务自定义处理、数据前置后置处理、数据联动处理。

5.1.2.3.3.1 配置设备驱动数据

提供了两种方法，编写Python配置设备驱动数据，或者通过编写xml文件配置设备驱动数据，下面分别进行说明。

通过代码配置设备驱动信息

- **基于通用能力定制示例，定义与设备的交互策略**

  编写基于通用能力的交互策略。常用参数请参见下表，详细的参数说明请参考《 插件包API参考 》。

  表 5-6 设备通用驱动参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
checkSync	否	bool	true/false	false	在向设备下发业务数据过程中检查设备和控制器的数据是否一致。
unsupportedOperations	否	STRING	"create"/"delete"/"create,delete"	N/A	设备不支持的操作类型，自动进行转换。如将 "create,delete"转换为"merge,remove" 下发配置时可能会修改设备已有配置，请谨慎配置，以免引发业务安全风险。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
deleteStrategy	否	DeleteStrategy	PATH_ADD/CONTAINER_WITH_PRESENCE	CONTAINER_WITH_PRESENCE	对于delete、remove操作的报文，是否删除报文中的具体信息 PATH_ADD时，指定PATH的报文内容删除。 CONTAINER_WITH_PRESENCE时，下发报文不做内容删除。
syncToDel	否	N/A	syncToDel.key = "sync-to-del-enable" syncToDel.value 可以为 "true"或“false”	false	设置数据对账时是否支持删除南向设备配置实例。 当此参数设置为true时，会在数据对账时删除仅设备侧存在的配置数据，可能引发业务安全风险。建议将此参数设置为false。
pathNeedPreProcess	否	STRING	N/A	N/A	该字段填写需要进行报文前置处理的特性QName。若该字段填写了对应QName，对于设备返回的查询结果报文，会在处理之前先调用netconfPreprocess进行加工，加工之后再进行报文转换。
pathNeedPostProcess	否	STRING	N/A	N/A	该字段填写需要进行报文后置处理的特性QName。若该字段填写了对应QName，对于即将下发设备的报文，会在下发之前先调用netconfPostprocess进行加工，加工之后再下发设备。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
pathNeedEcsMapping	否	STRING	N/A	N/A	该字段填写需要进行业务报文定制处理的特性QName。若该字段填写了对应QName，对于下发设备的报文和设备返回的都统一调用toDocument、toDataObject进行转换。
pathNeedEcsRpcMapping	否	STRING	N/A	N/A	该字段填写需要进行rpc报文定制处理的rpc QName。若该字段填写了对应QName，对于下发设备的rpc报文和设备返回的都统一调用toRpcDocument、toRpcDataObject进行转换。
pathNeedEcsDBHook	否	STRING	N/A	N/A	下发配置到设备后，某些配置会联动生成其他配置，而控制器只会存储联动前的配置，会导致控制器和转发器配置不一致。因此，控制器端也需要在配置下发前进行联动处理。

通用能力定制代码示例如下。

```
# 导入CommonDriverInfo接口，重写getCommonDriverInfo方法编写设备类型时使用，基于通用能力定制，用于定制通用操作类型转换等
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo


    # 重写getCommonDriverInfo方法，配置设备通用驱动，不重写则按默认值配置
    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()

        # 在向设备下发业务数据过程中检查设备和控制器的数据是否一致。建议设置此参数为true，并实现getSyncId方法获取SyncId。
        common_driver.checkSync = True

    # 设备不支持的操作类型，自动进行转换，如将 "create,delete"转换为"merge,remove"。
    common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()

    # 定义key值，同步删除
    syncToDel.key = "sync-to-del-enable"

    # 设置数据对账时是否支持删除南向设备配置实例。取值：true（支持）。
    syncToDel.value = "true"
    return common_driver
```

- **基于协议类型定制示例，定义Netconf协议参数**

  支持NETCONF协议，需要定义NETCONF协议参数。常用参数请参见下表，详细的参数说明请参考《插件包API参考》。

 版权所有 © 华为技术有限公司

表 5-7 设备协议类参数

参数	是否必选	参数类型	参数值域	默认值	说明
classification	否	STRING	"huawei-v5"/"huawei-v8"/"cisco"	huawei-v5	设备原型。 huawei-v5:报文转换规则为match huawei-v8:报文转换规则为same cisco:报文转换规则为match
phase	否	STRING	"one"/"two"	one	设备支持的几阶段下发。
maxPkgLength	否	INT32	>0	N/A	设备支持的最大单个报文的大小。
netconfLock	否	bool	true/false	false	下发设备时是否需要先获取设备锁。若设备为juniper的设备，该字段须为true。
errorOption	否	STRING	"rollback-on-error"/"stop-on-error"/"continue-on-error"	"rollback-on-error"	设备配置数据出错之后的处理策略。
testOption	否	STRING	"set"	N/A	下发设备的报文是否需要加上<test-option>标签，当该字段为“set”时，下发报文会加上<test-option>set</test-option>。
netconfHelloEntity	否	NetconfHelloEntity	standardType/extendType/defaultType	defaultType	设备管理hello报文定制： standardType：标准报文 extendType：自定义报文 defaultType：默认报文（YANG对接）

版权所有 © 华为技术有限公司

参数	是否必选	参数类型	参数值域	默认值	说明
modelDiff	否	STRING	"same"/"match"	match	netconf报文转换类型。当为same时，下发的报文的namespace会统一转换为"https://www.huawei.com/netconf/vrp"。

协议类型定制代码示例如下。

```
# 导入NetconfDriverInfo接口，重写getNetconfDriverInfo方法编写设备类型时使用，基于Netconf协议类型能力定制，用于配置netconf协议参数
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo


    # 重写getNetconfDriverInfo方法，配置设备NETCONF驱动，不重写则按默认值配置
    def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()

    # 设置neconf下发阶段，一阶段还是二阶段。如果是支持二阶段的设备，设置此参数为two。
    netconf_driver.phase = "two"

    # 定义驱动设备原型类别，大部分华为设备是huawei-v5
    netconf_driver.classification = "huawei-v5"

    """
    定义报文转换类型。当为same时，下发报文为Schema标准报文；
    当为match时，下发报文为YANG标准报文
        netconf_driver.modelDiff = "match"
    """

    # 定义报文校验策略，当该字段为set时，直接根据报文处理不用校验。
    netconf_driver.testOption = "set"
    return netconf_driver
```

通过文件配置设备驱动信息

通过文件配置设备驱动信息，系统自动解析文件或用户自定义解析文件逻辑。用户按照要求命名和编写xml配置文件，不需要编码系统可自动解析xml文件配置设备驱动信息。

用户通过“resources/netconf-driver.xml”文件，可以定制设备驱动信息。

“resources/netconf-driver.xml”文件通用配置代码示例如下。具体参数请参考《插件包API参考》。

```
<?xml version="1.0" encoding="utf-8"?>
<netconf-driver xmlns="https://huawei.com/nce/netconf-driver">
 <netconfDriverInfo>
  <!-- 类型：string -->
  <classification>huawei-v8</classification>
  <!-- 类型：string -->
  <errorOption>one</errorOption>
  <!-- 类型：int32 -->
  <maxPkgLength></maxPkgLength>

  <!-- 类型：boolean -->
  <netconfLock>false</netconfLock>
  <!-- 类型：string -->
```


版权所有 © 华为技术有限公司

```
  <errorOption>stop-on-error</errorOption>
  <!-- 类型：string -->
  <testOption>set</testOption>
  <!-- 类型：NetconfHelloEntity -->
  <netconfHelloEntity>
     <!-- 类型：ENUM NetconfHelloType, 可取值： standardType，extendType，defaultType-->
     <helloType>standardType</helloType>
     <extendEntityList>
        <!-- 类型：string -->
        <extendEntity></extendEntity>
     </extendEntityList>
  </netconfHelloEntity>
  <!-- 类型：string -->
  <modelDiff></modelDiff>
  <rpcPrefixList>
     <!-- 类型：RpcPrefix -->
     <rpcPrefix>
        <!-- 类型：string -->
        <rpcQname></rpcQname>
        <!-- 类型：string -->
        <prefix></prefix>
     </rpcPrefix>
  </rpcPrefixList>
</netconfDriverInfo>
<commonDriverInfo>
  <!-- 类型：boolean -->
  <rollbackLastErrPackage>false</rollbackLastErrPackage>
  <!-- 类型：string -->
  <pathNeedPreProcessList>
     <pathNeedPreProcess></pathNeedPreProcess>
  </pathNeedPreProcessList>
  <!-- 类型：string -->
  <pathNeedPostProcess>
     <pathNeedPostProcess></pathNeedPostProcess>
  </pathNeedPostProcess>
  <!-- 类型：string -->
  <pathNeedEcsMappingList>
     <pathNeedEcsMapping></pathNeedEcsMapping>
  </pathNeedEcsMappingList>
  <!-- 类型：string -->
  <pathNeedEcsRpcMappingList>
     <pathNeedEcsRpcMapping></pathNeedEcsRpcMapping>
  </pathNeedEcsRpcMappingList>
  <!-- 类型：string -->
  <EcsDeviceDriverParaList>
     <!-- 类型：string, 属性value -->
     <EcsDeviceDriverPara key=""></EcsDeviceDriverPara>
  </EcsDeviceDriverParaList>
  <isNeedGlobalPush></isNeedGlobalPush>
  <isPathNeedPushList>
     <isPathNeedPush></isPathNeedPush>
  </isPathNeedPushList>
  <!-- 类型：string -->
  <pathNeedEcsDBHookList>
     <pathNeedEcsDBHook></pathNeedEcsDBHook>
  </pathNeedEcsDBHookList>
  <checkSync>false</checkSync>
  <!-- 类型：string，逗号（,）分隔 -->
  <unsupportedOperations>create</unsupportedOperations>

  <!-- 类型：Map<String, String> -->
  <pathUnsupportedOperationsMap>
     <pathUnsupportedOperations key=""></pathUnsupportedOperations>
  </pathUnsupportedOperationsMap>
  <!-- 类型：ENUM DeleteStrategy, 可选值：PATH_ADD，CONTAINER_WITH_PRESENCE -->
  <deleteStrategy></deleteStrategy>
  <uiSupportAbility>
     <supportOperationsList>
        <!-- 类型：string -->
```


版权所有 © 华为技术有限公司

```
            <supportOperations></supportOperations>
        </supportOperationsList>
        <connectProtocolsList>
            <!-- 类型：string -->
            <connectProtocols></connectProtocols>
        </connectProtocolsList>
    </uiSupportAbility>
    <!-- 类型：ENUM ConfigDeliveryStrategy, 可选值：SYNC_DB_AND_SYNC_DEVICE，
SYNC_DB_AND_ASYNC_DEVICE -->
    <configDeliveryStrategy></configDeliveryStrategy>
    <!-- 类型：ENUM OutOfSyncDeviceStrategyInHASwitching, 可选值：ALARM，
SYNC_TO_DEVICE,CUSTOM_HOOK -->
    <outOfSyncDeviceStrategyInHASwitching></outOfSyncDeviceStrategyInHASwitching>
    <!-- 类型：string -->
    <SwitchHACustomHook></SwitchHACustomHook>
    <ecsCheckHookList>
        <ecsCheckHook>
            <!-- 类型：string -->
            <moduleName></moduleName>
            <!-- 类型：string -->
            <appClass></appClass>
        </ecsCheckHook>
    </ecsCheckHookList>
    <ecsBehaviourHookList>
        <ecsBehaviourHook>
            <!-- 类型：string -->
            <apiClass></apiClass>
            <!-- 类型：string -->
            <implClass></implClass>
        </ecsBehaviourHook>
    </ecsBehaviourHookList>
    <!-- 类型：string 可选值：one-by-one, all-together -->
    <commitMode></commitMode>
  </commonDriverInfo>
  <customDriverInfo>
    <!-- 类型：string -->
    <maxPkgLength></maxPkgLength>
    <!-- 类型：ENUM LanguageType, 可选值：JAVA，GROOVY,OTHER -->
    <CustomLanguageType></CustomLanguageType>
  </customDriverInfo>
  <netconf-keepalive-info>
        <!--keepAliveMsg为netconf的保活报文，必须用CDATA方式转义，它满足netconf xml报文格式，其中
message-id的值必须用${messageId}替换-->
        <keepAliveMsg><![CDATA[<?xml version="1.0" encoding="UTF-8"?><rpc message-id="${messageId}"
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"><get-config><source><running/></source><filter
type="subtree"><system xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0" format-
version="1.0"> <systemInfo><sysUpTime/></systemInfo></system></filter></get-config></rpc>]]></
keepAliveMsg>
        <!--keepAlivePeriod：保活时间间隔，最小值10，单位：秒-->
        <keepAlivePeriod>15</keepAlivePeriod>
    </netconf-keepalive-info>
</netconf-driver>
```

5.1.2.3.3.2 设置业务自定义处理（可选）

当开放可编程提供的设备报文转换通用能力不能满足交互时，用户可以使用该方法自定义设备报文转换处理。

业务自定义处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfTransformer方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


    # 重写netconfTransformer方法
    def netconfTransformer(self, aoccontext, request=None):
        netconf_msg = NetconfMsg()
        if input.data.find('xmlns="urn:huawei:yang:huawei-ac-ne-snmp"') > -1:
            netconf_msg.msg =
```


版权所有 © 华为技术有限公司

```
            /input.data.replace('xmlns="urn:huawei:yang:huawei-ac-ne-snmp"',
            /'xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0"  '
            /'format-version="1.0"')
    else:
        netconf_msg.msg = input.data.replace('http://www.huawei.com/netconf/vrp',
                    /'urn:huawei:yang:huawei-ac-ne-snmp')
```

5.1.2.3.3.3 设置数据前置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异，如网络和设备枚举值定义可能不一致。设置前置处理后，当数据从设备同步NCE的时候，可以对数据进行处理，从而使NCE能够识别数据。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPreProcess字段。

数据前置处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfPreprocess方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


  # 重写netconfPreprocess方法
  def netconfPreprocess(self, aoccontext, request=None):
  self.logger.info('netconfPreprocess start.')
  doc = request.data

  # 对数据进行前置处理调整
  new_doc = doc.replace("<viewName>", "<viewName>pre")
  netconfAtomicConfig = NetconfAtomicConfig()
  netconfAtomicConfig.data = new_doc
  self.logger.info('netconfPreprocess end.')
  return netconfAtomicConfig
```

5.1.2.3.3.4 设置数据后置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异。设置后置处理后，当数据从NCE同步到设备的时候，可以对数据进行处理，从而使设备能够识别数据。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPostProcess字段。

数据后置处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfPostprocess方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


  # 重写netconfPostprocess方法
  def netconfPostprocess(self, aoccontext, request=None):
  self.logger.info('netconfPostprocess start.')
  doc = request.data

  # 对数据进行后置处理调整
  new_doc = doc.replace("<viewName>", "<viewName>post")
  netconfAtomicConfig = NetconfAtomicConfig()
  netconfAtomicConfig.data = new_doc
  self.logger.info('netconfPostprocess end.')
  return netconfAtomicConfig
```

5.1.2.3.3.5 设置数据联动（可选）

给设备下发数据的时候，设备可能会进行数据的联动处理，此时需要NCE也做数据联动处理，保持NCE与设备的同步。

 版权所有 © 华为技术有限公司

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedEcsDBHook字段。

数据联动代码示例如下。

```
# 导入EcsDbConfigOut, EcsDbConfig接口，重写dbPostProcess方法配置数据联动时使用
from aoc.snd.snd_model_pb2.ecsDbProcess_pb2 import EcsDbConfigOut, EcsDbConfig


    # 重写dbPostProcess方法
    def dbPostProcess(self, aoccontext, request):
        ecsData = request.data
        ecsPath = request.path
        ecsopType = request.opType
        ecsDbConfigOut = EcsDbConfigOut()
        if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/huawei-ifm:service-name"
        elif "huawei-ifm:l2vc/secondary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig1 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig2 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig1.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/reroute"
            ecsDbconfig2.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/redundancy"
        return ecsDbConfigOut
```

5.1.2.3.3.6 获取 syncId（可选）

syncId是数据更改的记录ID。通过比较设备上syncId和NCE上的syncId，可以了解设备上数据是否与NCE一致。例如，用户通过命令行对设备下发过业务，就可能会造成数据不一致。当数据不一致时，可以通过同步或对账对齐数据。

在向设备下发业务数据过程中检查设备和控制器的数据是否一致时，则需要在设备驱动配置common_driver.checkSync = True，并重写getSyncId方法获取syncId。

获取syncId代码示例如下。

```
# netconf协议报文方式查询获取syncId示例
# 导入devicemgr接口，重写getSyncId方法使用
from aoc import devicemgr


    # netconf协议报文方式查询获取syncId示例
    def getSyncId(self, aoccontext, request):
        """
        获取设备configFlowId
        :param aoccontext: 上下文环境
        :param request: 方法携带的参数
        :return: sync_id_info_output
        """
        self.logger.info('getSyncId start. aoccontext is %s,'
                         ' request is %s' % (aoccontext, request))
        device_id = request.deviceId
        # 设备yang path
        path = '/huawei-cfg:cfg'
        # 构造查询设备netconf subtree报文
        properties = {
            'subtree': self.CFG_SUBTREE.strip()
        }
        # 查询设备get报文
        data = devicemgr.query_data_from_device(device_id, 'OPERATIONAL', path, properties, 60000)
        self.logger.info('query data rsp is %s' % data)
        data_dict = {}
```


版权所有 © 华为技术有限公司

```
        if data.isSuccessful == "success" and data.result is not None:
            data_dict = NcsService.xmltodictnode(data.result)
            self.logger.info('data_dict is %s' % data_dict)
        sync_id_info_output = SyncIdInfoOutput()
        if 'cfg' not in data_dict or 'checkpoints' not in data_dict['cfg'] \
                or 'checkpoint' not in data_dict['cfg']['checkpoints']:
            return sync_id_info_output
        commit_ids = []
        for checkpoint in data_dict['cfg']['checkpoints']['checkpoint']:
            if checkpoint['commit-id']:
                commit_ids.append(checkpoint['commit-id'])
        # 取最新commit_id作为sync_id
        sync_id_info_output.SyncId = max(commit_ids)
        self.logger.info('getSyncId end. SyncId is: %s' % sync_id_info_output.SyncId)
        return sync_id_info_output
```

5.1.2.3.4 编写设备特性能力定制（可选）

当用户由于业务需要，可能会对差异发现范围、对账数据、同步数据进行定制处理，则需要编写设备特性能力定制代码。如果没有定制，则会对所有的特性进行全量差异发现。

特性能力定制代码示例如下。

```
# 导入FeatureCfgsMsg, Feature, Function接口，重写编写设备特性能力方法时使用
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function


    # 重写getFeatures方法，新增特性定制
    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True

         # 新增一个特性定制，编写差异发现的路径
        feature_msg.features.extend(self.build_feature('huawei-l3vpn', 'huawei-rtp', '(http://www.huawei.com/
netconf/vrp/huawei-l3vpn?revision=2018-06-11)l3vpn'))
        self.logger.info('getFeatures end.')
        return feature_msg


    def build_feature(self, name, depends, path):
        feature = Feature()

        # 特性名称
        feature.name = name

        # 操作类型
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()

        # 设置差异比较路径
        function.value = path

        # 设置从南向设备采集数据的路径
        function.collectPath = path
        feature_name = name.replace('huawei-', '')

        # 设置对账前定制，设置为“True”，则调用重写的preSyncToNe过程，设置为“False”，则不定制
        function.preSyncToNe = True

        # 设置同步前定制，设置为“True”，则调用重写的preSyncFromNe过程，设置为“False”，则不定制
        function.preSyncFromNe = False

        # 设置同步后定制，设置为“True”，则调用重写的postSyncFromNe过程，设置为“False”，则不定制
        function.postSyncFromNe = False
        feature.functions.extend([function])
```


版权所有 © 华为技术有限公司

```
    return [feature]


# 重写对账前定制过程
def preSyncToNe(self, request, aoccontext=None):
    self.logger.info('preSyncToNe start.')
    dataIn = request

    # 获得业务注册的数据路径
    regPath = dataIn.regPath

    # 获得特性名称
    featureName = dataIn.feature

    # 获得事务id
    transId = dataIn.transId
    self.logger.info("preSyncToNe begin regPath ={}, feature={}, transId={}", regPath, featureName, transId)

    # 获得业务处理后返回的差异数据
    diffDatas = dataIn.diffDatas

    # 获得差异数据列表的容量
    diffSize = len(diffDatas)

    # 如果差异为null或者差异数据容量为0，表明没有差异，函数直接返回null
    if diffDatas is None or diffSize == 0:
        return None

    # 定制数据一致性对账使用的数据
    dataOut = DiffDataOutMsg()

    # 遍历差异数据列表，处理差异数据
    for data in diffDatas:
        # 获得xml格式的差异数据
        diffXml = data

        # TODO 处理diffXml
        diffDataOut = diffData()

        # 该差异数据是否需要单独提交到南向设备。false表示不需要单独提交到南向设备
        diffDataOut.singleMsg = False
        diffDataOut.diffData = diffXml
        dataOut.diffDatas.extend([diffDataOut])
    self.logger.info('preSyncToNe end.')
    return dataOut
```

说明

新老设备被同一款SND包管理，不同设备支持的特性范围不一致时，则会导致同步出现异常。例如：SND包支持管理A、B两个特性，老设备仅支持A特性，新设备支持A和B特性，系统发起同步时，会尝试同时同步两个特性，系统会报老设备同步异常。

5.1.3 验证原子驱动包

5.1.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

 版权所有 © 华为技术有限公司

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

5.1.4 编译原子驱动包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件复制到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

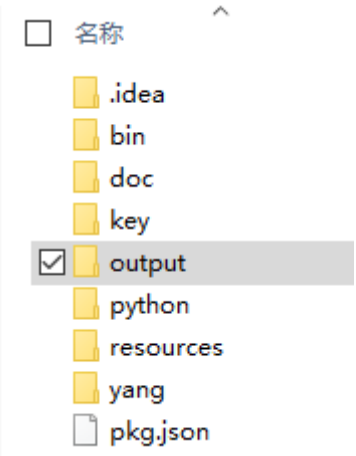
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
```


版权所有 © 华为技术有限公司

```
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

5.1.5 开发 NETCONF SND 示例

5.1.5.1 开发 NE8000 M8 NETCONF SND 示例

本示例开发华为NE8000 M8 NETCONF SND。华为NE8000 M8设备和控制器是同一厂家，数据格式、报文格式无差异不需要定制，SND只需要编写设备纳管定制代码、设备驱动数据代码，同时设备特性能力不定制，默认全量差异发现。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表5-8设置软件包的相关属性。

   表 5-8 NE8000M8 SND 属性

属性名称	属性值
名称	NE8000M8_SND
版本	1.0.0
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	NetEngine8000M8
协议类型	NETCONF

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
    "name": "NE8000M8_SND",
    "version": "1.0.0",
    "description": "NE8000M8_SND",
    "package-type": "snd",
    "producer": "HUAWEI",
    "nce-min-versions": [
```


版权所有 © 华为技术有限公司

```
      "1.0.0"
    ],
    "snd-id": "NE8000M8_SND",
    "devices": [
     {
      "vendor": "HUAWEI",
      "device-type": "NetEngine8000M8",
      "device-version": "V800R012C00SPC300"
     }
    ],
    "hooks": [
     {
      "type": "snd",
      "key": "ecs-driver",
      "python-class-name": "com.huawei.controller.devicetype.ne8000m8snd.NE8000M8NETCONF"
     }
    ]
}
```

步骤3 编写设备YANG模型

从厂商获取NE8000设备的YANG模型，替换掉软件包中默认的YANG模型文件。

步骤4 开发代码

1. 编写设备纳管定制
   a. 注册设备sysoid信息

```
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class NE8000M8NETCONF(NetconfSND):
      def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"
    sysoidEntity.deviceType = "ROUTER"
    sysoidEntity.deviceModel = "NetEngine 8000 M8"
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

   b. 配置设备的连接能力信息

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    primaryConnectInfo = connectInfos.connectInfo.add()
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
    primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.extendType
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

2. 编写设备类型能力定制
   a. 配置设备通用驱动数据

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
```

 版权所有 © 华为技术有限公司

```
    syncToDel.value = "true"
    return common_driver
```

b. 配置设备协议类驱动数据

设备支持二阶段下发，phase配置为"two"。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo

    def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()
    netconf_driver.phase = "two"
    netconf_driver.classification = "huawei-v5"
    netconf_driver.testOption = "set"
    return netconf_driver
```

----结束

5.1.5.2 开发 S6720 NETCONF SND 示例

本示例开发华为S6720 NETCONF SND。华为S6720是一款huawei-v5交换机，S6720设备和控制器是同一厂家，数据格式、报文格式无差异不需要定制，SND只需要编写设备纳管定制代码、设备驱动数据代码，同时定制设备特性能力不定制，默认全量差异发现。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表5-9设置软件包的相关属性。

   表 5-9 S6720 SND 属性

属性名称	属性值
名称	S6700-SND
版本	0.0.12
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	S6720-30C-EI-24S-AC
协议类型	NETCONF

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
    "name": "S6700-SND",
    "version": "0.0.12",
    "description": "SND",
    "package-type": "snd",
    "producer": "HUAWEI",
```


版权所有 © 华为技术有限公司

```
  "nce-min-versions": [
      "1.0.0"
  ],
  "snd-id": "S6720-30C-EI-24S-AC-V200R021C10",
  "devices": [
    {
      "vendor": "HUAWEI_S",
      "device-type": "S6720-30C-EI-24S-AC",
      "device-version": "V200R021.*"
    }
  ],
  "hooks": [
    {
      "type": "snd",
      "key": "ecs-driver",
      "python-class-name": "com.huawei.controller.devicetype.snd_s6720_v2r21c10.SND_S6720_V2R21C10"
    }
  ]
}
```

步骤3 编写设备YANG模型

从厂商获取S6720设备的YANG模型，替换掉软件包中默认的YANG模型文件。

步骤4 开发代码

1. 编写设备纳管定制
   a. 注册设备sysoid信息

```
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class SND_S6720_V2R21C10(NetconfSND):
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.23.340"
    sysoidEntity.deviceType = "SWITCH"
    sysoidEntity.deviceModel = "S6720-30C-EI-24S-AC"
    sysoidEntity.deviceVendor = "HUAWEI_S"
    return sysoidInfo
```

   b. 配置设备的连接能力信息

   连接策略配置为compatible agent处理，在agentImpl参数中配置实现类agent。

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    primaryConnectInfo = connectInfos.connectInfo.add()
        primaryConnectInfo.connectPolicy = COMPATIBLE_CONNECT
        primaryConnectInfo.channelInfo.readChannel = SINGLE_CHANNEL
        primaryConnectInfo.channelInfo.is_read_share_write = True
        primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
        primaryConnectInfo.connectEntity.methodType = 'java'
        primaryConnectInfo.connectEntity.agentImpl =
'com.huawei.enterprise.naas.netconfcliagent.service.NetconfCliAgentImpl'
        primaryConnectInfo.connectEntity.priority = 0
        primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.defaultType
        primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
        self.logger.info('getConnectInfo end.')
        return connectInfos
```

2. 编写设备类型能力定制
   a. 配置设备通用驱动数据

 版权所有 © 华为技术有限公司

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
    return common_driver
```

b. 配置设备协议类驱动数据

设备支持一阶段下发，phase配置为"one"。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo

    def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()
    netconf_driver.phase = "one"
    netconf_driver.classification = "huawei-v5"
    netconf_driver.testOption = "set"
    return netconf_driver
```

----结束

5.2 开发 CLI 协议设备原子驱动包

本章节介绍CLI协议设备原子驱动包的开发过程。

用户在开发SND软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

5.2.1 创建原子驱动包模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 在弹出的对话框中设置软件包的相关属性。

请根据表5-10设备软件包的相关属性。

表 5-10 CX600 软件包属性

属性名称	属性值
名称	CLI_SND
版本	1.0.0
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	CLI
协议类型	CLI

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

版权所有 © 华为技术有限公司

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

5.2.2 开发原子驱动包

本节详细介绍开发SND包全过程。

5.2.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 请按照参数表修改对应参数的值。

表 5-11 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：ssp（Specific Service plugin）、gnd（Generic NE driver）、snd（Specific NE driver）、mp（SND Migrate plugin）。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。不填写则取默认值global。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。仅支持该字段为“huawei”（不区分大小写）的SND和GND包。	必选
augment-isolation	模型范围标识。	布尔值，只能为 true 或 false。DCN单站场景需要配置为true，其他场景为false。	可选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。 version（必选）字段：版本号三段式规则。	可选
snd-type	snd的类型。	值为“system”，表示为系统内置的snd包。系统内置的snd包，该参数值不建议修改。 该参数未设置时，默认值为“specific”，表示用户自定义包。	可选
snd-id	snd包才需要设置的值。id信息区分不同snd包，不同名的包不能使用相同的snd-id。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。 snd包时，必须设置，其他类型软件包时，不需要设置。	可选
devices	snd, gnd包特有值，驱动匹配的设备信息。	如果配置，则必须包含： ● 驱动设备厂商vendor（必选），不能超过 128个字符。 ● 具体设备类型device-type（必选），不能超过 128个字符。 ● 具体设备的版本device-version（必选），不能超过 128个字符。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。只能包含，数字、大小写字母，下划线（“_”），连字符(“-”)，不能超过128个字符。SND包设置为"snd"。 ● key（必选），回调key值，不能超过128个字符。SND包一般只需配置“ecs-driver”，指定在SND包中具体实现类；用户不使用系统默认连接agent时，再补充配置“ecs-connect-agent”，自定义hook中配置的连接agent，要生效需要同时设备的连接能力参数connectPolicy 配置为“COMPATIBLE_CONNECT”。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“device-version”、“python-class-name”，参数含义和修改方法请参见上述表格。编辑文件示例如下：

```
{
    "name": "CLI_SND",
    "version": "1.0.0",
    "description": "CLI_SND",
    "package-type": "snd",
    "producer": "HUAWEI",
    "nce-min-versions": [
        "1.0.0"
    ],
    "snd-id": "CLI_SND",
    "devices": [
     {
       "vendor": "HUAWEI",
       "device-type": "CLI",
       "device-version": "V800R018C10SPC612B612"
     }
    ],
    "hooks": [
     {
       "type": "snd",
       "key": "ecs-driver",
       "python-class-name": "com.huawei.controller.devicetype.clisnd.CLI"
     }
```

 版权所有 © 华为技术有限公司

```
  ]
}
```

----结束

5.2.2.2 编写设备 YANG 模型

SND包中的YANG不参与编译，用于描述设备的配置能力、设备的北向接口以及生成南向报文。用户根据SND包类型，获取或编写对应的YANG模型，替换掉软件包中默认的YANG模型文件。

YANG模型构建流程：

1. 节点定义：准备设备的命令行信息、命令行手册等，分析命令行中前缀、进子视图命令、配置命令特点，根据命令行样式特点初步建模，通过默认YANG/CLI转换规则映射到对应的节点类型上。
2. 优化模型和使用扩展项：审视初步建模结果，如果发现有例外的节点类型不满足YANG/CLI转换诉求，则分析是否可以优化构建模型，查阅系统提供的基础扩展项，分析是否有可满足诉求的扩展项。
   - 如果存在，则在对应节点使用基础扩展项。
   - 如果不存在，则需要用户在py文件中自定义开发自己的扩展项语法，并在对应节点使用。
3. 验证模型：在真实设备上验证构建的YANG模型是否正确。如果存在问题，则分析问题原因，优化调整YANG模型后，再重新验证。

5.2.2.2.1 节点定义

步骤1 建模准备：准备设备的命令行信息、命令行手册等。推荐采集设备的命令行dump文件，并保留命令行换行等有效信息。

步骤2 熟悉默认YANG/CLI转换规则，YANG/CLI转换规则在不使用任何扩展项情况下，根据自定义YANG模型默认生成CLI命令。

默认YANG/CLI转换规则如下：

- container节点转换为命令行的前缀，能够叠加到任意深度。
- list节点转换为子视图，其中list的key值作为进入子视图的参数。
- leaf节点转换为单个命令，跟着一个leaf的参数值。
- leaf-list节点转换为单个命令，跟着空格间隔的参数序列值。

举例说明下，例如YANG文件如下所示：

```
list server {
  key “name”;
  leaf name {
    type string;
  }
  leaf description {
    type string;
  }
  container ip {
      leaf ip-address {
          type inet:ip-address;
      }
      leaf ip-port{
          type inet:port-number;
      }
```

 版权所有 © 华为技术有限公司

```
  }
}
```

用户输入的配置数据如下所示：

```
<server>
  <name>DCServer1</name>
  <description>testAOC</description>
  <ip>
    <ip-address>192.168.1.1</ip-address>
    <ip-port>803</ip-port>
  </ip>
</server>
```

默认转换成命令行：

```
server  DCServer1
   description   testAOC
   ip ip-address 192.168.1.1
   ip ip-port  803
```

步骤3 命令行特征分析：分析命令行特征，对应到YANG模型节点类型上。

表 5-12 命令行示例

命令行样例	特征分析	节点类型	YANG模型
interface Ethernet0/1/0 mtu 100 description 1 undo shutdown dcn # **interface** Ethernet0/1/1 undo shutdown dcn address 10.2.3.4/32 10.2.3.5/32	interface 作为命令行前缀	container	container interface

	Ethernet0/1/0进入子视图，	list，0/1/0作为list的key值	list Ethernet{
			key 'name';
			leaf name{
			type string;
			}
			..

	mtu 100等，一个关键字，一个值	leaf	leaf mtu{
			type int32;
			}

	address，一个关键字，多个值	leaf-list	leaf-list address {
			type inet:ipv4-prefix;
			}

步骤4 初步建模：根据命令行特征分析结果，输出初步建模文件。

```
container interface{
  list Ethernet{
    key 'name';
    leaf name{
      type string;
    }
    leaf mtu{
      type int32;
    }
```


版权所有 © 华为技术有限公司

```
    leaf description{
      type string;
    }
    leaf-list address{
      type inet:ipv4-prefix;
    }
  }
}
```

----结束

5.2.2.2.2 优化模型和使用扩展项

步骤1 分析例外情况：分析初步建模文件，梳理默认YANG/CLI转换规则无法满足的情况。

```
interface Ethernet0/1/0
 mtu 100
 description 1
 undo shutdown
 ip address 192.168.4.20 255.255.255.0
```

如ip address 192.168.4.20 255.255.255.0，有ip和address2个关键字，并有2个值，无法使用默认规则直接建模。

步骤2 优化模型使用扩展项：在基础扩展项库中，查找是否有可满足例外情况的扩展项。基础扩展项请参见**13.2.1 基础扩展项简表**。

- 有，在对应YANG模型节点使用基础扩展项。

  在节点名称后补充基础扩展项。用法为“cli_annotations.yang的prefix名称:扩展项标签名称”，如下例所示。

  ```
  list l2 {
    hw:render-view “false”;
    key k1;
    leaf k1 {
    ..
  ```

- 没有，则需要用户在py文件中自定义开发自己的扩展项语法，并在对应节点使用自定义扩展项。系统提供了4种自定义扩展项，如**表5-13**所示。

表 5-13 自定义扩展项

自定义扩展项标签	扩展项参数	说明
cli-custom-transform	create,update,delete，可以用逗号“,”间隔，组合使用	由SND包中yangToCli方法处理YangToCli的转换逻辑 使用举例： cli-custom-transform 'create,update,delete'
cli-custom-read	two-step，one-step和one-step-no-parse，三个参数只能同时出现一个，不能叠加	由SND包cliToYang方法处理CliToYang的转换逻辑 使用举例： cli-custom-read 'two-step'

版权所有 © 华为技术有限公司

自定义扩展项标签	扩展项参数	说明
cli-custom-processor	pre-read、post-config和post-config-by-line、post-config-rollback，可以用逗号“,”间隔，组合使用	由SND包中readCliPreProcessor、configCliPostProcessor方法进行YANG/CLI转换的前置后置处理 使用举例： cli-custom-processor 'post-config,pre-read'
cli-custom-rpc	无	将rpc的YANG与CLI的正向与逆向转换过程分别委托给SND包中的yangToCli和cliToYang两个方法来实现 使用举例： cli-custom-rpc

在YANG模型节点名称后补充自定义扩展项。用法为“cli_annotations.yang的prefix名称:扩展项标签 参数”，如下例所示。

```
leaf router-id {
    hw:cli-custom-read 'two-step';
    type string;
}
```

----结束

5.2.2.2.3 验证模型

步骤1 请编译SND包，并在环境导入安装SND包。

步骤2 向设备下发YANG配置，观察YANG/CLI双向转换结果是否符合预期。

- 符合预期，验证YANG模型成功。
- 不符合预期，优化模型后重新验证。

----结束

5.2.2.2.4 YANG 模型自定义功能

如果用户需要使用自定义数据加密处理、高危操作定义、YANG模块黑名单屏蔽、YANG模型免校验、自定义模型权限等功能，请参考下文配置。

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

版权所有 © 华为技术有限公司

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

表 5-14 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d
-	-	type	节点类型，取值范围： ● leaf ● leaf-list ● typedef –自定义类型 默认是leaf

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
    <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
    <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
    <path>/db-security-test:security-top/security-list/security-list-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
    <rpcs>
        <rpc>huawei-aaa:reactivateUser</rpc>
        <rpc>huawei-aaa:changeMyPassword</rpc>
        <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
    </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

 版权所有 © 华为技术有限公司

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
```

 版权所有 © 华为技术有限公司

```
    },
    {
      "uri-pattern":"resetStatictic",
      "method":"exec"
    }
    ]
  }
  ]
}
```

5.2.2.2.5 实例开发 YANG 模型

本示例软件包为CLI SND包，请用户编写YANG模型并替换软件包中默认的YANG模型文件。下面以创建子接口相关命令分析其对应的YANG模型。

表 5-15 分析 CLI 命令并设计 YANG 模型

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
ip vpn-instance 2G-RAN	ip：前缀，转换为container节点。	无。
	vpn-instance：子视图，转换为list节点。	无。
	2G-RAN：命令参数值，转换为leaf节点。	不显示leaf节点名称，需要annotations定制。
ipv4-family	ipv4-family：子视图，因为有许多子节点，所以转换为container节点。	container不能进入视图，进入子视图需要annotations定制。
route-distinguisher 100:1	route-distinguisher 100:1：单个命令，转换为leaf节点。	删除命令需要包含leaf的值，需要annotations定制。
tnl-policy LDP	tnl-policy LDP：单个命令，转换为leaf节点。	无。
apply-label per-instance	apply-label per-instance：单个命令，转换为leaf节点。	删除命令需要包含leaf的值，需要annotations定制。
vpn-target 100:11 export-extcommunity vpn-target 100:11 import-extcommunity	vpn-target：前缀，转换为list节点。	删除命令需要包含leaf的值，需要annotations定制。
	100:11：命令参数值，转换为leaf节点。	通过annotations定制，将list中所有leaf节点转换成一行命令。 需要annotations来定制，不进入视图。
	export-extcommunity：命令参数值，转换为leaf节点。	无。

版权所有 © 华为技术有限公司

输出的完整YANG文件如下：

```
module huawei-ip {

    yang-version 1;

    namespace "http://huawei.com/ip";

    prefix ip;

    import cli_annotations {
        prefix "hw";
    }

    revision "2019-09-01" {
        description "Specific NE Driver for ip vpn-instance cli";
    }


    container ip {
        hw:custom-read 'one-step-no-parse';
        list vpn-instance {
            key vpn-instance-name;
            leaf vpn-instance-name {
                hw:render-name false;
                type string;
            }
            leaf description {
                type string;
            }

            container ipv4-family {
                hw:render-view true;
                leaf route-distinguisher {
                    hw:render-undo-value true;
                    type string {
                      length "3..21";
                    }
                }

                leaf tnl-policy {
                    type string;
                }

                leaf apply-label {
                    hw:render-undo-value true;
                    type enumeration {
                        enum per-instance;
                        enum per-route;
                    }
                }
                list vpn-target {
                    hw:render-single-line;
                    hw:render-view false;
                    key "name mode";
                    leaf name {
                        hw:render-name false;
                        hw:render-undo-value true;
                        type string {
                            length "3..21";
                        }
                    }
                    leaf mode {
                        hw:render-name false;
                        hw:render-undo-value true;
                        type enumeration {
                            enum "export-extcommunity";
                            enum "import-extcommunity";
                            enum "both";
                        }
```


版权所有 © 华为技术有限公司

```
            }
          }
        }
      }
    }

}
```

5.2.2.3 开发代码

开发Python代码，实现设备的纳管、设备的配置管理、定制差异发现及同步和对账能力。

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码。用户可以重命名py文件名称和路径，将名称改为有意义的名称，需要与pkg.json文件的python-class-name中py文件名称字段保持一致。

5.2.2.3.1 引入必要的头文件

在代码最前面，首先需要引入必要的头文件中的相关接口。

下文在重写各个方法时描述了需要引用的头文件接口，请用户根据SND包类型、业务逻辑，参考示例代码导入需要的接口。

import语句应当放在文件头部，置于模块说明之后，全局变量之前。

5.2.2.3.2 编写设备纳管定制

编写设备纳管定制，包含两部分：注册设备sysoid信息和配置设备的连接能力信息。

注册设备 sysoid 信息

用户通过getSysoidInfo方法设置设备的sysoid信息，用于纳管此款型设备。

该方法不是必须要重写。纳管时期望对设备的sysoid信息校验，则需要重写方法，并准确配置sysoid字段信息。不对设备的sysoid信息校验，则可以不重写方法，通过pkg.json文件里的三元组来匹配。

注册设备sysoid信息的参数如下表所示。

表 5-16 注册设备 sysoid 参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
sysoid	否	STRING	N/A	N/A	设备唯一标识。 sysoid 可以通过以下方式获取： • 命令行查询，如华为设备一般可以执行display system information查询 • mib browser工具查询 • 通过OID网站查找，**https://oidref.com**

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
deviceType	否	STRING	N/A	N/A	设备类型。 请根据具体设备，匹配对应的设备类型，常用取值ROUTER、SWITCH、FIREWALL。
deviceModel	否	STRING	N/A	N/A	设备款型。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。
deviceVendor	否	STRING	N/A	N/A	设备厂商。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。

注册设备信息代码示例如下。

```
# 保留默认代码中导入CliSND、SysoidInfo接口，重写getSysoidInfo方法时需要使用。
from aoc.snd.clisnd import CliSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo


# 默认代码定义实现类代码可全部删除，按如下示例定义继承CliSND父类的子类
# CliSND是所有CLI协议原子驱动包的父类，默认的配置已在父类中实现
class CLI(CliSND):

    # 重写getSysoidInfo方法，注册设备sysoid值、设备类型信息、设备型号信息、设备厂商信息
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()

    # 设置设备的sysoid
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.87.27"

    # 设置设备的设备类型
    sysoidEntity.deviceType = "ROUTER"

    # 设置设备的设备款型
    sysoidEntity.deviceModel = "CLI"

    # 设置设备的设备厂商
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

配置设备的连接能力信息

设备连接能力信息常用参数请参见下表，详细的参数说明请参考《插件包API参考》。继承的父类不同，默认值则可能存在差异，以下表格中不做特殊说明，都是继承CliSND父类后的默认值。

版权所有 © 华为技术有限公司

表 5-17 设备连接能力常用参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
connectPolicy	否	ENUM	DEFAULT_CONNECT/ COMPATIBLE_CONNECT	DEFAULT_CONNECT	连接策略 DEFAULT_CONNECT，由默认agent处理 COMPATIBLE_CONNECT，由compatible agent处理
channelInfo.writeChannel	否	ENUM	SINGLE_CHANNEL/ PROTECTED_MODE	SINGLE_CHANNEL	写通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护
channelInfo.readChannel	否	ENUM	SINGLE_CHANNEL/ PROTECTED_MODE	N/A	读通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护
channelInfo.is_read_share_write	否	bool	True/ False	True	是否读写通道共享 True：读写通道共享。只需要配置写通道类型即可，另一种通道无需配置。 False：读写通道不共享。读写通道类型都需要配置。
protocolType	否	ENUM	netconf/cli/restful/ restconf	cli	设置SND包的协议类型
helloType	否	ENUM	defaultType/ standardType/ extendType	N/A	报文类型，CLI协议不需要配置 defaultType，建立schema对接的NETCONF通道，能力集置为空。 standardType，建立标准yang能力集的NETCONF通道，能力集置为空。 extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。
extendEntity	否	ARRAY_STRING	N/A	N/A	能力集列表，CLI协议不需要配置 若报文类型为extendType类型时，需要携带扩展能力集。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
connectionPriority	否	ENUM	PRIMARY_CONNECTION/AUXILIARY_CONNECTION	PRIMARY_CONNECTION	连接调用优先级 PRIMARY_CONNECTION：单协议SND包、双协议SND包的主协议需配置为该值。 AUXILIARY_CONNECTION：双协议SND包时，辅助协议需配置为该值。

配置主协议设备连接能力信息代码示例如下。

```
# 导入ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION接口，重写getConnectInfo方法配置设备连接能力时使用，用于配置连接信息，如与设备的建连协议，建立几个通道等
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION
# 导入SINGLE_CHANNEL接口，重写getConnectInfo方法配置设备连接通道时使用
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL


    # 重写getConnectInfo方法，配置CLI协议类型及连接能力信息，若不重写则按默认值配置
    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()

    # 所有类型的SND包，都需要配置主协议能力信息
    primaryConnectInfo = connectInfos.connectInfo.add()

    # 设置连接策略
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT

    """
    设置通道配置。NETCONF协议需要配置不同的通道类型，
    SINGLE_CHANNEL表示单链路无保护，
    当is_read_share_write为true表示读写通道共享，readChannel无需配置。
    """
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True

    # 设置主协议类型
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli

    # 设置连接调用优先级
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION

    self.logger.info('getConnectInfo end.')
    return connectInfos
```

5.2.2.3.3 编写设备类型能力定制

编写设备类型能力主要包括：配置设备驱动数据、业务自定义处理、数据前置后置处理、数据联动处理。

5.2.2.3.3.1 配置设备驱动数据

提供了两种方法，编写Python配置设备驱动数据，或者通过编写xml文件配置设备驱动数据，下面分别进行说明。

版权所有 © 华为技术有限公司

通过代码配置设备驱动信息

- **基于通用能力定制示例，定义与设备的交互策略**

  编写基于通用能力的交互策略。常用参数请参见下表，详细的参数说明请参考《 插件包API参考 》。

  表 5-18 设备通用驱动参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
checkSync	否	bool	true/false	false	在向设备下发业务数据过程中检查设备和控制器的数据是否一致。
unsupportedOperations	否	STRING	"create"/"delete"/"create,delete"	N/A	设备不支持的操作类型，自动进行转换。如将 "create,delete"转换为 "merge,remove" 下发配置时可能会修改设备已有配置，请谨慎配置，以免引发业务安全风险。
deleteStrategy	否	DeleteStrategy	PATH_ADD/CONTAINER_WITH_PRESENCE	CONTAINER_WITH_PRESENCE	对于delete、remove操作的报文，是否删除报文中的具体信息 PATH_ADD时，指定PATH的报文内容删除。 CONTAINER_WITH_PRESENCE时，下发报文不做内容删除。
syncToDel	否	N/A	syncToDel.key = "sync-to-del-enable" syncToDel.value 可以为 "true"或“false”	false	设置数据对账时是否支持删除南向设备配置实例。 当此参数设置为true时，会在数据对账时删除仅设备侧存在的配置数据，可能引发业务安全风险。建议将此参数设置为false。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
pathNeedPreProcess	否	STRING	N/A	N/A	该字段填写需要进行报文前置处理的特性QName。若该字段填写了对应QName，对于设备返回的查询结果报文，会在处理之前先调用netconfPreprocess进行加工，加工之后再进行报文转换。
pathNeedPostProcess	否	STRING	N/A	N/A	该字段填写需要进行报文后置处理的特性QName。若该字段填写了对应QName，对于即将下发设备的报文，会在下发之前先调用netconfPostprocess进行加工，加工之后再下发设备。
pathNeedEcsMapping	否	STRING	N/A	N/A	该字段填写需要进行业务报文定制处理的特性QName。若该字段填写了对应QName，对于下发设备的报文和设备返回的都统一调用toDocument、toDataObject进行转换。
pathNeedEcsRpcMapping	否	STRING	N/A	N/A	该字段填写需要进行rpc报文定制处理的rpc QName。若该字段填写了对应QName，对于下发设备的rpc报文和设备返回的都统一调用toRpcDocument、toRpcDataObject进行转换。
pathNeedEcsDBHook	否	STRING	N/A	N/A	下发配置到设备后，某些配置会联动生成其他配置，而控制器只会存储联动前的配置，会导致控制器和转发器配置不一致。因此，控制器端也需要在配置下发前进行联动处理。

通用能力定制代码示例如下。

```
# 导入CommonDriverInfo接口，重写getCommonDriverInfo方法编写设备类型时使用，基于通用能力定制，用于定制通用操作类型转换等
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo


    # 重写getCommonDriverInfo方法，配置设备通用驱动，不重写则按默认值配置
    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()

        # 在向设备下发业务数据过程中检查设备和控制器的数据是否一致。建议设置此参数为true，并实现getSyncId方法获取SyncId。
        common_driver.checkSync = True

    # 设备不支持的操作类型，自动进行转换，如将 "create,delete"转换为"merge,remove"。
    common_driver.unsupportedOperations = "create,delete"
```

 版权所有 © 华为技术有限公司

```
    common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
syncToDel = common_driver.para.add()

# 定义key值，同步删除
syncToDel.key = "sync-to-del-enable"

# 设置数据对账时是否支持删除南向设备配置实例。取值：true（支持）。
syncToDel.value = "true"
return common_driver
```

- **基于协议类型定制示例，定义CLI协议参数**

  CLI SND包，支持CLI协议，必须要定义CLI协议参数。

  协议类型定制代码示例如下。

```
# 导入CLIDriverInfo接口，重写getCliDriverInfo方法时使用
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo
import os
import codecs


# 重写getCliDriverInfo方法，配置设备Cli驱动，不重写则按默认值配置
# 这里是通过自定义读取cli-driver.properties文件进行配置的，Cli驱动参数较多，通过文件配置较方便
def getCliDriverInfo(self, aoccontext, request=None):
    self.logger.info('getCliDriverInfo start.')
    cliDriverInfo = CliDriverInfo()
    self.logger.info(self.get_dict())

# 将字典中的键值对赋值给配置项
    for key, value in self.get_dict().items():
        cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
        cliDriverEntity.key = key
        cliDriverEntity.value = value
    self.logger.info('getCliDriverInfo end.')
    return cliDriverInfo


# 将resources/cli-driver.properties设备驱动文件中的信息读入到字典中
def get_dict(self):
    try:
        path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
        self.logger.info(self.resourceDir)
        self.logger.info(path)
        pro_file = open(path, 'r')
        properties = {}
        for line in pro_file:
            if line.find('=') > 0:
                strs = line.replace('\n', '').split('=')
                properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
    except IOError as e:
        self.logger.info('getCliDriverInfo start.')
        self.logger.info(e)
        return {}
    else:
        pro_file.close()
    self.logger.info(properties)
    return properties
```

通过文件配置设备驱动信息

通过文件配置设备驱动信息，系统自动解析文件或用户自定义解析文件逻辑。用户按照要求命名和编写xml配置文件，不需要编码系统可自动解析xml文件配置设备驱动信息。

用户通过“resources/cli-driver.properties”文件，可以定制CLI设备驱动信息，用于解析和识别设备的CLI回显。配置项详细信息如下表所示。

 版权所有 © 华为技术有限公司

表 5-19 配置项

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
userModePrompt	^<HUAWEI.*>$	-	是	用户模式下设备命令行提示符正则表达式，用于解析返回报文是否为用户模式提示符。	<HUAWEI123>
privilegedModePrompt	^\\[[~*]HUAWEI.*\\]$	-	是	特权模式下设备命令行提示符正则表达式，用于解析返回报文是否为特权模式的提示符。	[~HUAWEI123]
promptSkipRegex	(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}(@prompt@){1}$	-	否	跳过解析的提示符正则表达式，用于跳过解析返回报文中的提示符场景。 正则表达式中的@prompt@将被替换为userModePrompt或privilegedModePrompt配置值。	-
userNameResponse	Login:	-	否	登录时提示输入用户名的命令行提示符匹配规则，用于解析返回报文是否为输入用户名提示符。	Login:
passwordResponse	Password:	-	否	登录时提示输入密码的命令行提示符匹配规则，用于解析返回报文是否为输入密码提示符。	Password:
InitCommand	display users	-	否	初始连接设备后发送的初始化命令，用于系统识别到成功登录设备后自动执行和显示的初始化命令。	<HUAWEI123>display users User-Intf Delay Type Network Address AuthenStatus AuthorcmdFlag + 34 VTY 0 00:00:00 TEL 10.183.45.110 pass yes Username : huawei <HUAWEI123>

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
keepAliveCommand	display users	-	否	保活连接命令，用于定期向设备发送该命令以保持连接。	<HUAWEI123>display users User-Intf Delay Type Network Address AuthenStatus AuthorcmdFlag + 34 VTY 0 00:00:00 TEL 10.183.45.110 pass yes Username : huawei <HUAWEI123>
cliTxCreateCommand	return\n	-	是	事务创建命令，用于发送配置前先下发创建事务。	<HUAWEI123>return <HUAWEI123>
cliTxCommitCommand	commit\n	-	是	事务提交命令，用于配置提交后再下发提交事务。	[~HUAWEI123]screen-length 24 [*HUAWEI123]commit [~HUAWEI123]
cliTxSaveCommand	return\n	-	是	事务保存命令，用于配置提交成功后再下发事务保存。	-
cliTxCancelCommand	clear configuration candidate \n	-	是	事务取消命令，用于取消配置等情况时下发事务取消。	[~HUAWEI123]screen-length 0 [*HUAWEI123]clear configuration candidate [~HUAWEI123]
cliTxCancelInterval	2000	2000	否	设备事务异常时，取消事务间隔时间，单位毫秒。用于多次取消事务操作时间间隔。	-
cliTxCancelRetryTimes	3	3	否	取消事务重试次数，用于设备事务异常时，取消事务可重试的次数。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
abortCommand	quit	-	否	异常停止命令，用于CLI协议出现异常时，通过该命令强制终止当前命令的执行，最好配置成Ctrl+C的unicode码。 未配置值时，CLI协议异常时处理策略是先断连，再建连。	[~HUAWEI123]quit <HUAWEI123>
connectionTimeout	10000	10000	否	连接超时时间，单位ms，用于判断连接超时。	-
keepAliveInterval	120000	20000	否	发送保活命令进行保活的间隔时间，单位为ms，用于保持设备连接。	-
errorMessagePattern	^Error:?[ ]{1}.*	-	否	配置Error正则表达式，用于解析返回报文中是否为配置失败的信息。	Error: Unrecognized command found at '^' position.
ignoreMessagePattern	randomvalueshould#not#match	-	否	忽略配置Error匹配正则表达式，用于当解析返回报文中为配置失败情况时但该场景又需要被忽略时。	-
warningMessagePattern	^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*	-	否	回显信息告警正则表达式，用于解析返回报文是否为告警信息。	Info: Removing directory cfcard:/1.txt/.....Done!
maxResponseSize	400	400	否	执行一条命令后，能够允许的最大返回报文大小，单位为KB。用于限制超大报文等异常场景。	-
showConfigCommand	return \ndisplay current-configuration	-	否	配置全量读的设备命令，用于查询设备配置。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
showConfigSkipRegex	^(--|echo|#).*$	-	否	不需要解析的关键字正则表达式，用于忽略解析含有该关键字正则表达式的返回报文。	-
configModeCommand	system-view	-	否	配置模式进入命令，用于下发实际配置命令（创建、删除、更新等）前下发该命令进入配置模式。	<HUAWEI123>system-view
partialReadCommand	return \nsystem-view\n${context}\ndisplay this	-	否	配置部分读的设备命令，用于查询设备部分配置。提前构造部分读命令，然后替换${context}生成具体的部分读命令。	<HUAWEI123>return <HUAWEI123>system-view [~HUAWEI123]interface Ethernet0/1/2 [~HUAWEI123-Ethernet0/1/2]display this
indentationSequence	\u0020\u0020\u0020\u0020	\u0020\u0020\u0020\u0020	否	命令行缩进序列，用于解析设备命令行的缩进序列。HUAWEI设备通常左缩进，四个空格键的缩进序列。	-
exitSubmodeCommand	quit	-	否	返回上级视图的命令，用于返回上级视图下发该命令。通常为quit、exit。	[~HUAWEI123]quit <HUAWEI123>
submodeStartSequence	@indent@	-	否	子模式开始序列，用于解析设备命令行子模式开始的缩进序列。@indent@表示缩进一级。	acl number 10001 rule 2 name xxx deny
submodeEndSequence	@dedent@	-	否	子模式结束序列，用于解析设备命令行子模式结束的缩进序列。@dedent@表示反缩进一级。	acl number 10001 rule 2 name xxx deny #

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
negateCommand	undo \u0020	-	否	逆命令表达式，用于解析设备命令行的no、undo等关键字，识别是否为逆命令。	[~HUAWEI123]undo
shutdownCommand	\u0020shutdown	\u0020shutdown	否	shutdown命令表达式，用于解析当下发delete命令时设备返回报文中含有shutdown等关键字，需要用户输入后才能继续的情况。	[~HUAWEI123-Ethernet0/1/5]dis this # interface Ethernet0/1/5 shutdown
contextSupportsSinglelineCommand	false	-	否	表示删除命令是否长命令模式，用于配置是否支持解析长命令模式。HUAWEI设备不支持，所以配置成false。 配置值包括true和false。	-
defaultDeleteOperationWithValue	false	-	否	表示删除叶子节点时，生成的命令是否需要带上叶子的值。用于其他厂家设备的特殊情况，HUAWEI设备不需要，所以配成false。 配置值包括true和false。	-
supportsSinglelineCommand	false	-	否	表示创建、修改命令是否支持长命令模式。用于其他厂家设备的特殊情况，HUAWEI设备不支持，所以配置成false。 配置值包括true和false。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
cliInteractionMatchKey	Warning.*(base on this interface, continue|BGP process will be deleted. Continue).* y(es)/n(o)	-	否	交互匹配信息正则表达式，用于下发设备命令时，解析设备返回提示信息，需要用户通过命令做出交互选择。支持分号分隔多个正则表达式。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:
cliInteractionGoOnCommond	y	-	否	交互匹配信息响应，用于解析出设备交互匹配信息时，对设备进行响应，与cliInteractionMatchKey配置项组合使用。支持分号分隔多个正则表达式，分号数量需要跟cliInteractionMatchKey对应。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:y Info: Removing directory cfcard:/1.txt/.....Done! <HUAWEI123>
judgeContinueCondition	.*--- More ---*	-	否	判断继续条件的正则表达式，用于解析返回报文是否包含继续条件正则表达式。常用于查询的配置没有显示完整的情况。	[~HUAWEI123]dis current-configuration !Software Version V800R020C00SPC200B336 ---- More ----
partialReadSubmode	false	-	否	部分读取子模式，用于配置是否支持通过进入子视图，然后输入一条命令，查询部分配置。例如，HUAWEI支持进入相应视图，display this查询当前配置，就配置为true。 配置值包括true和false。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
continueReadingCommand	\u0020	-	否	继续读取命令，用于查询设备配置时，设备没有全部返回查询的配置，通过该命令，让设备继续返回配置。	[~HUAWEI123]dis current-configuration !Software Version V800R020C00SPC200B336 !Last configuration was updated at 2021-09-02 07:30:54+00:00 by huawei [~HUAWEI123]
changePasswordPrompt	^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*	-	否	修改密码提示正则表达式，用于解析与设备建连时设备要求修改密码的提示信息。	Password: Warning: The initial password poses security risks. The password needs to be changed. Change now? [Y/N]:
skipChangePasswordCmd	N	-	否	连接提示更改密码时下发的命令，用于解析出changePasswordPrompt提示信息时，下发设备的命令。	The password needs to be changed. Change now? [Y/N]:n <HUAWEI123>
cliNoPrintCommandPattern	cliNoPrintCommandPattern=change password.*;set key.*	-	否	非打印命令模式，用于打印日志时过滤敏感命令，支持以分号分隔，默认空不过滤敏感命令。	-
channelOpeningTimeout	10000	10000	否	通道建连超时时间，用于防止通道建连异常长时间空耗资源。单位ms。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
readMessageTimeout	10000	10000	否	通道读取返回报文超时时间，用于防止通道异常长时间读取报文。单位ms。	-
cliInteractionCommand	-	-	否	批量下发配置时可能出现交互的命令，用于提前识别批量配置需要交互的命令。以分号分隔，可配置多条命令。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:
ifBatchSendCommand	off	off	否	是否启用拼接命令方法的批量下发配置，用于其他厂家设备支持启用拼接命令方法批量下发配置时。 配置值包括on和off。	-
cliCmdAppender	0	0	否	命令回车符，用于解析命令行中的回车符。通常回车符为\r 或\n 。 配置为0表示\n ；配置为1表示\r。	-
cliIndentSpace	\u0020\u0020\u0020\u0020	\u0020\u0020\u0020\u0020	否	子命令行缩进，用于下发进入视图命令时，子命令行缩进量。	-
rangeCommand	batch	-	否	批量命令行的范围命令，用于render-batch-list-syntax扩展字使用的配置项。	-
rangeIndicator	\u0020to\u0020	-	否	批量命令行的范围标识，用于render-batch-list-syntax扩展字使用的配置项。	-
rangeSeparator	\u0020	-	否	批量命令行的分隔符，用于render-batch-list-syntax扩展字使用的配置项。	-

版权所有 © 华为技术有限公司

5.2.2.3.3.2 设置业务自定义处理（可选）

YANG/CLI默认转换逻辑和基础扩展项都无法满足YANG/CLI时，用户自定义代码实现CLI转YANG和YANG转CLI的逻辑。

用户自定义转换逻辑，在YANG模型被标记cli-custom-transform或cli-custom-read标签的节点上使用。详细请参考**13.2.3 CLI自定义扩展**。

业务自定义处理代码示例如下。

```
# 导入CliCustomTransformOutput接口，重写业务自定义处理方法需要使用
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
# 导入cliproxy接口，重写CommandProxy方法需要使用
from aoc.sys import cliproxy


    # 自定义代码实现cli转yang逻辑
    def cliToYang(self, aoccontext, request):
        self.logger.info('cliToYang start.')
        self.logger.info('yangToCli current path : %s' % request.path)
        if request.path == '/huawei-ifm:interfaces':
            return CliToYangCustom.clitoyang_interfaces(request, self.logger)
        self.logger.info('cliToYang end.')


    """
    在huawei-ifm.yang文件中，节点标记了“one-step-no-parse”自定义扩展，SND包中的cliToYang方法只要返回CLI配置报文即可，无需做到CLI到yang的转换。在SND包cliToYang方法中编写自定义代码，通过CommandProxy代理类给设备下发自定义read命令，获取设备返回配置报文后，做适当数据处理，最后返回CLI配置报文给CLIDriver，CLIDriver负责根据yang模型定义，将此CLI配置报文转换成yang数据。
    """
    def clitoyang_interfaces(request, logger):
        out = CliCustomTransformOutput()
        proxy = CommandProxy(request.neId, logger)
        # 向设备发送命令行，进入系统视图
        proxy.send_command(['return', 'system-view'])
        # 向设备发送命令行，查询包含interface字段的配置内容
        response_data = proxy.send_command(['display current-configuration | section include ^interface'],
120)
        # 获取设备回显内容
        response_body = response_data.response if hasattr(response_data, 'response') else response_data
        logger.info('clitoyang interfaces response_body: %s' % response_body)
        # 对设备回显内容进行处理，获取设备上已有的interface相关配置
        lines = response_body.splitlines()
        logger.info(lines)
        out.data = '<interfaces xmlns=\'http://huawei.com/ifm\'>\n'
        for line in lines:
            logger.info(line)
            # 去除回显结果中的显示命令与interface loopback相关配置
            if 'interface' in line and 'display' not in line and 'loopback' not in line:
                out.data += '<interface>\n'
                out.data += '<name>' + line.split()[1] + '</name>\n'
                str = line.split()[1]
                out.data += '<description>' + str[-1] + '</description>\n'
                out.data += '<shutdown>' + 'false' + '</shutdown>\n'
                out.data += '</interface>\n'
        out.data += '</interfaces>'
        out.endOffset = -1
        logger.info('yangToCli vlan-conf out.data %s' % out.data)
        return out


# 自定义实现类
class CommandProxy(object):
    """
    invoke java, send command to device
    """

    def __init__(self, ne_id, logger):
```


版权所有 © 华为技术有限公司

```
    logger.info("build CommandProxy for " + ne_id)
    self.ne_id = ne_id
    self.logger = logger

def send_command(self, commands, timeout=20):
    """
    向设备发送命令行
    :param commands: 命令行列表 string
    :param timeout: 超时时间
    :return: NONE
    """
    for command in commands:
        try:
            return cliproxy.sendCli(self.ne_id, command, timeout)
        except Exception as e:
            self.logger.error(e)
            return ""

def send_command_list(self, commands, timeout=20):
    """
    向设备发送命令行合集并获取回显
    :param commands: string
    :param timeout: 超时时间
    :return: SendCliOutput
    """
    response = []
    for command in commands:
        try:
            response.append(cliproxy.sendCli(self.ne_id, command, timeout))
        except Exception as e:
            self.logger.error(e)
            return ""
    return response[-1]
```

5.2.2.3.3.3 设置数据前置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异，如网络和设备枚举值定义可能不一致。设置前置处理后，当数据从设备同步NCE的时候，可以对数据进行处理，从而使NCE能够识别数据。

用户自定义前置后置处理逻辑，在YANG模型被标记cli-custom-processor标签的节点上使用。详细请参考**13.2.3 CLI自定义扩展**。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPreProcess字段。

数据前置处理代码示例如下。

```
# 导入CliCustomTransformOutput接口，重写数据前置处理方法需要使用
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput


def readCliPreProcessor(self, aoccontext, request):
    # 获取设备返回的命令行
    req_data = str(request.data)

    # 对命令行进行自定义处理
    if "c1 leaf1 value1" in req_data:
        req_data.replace("c1 leaf1 value1", "c1 leaf1 value2")

    # 构建返参proto对象
    out = CliCustomTransformOutput()
    out.data = req_data
    return out
```


版权所有 © 华为技术有限公司

5.2.2.3.3.4 设置数据后置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异。设置后置处理后，当数据从NCE同步到设备的时候，可以对数据进行处理，从而使设备能够识别数据。

用户自定义前置后置处理逻辑，在YANG模型被标记cli-custom-processor标签的节点上使用。详细请参考**13.2.3 CLI自定义扩展**。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPostProcess字段。

数据后置处理代码示例如下。

```
# 导入CliCustomTransformOutput接口，重写数据后置处理方法需要使用
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput


    def configCliPostProcessor(self, aoccontext, request):
        # 获取需要后置处理的命令行数据
        req_data = request.data
        req_path = request.path

        # 获取请求Path，根据不同Path，对data做不同处理
        if req_path == '/path':
            req_data = req_data + 'a'
        else:
            req_data = req_data + 'b'

        # 构建返参对象
        out = CliCustomTransformOutput()
        out.data = req_data
        return out
```

5.2.2.3.3.5 设置数据联动（可选）

给设备下发数据的时候，设备可能会进行数据的联动处理，此时需要NCE也做数据联动处理，保持NCE与设备的同步。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedEcsDBHook字段。

数据联动代码示例如下。

```
# 导入EcsDbConfigOut, EcsDbConfig接口，重写dbPostProcess方法配置数据联动时使用
from aoc.snd.snd_model_pb2.ecsDbProcess_pb2 import EcsDbConfigOut, EcsDbConfig


    # 重写dbPostProcess方法
    def dbPostProcess(self, aoccontext, request):
        ecsData = request.data
        ecsPath = request.path
        ecsopType = request.opType
        ecsDbConfigOut = EcsDbConfigOut()
        if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/huawei-ifm:service-name"
        elif "huawei-ifm:l2vc/secondary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig1 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig2 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig1.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/reroute"
            ecsDbconfig2.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/redundancy"
        return ecsDbConfigOut
```


版权所有 © 华为技术有限公司

5.2.2.3.3.6 获取 syncId（可选）

syncId是数据更改的记录ID。通过比较设备上syncId和NCE上的syncId，可以了解设备上数据是否与NCE一致。例如，用户通过命令行对设备下发过业务，就可能会造成数据不一致。当数据不一致时，可以通过同步或对账对齐数据。

在向设备下发业务数据过程中检查设备和控制器的数据是否一致时，则需要在设备驱动配置common_driver.checkSync = True，并重写getSyncId方法获取syncId。

获取syncId代码示例如下。

```
# CLI命令方式查询获取syncId示例
from tool.CommandProxy import CommandProxy


    def getSyncId(self, aoccontext, request):
        self.logger.info('getSyncId start.')
        # 读配置
        proxy = CommandProxy(request.deviceId, self.logger)
        response_data = proxy.send_command_list([ 'return', 'system-view','diagnose','display configflowid'],
120)
        response_body = response_data.response if hasattr(response_data, 'response') else response_data
        lines = response_body.splitlines()
        result = lines[4].strip().split(' ')[0]
        syncIdInfoOutput = SyncIdInfoOutput()
        syncIdInfoOutput.SyncId = result
        self.logger.info(result)
        self.logger.info('getSyncId end.%s' % (syncIdInfoOutput.SyncId))
        return syncIdInfoOutput
```

5.2.2.3.4 编写设备特性能力定制（可选）

当用户由于业务需要，可能会对差异发现范围、对账数据、同步数据进行定制处理，则需要编写设备特性能力定制代码。如果没有定制，则会对所有的特性进行全量差异发现。

特性能力定制代码示例如下。

```
# 导入FeatureCfgsMsg, Feature, Function接口，重写编写设备特性能力方法时使用
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function


    # 重写getFeatures方法，新增特性定制
    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True

         # 新增一个特性定制，编写差异发现的路径
        feature_msg.features.extend(self.build_feature('huawei-l3vpn', 'huawei-rtp', '(http://www.huawei.com/
netconf/vrp/huawei-l3vpn?revision=2018-06-11)l3vpn'))
        self.logger.info('getFeatures end.')
        return feature_msg


    def build_feature(self, name, depends, path):
        feature = Feature()

        # 特性名称
        feature.name = name

        # 操作类型
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()

        # 设置差异比较路径
```


版权所有 © 华为技术有限公司

```
    function.value = path

    # 设置从南向设备采集数据的路径
    function.collectPath = path
    feature_name = name.replace('huawei-', '')

    # 设置对账前定制，设置为“True”，则调用重写的preSyncToNe过程，设置为“False”，则不定制
    function.preSyncToNe = True

    # 设置同步前定制，设置为“True”，则调用重写的preSyncFromNe过程，设置为“False”，则不定制
    function.preSyncFromNe = False

    # 设置同步后定制，设置为“True”，则调用重写的postSyncFromNe过程，设置为“False”，则不定制
    function.postSyncFromNe = False
    feature.functions.extend([function])
    return [feature]


# 重写对账前定制过程
def preSyncToNe(self, request, aoccontext=None):
    self.logger.info('preSyncToNe start.')
    dataIn = request

    # 获得业务注册的数据路径
    regPath = dataIn.regPath

    # 获得特性名称
    featureName = dataIn.feature

    # 获得事务id
    transId = dataIn.transId
    self.logger.info("preSyncToNe begin regPath ={}, feature={}, transId={}", regPath, featureName, transId)

    # 获得业务处理后返回的差异数据
    diffDatas = dataIn.diffDatas

    # 获得差异数据列表的容量
    diffSize = len(diffDatas)

    # 如果差异为null或者差异数据容量为0，表明没有差异，函数直接返回null
    if diffDatas is None or diffSize == 0:
        return None

    # 定制数据一致性对账使用的数据
    dataOut = DiffDataOutMsg()

    # 遍历差异数据列表，处理差异数据
    for data in diffDatas:
        # 获得xml格式的差异数据
        diffXml = data

        # TODO 处理diffXml
        diffDataOut = diffData()

        # 该差异数据是否需要单独提交到南向设备。false表示不需要单独提交到南向设备
        diffDataOut.singleMsg = False
        diffDataOut.diffData = diffXml
        dataOut.diffDatas.extend([diffDataOut])
    self.logger.info('preSyncToNe end.')
    return dataOut
```

说明

新老设备被同一款SND包管理，不同设备支持的特性范围不一致时，则会导致同步出现异常。例如：SND包支持管理A、B两个特性，老设备仅支持A特性，新设备支持A和B特性，系统发起同步时，会尝试同时同步两个特性，系统会报老设备同步异常。

5.2.3 验证原子驱动包

版权所有 © 华为技术有限公司

5.2.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

5.2.4 编译原子驱动包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件复制到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

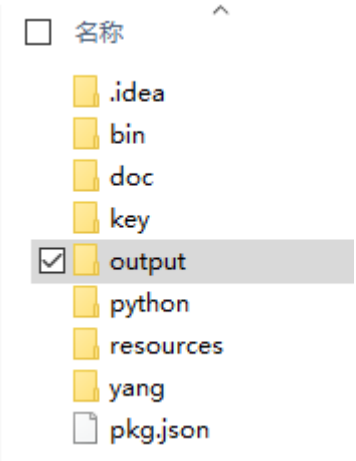
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
```


版权所有 © 华为技术有限公司

```
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

5.2.5 开发 CLI SND 示例

5.2.5.1 开发 CX600 CLI SND 示例

本示例开发华为CX600 CLI SND。华为CX600设备使用基础扩展项无法满足YANG/CLI转换，需要自定义开发扩展项语法，并在对应节点使用自定义扩展项。SND需要编写设备纳管定制代码、设备驱动数据代码、设备业务自定义处理、设备特性能力定制。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表5-20设置软件包的相关属性。

   表 5-20 CX600 SND 属性

属性名称	属性值
名称	CX600_SND
版本	1.0.0
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	CX600
协议类型	CLI

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

版权所有 © 华为技术有限公司

4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
    "name": "CX600_SND",
    "version": "1.0.0",
    "description": "CX600_SND",
    "package-type": "snd",
    "producer": "HUAWEI",
    "nce-min-versions": [
        "1.0.0"
    ],
    "snd-id": "CX600_SND",
    "devices": [
     {
       "vendor": "HUAWEI",
       "device-type": "CX600",
       "device-version": "V800R018C10SPC612B612"
     }
    ],
    "hooks": [
     {
       "type": "snd",
       "key": "ecs-driver",
       "python-class-name": "com.huawei.controller.devicetype.cx600snd.CX600CLI"
     }
    ]
}
```

步骤3 编写设备YANG模型

下面以创建子接口相关命令分析其对应的YANG模型。

表 5-21 分析 CLI 命令并设计 YANG 模型

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
ip vpn-instance 2G-RAN	ip：前缀，转换为container节点。	无。
	vpn-instance：子视图，转换为list节点。	无。
	2G-RAN：命令参数值，转换为leaf节点。	不显示leaf节点名称，需要annotations定制。
ipv4-family	ipv4-family：子视图，因为有许多子节点，所以转换为container节点。	container不能进入视图，进入子视图需要annotations定制。
route-distinguisher 100:1	route-distinguisher 100:1：单个命令，转换为leaf节点。	删除命令需要包含leaf的值，需要annotations定制。
tnl-policy LDP	tnl-policy LDP：单个命令，转换为leaf节点。	无。

 版权所有 © 华为技术有限公司

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
apply-label per-instance	apply-label per-instance：单个命令，转换为leaf节点。	删除命令需要包含leaf的值，需要annotations定制。
vpn-target 100:11 export-extcommunity vpn-target 100:11 import-extcommunity	vpn-target：前缀，转换为list节点。	删除命令需要包含leaf的值，需要annotations定制。
	100:11：命令参数值，转换为leaf节点。	通过annotations定制，将list中所有leaf节点转换成一行命令。 需要annotations来定制，不进入视图。
	export-extcommunity：命令参数值，转换为leaf节点。	无。

输出的完整YANG文件如下：

```
module huawei-ip {

    yang-version 1;

    namespace "http://huawei.com/ip";

    prefix ip;

    import cli_annotations {
        prefix "hw";
    }

    revision "2019-09-01" {
        description "Specific NE Driver for ip vpn-instance cli";
    }


    container ip {
        hw:custom-read 'one-step-no-parse';
        list vpn-instance {
            key vpn-instance-name;
            leaf vpn-instance-name {
                hw:render-name false;
                type string;
            }
            leaf description {
                type string;
            }

            container ipv4-family {
                hw:render-view true;
                leaf route-distinguisher {
                    hw:render-undo-value true;
                    type string {
                       length "3..21";
                    }
                }
```


版权所有 © 华为技术有限公司

```
            leaf tnl-policy {
                type string;
            }

            leaf apply-label {
                hw:render-undo-value true;
                type enumeration {
                    enum per-instance;
                    enum per-route;
                }
            }
            list vpn-target {
                hw:render-single-line;
                hw:render-view false;
                key "name mode";
                leaf name {
                    hw:render-name false;
                    hw:render-undo-value true;
                    type string {
                        length "3..21";
                    }
                }
                leaf mode {
                    hw:render-name false;
                    hw:render-undo-value true;
                    type enumeration {
                        enum "export-extcommunity";
                        enum "import-extcommunity";
                        enum "both";
                    }
                }
            }
        }
    }

}
```

步骤4 开发代码

1. 编写设备纳管定制

   a. 注册设备sysoid信息

```
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class NE8000M8NETCONF(NetconfSND):
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.87.27"
    sysoidEntity.deviceType = "ROUTER"
    sysoidEntity.deviceModel = "CX600"
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

   b. 配置设备的连接能力信息

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    primaryConnectInfo = connectInfos.connectInfo.add()
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
```

 版权所有 © 华为技术有限公司

```
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

2. 编写设备类型能力定制

   a. 配置设备通用驱动数据

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
    return common_driver
```

   b. 配置设备协议类驱动数据

   自定义逻辑，读取cli-driver.properties文件中的配置。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo
import os
import codecs

    def getCliDriverInfo(self, aoccontext, request=None):
        self.logger.info('getCliDriverInfo start.')
        cliDriverInfo = CliDriverInfo()
        self.logger.info(self.get_dict())
        for key, value in self.get_dict().items():
            cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
            cliDriverEntity.key = key
            cliDriverEntity.value = value
        self.logger.info('getCliDriverInfo end.')
        return cliDriverInfo

    def get_dict(self):
        try:
            path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
            self.logger.info(self.resourceDir)
            self.logger.info(path)
            pro_file = open(path, 'r')
            properties = {}
            for line in pro_file:
                if line.find('=') > 0:
                    strs = line.replace('\n', '').split('=')
                    properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
        except IOError as e:
            self.logger.info('getCliDriverInfo start.')
            self.logger.info(e)
            return {}
        else:
            pro_file.close()
        self.logger.info(properties)
        return properties
```

   cli-driver.properties文件中通过配置项定制CLI设备驱动驱动信息。

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2019-2019. All rights reserved.
#

#CLI协议使用，正则表达式：用户模式命令回显匹配规则
userModePrompt=^<.*>$
#CLI协议使用，正则表达式：特权模式命令回显匹配规则
privilegedModePrompt=^\\[.*\\]$|.*ontinue\\? \\[Y\\/N\\].*
#CLI协议使用，正则表达式，正确的配置该正则可以帮助CLI客户端在接收到与命令回显匹配规则匹配的返回时不中断且继续接收返回报文
promptSkipRegex=(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}(@prompt@){1}$
#CLI协议使用，切换特权模式命令

userNameResponse=Login:
```


版权所有 © 华为技术有限公司

```
#CLI协议使用，正则表达式：匹配登录输入密码的回显
passwordResponse=Password:
#CLI协议使用，建连后初始化命令
initCommand=display clock
#CLI协议使用，连接保活命令，建议使用轻量级短时响应的命令
keepAliveCommand=dis clock utc
cliTxCreateCommand=return \n
cliTxCommitCommand=commit \n
cliTxCancelCommand=clear configuration candidate \n
cliTxCancelInterval=2000
cliTxSaveCommand=return\nsave\ny\n
abortCommand=quit
#CLIDriver使用，设备事务异常时，取消事务重试次数。未配置则默认配置3次。非必须配置
cliTxCancelRetryTimes=3
#CLI协议使用，连接超时时间，单位ms，未配置则默认为10000ms。非必须配置
connectionTimeout=10000
#CLI协议使用，发送保活命令进行保活的间隔时间，单位为ms，未配置则默认为20000ms。非必须配置
keepAliveInterval=120000
#CLI协议和CLIDriver都使用，正则表达式：下发命令给设备时，设备的回显信息如果匹配上该正则，则表示配置失败
errorMessagePattern=^([\\s\\S]*)(Error.*:|Info: The interval between portswitch).*
#CLI协议使用，下发命令给设备时，设备的回显信息如果匹配上该正则，则表示应该忽略1
ignoreMessagePattern=randomvalueshould#not#match
warningMessagePattern=^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*
#CLI协议使用，限制最大返回报文长度，单位为KB，未配置则默认为400KB。非必须配置
maxResponseSize=400
showConfigCommand=return\ndisplay current-configuration
#CLIDriver使用，正则表达式：用于过滤配置的查询结果，用于将非设备配置的回显信息滤除
showConfigSkipRegex=^(--|echo|#).*$

#CLIDriver使用，下发实际配置命令（创建、删除、更新等）前，通过该命令进入配置模式
configModeCommand= return\nsystem-view
#CLIDriver使用，查询设备部分配置的命令模式，例子中的${context}表示CLIDriver根据Yang模型生成的查询命令
partialReadCommand=return\nsystem-view\ndisplay current-configuration configuration ${context}
#CLIDriver使用，左缩进，建议采用典型配置：四个空格键。未配置则默认是4个空格。非必须配置
indentationSequence=\u0020
cliIndentSpace=\u0020
#CLIDriver使用，返回上级视图的命令，如quit、exit
exitSubmodeCommand=quit
submodeStartSequence=@indent@
submodeEndSequence=@dedent@
#CLIDriver使用，删除命令的前缀，如no，undo等。示例中的\u0020是空格键对的unicode码。
negateCommand=undo\u0020
#CLIDriver使用，shutdown对应的命令。未配置则默认为\u0020shutdown。示例中的\u0020是空格键对的unicode码。非必须配置。
shutdownCommand=\u0020shutdown
contextSupportsSinglelineCommand=false
#CLIDriver使用，表示删除叶子节点时，生成的命令是否需要带上叶子的值。HUAWEI设备不需要，所以配成false
defaultDeleteOperationWithValue=false
supportsSinglelineCommand=false
#CLIDriver使用，正则表达式：匹配下发设备命令时，设备返回提示的信息，需要用户根据实际情况，通过命令做出选择的
cliInteractionMatchKey=Warning.*(base on this interface, continue| BGP process will be deleted. Continue).*
#CLIDriver使用，与上一条结合使用，表示用户对于设备的提示信息，做出的响应
cliInteractionGoOnCommond=Y
#CLI协议使用，正则表示式：查询设备配置时，该配置表示查询的配置没有显示完整
judgeContinueCondition=.*--- More ---.*
#CLIDriver使用，是否支持通过进入子视图，然后输入一条命令，查询部分配置。例如，HUAWEI支持进入相应视图，display this查询当前配置。所以配置为true
partialReadSubmode=false
#CLI协议使用，查询设备配置时，如果设备没有全部返回查询的配置，可以通过该命令，让设备继续返回配置。示例中的\u0020是空格键对的unicode码
continueReadingCommand=\u0020
changePasswordPrompt=^The password needs to be changed\\. Change now\\? \\[Y\\/N\
```


版权所有 © 华为技术有限公司

```
\].*;^Are you sure you want to continue connecting \\(yes\\/no\\)\\?.*
skipChangePasswordCmd=N;yes
# 匹配该规则的命令不会在日志中打印，例如：change password.*;set key.*
cliNoPrintCommandPattern= change password.*; set key.*
```

c. 设置业务自定义处理

自定义代码开发扩展项语法

```
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
from aoc.sys import cliproxy

    def cliToYang(self, aoccontext, request):
        self.logger.info('cliToYang start.')
        self.logger.info('yangToCli current path : %s' % request.path)
        if request.path == '/huawei-ifm:interfaces':
            return CliToYangCustom.clitoyang_interfaces(request, self.logger)
        self.logger.info('cliToYang end.')

    def clitoyang_interfaces(request, logger):
        out = CliCustomTransformOutput()
        proxy = CommandProxy(request.neId, logger)
        proxy.send_command(['return', 'system-view'])
        response_data = proxy.send_command(['display current-configuration | section include
^interface'], 120)
        response_body = response_data.response if hasattr(response_data, 'response') else
response_data
        logger.info('clitoyang interfaces response_body: %s' % response_body)
        lines = response_body.splitlines()
        logger.info(lines)
        out.data = '<interfaces xmlns=\'http://huawei.com/ifm\'>\n'
        for line in lines:
            logger.info(line)
            if 'interface' in line and 'display' not in line and 'loopback' not in line:
                out.data += '<interface>\n'
                out.data += '<name>' + line.split()[1] + '</name>\n'
                str = line.split()[1]
                out.data += '<description>' + str[-1] + '</description>\n'
                out.data += '<shutdown>' + 'false' + '</shutdown>\n'
                out.data += '</interface>\n'
        out.data += '</interfaces>'
        out.endOffset = -1
        logger.info('yangToCli vlan-conf out.data %s' % out.data)
        return out

# 自定义实现类
class CommandProxy(object):
    """
    invoke java, send command to device
    """

    def __init__(self, ne_id, logger):
        logger.info("build CommandProxy for " + ne_id)
        self.ne_id = ne_id
        self.logger = logger

    def send_command(self, commands, timeout=20):
        """
        向设备发送命令行
        :param commands: 命令行列表 string
        :param timeout: 超时时间
        :return: NONE
        """
        for command in commands:
            try:
                return cliproxy.sendCli(self.ne_id, command, timeout)
            except Exception as e:
                self.logger.error(e)
                return ""

    def send_command_list(self, commands, timeout=20):
```


版权所有 © 华为技术有限公司

```
        """
        向设备发送命令行合集并获取回显
        :param commands: string
        :param timeout: 超时时间
        :return: SendCliOutput
        """
        response = []
        for command in commands:
            try:
                response.append(cliproxy.sendCli(self.ne_id, command, timeout))
            except Exception as e:
                self.logger.error(e)
                return ""
        return response[-1]
```

3. 编写设备特性能力定制

   定义差异发现范围，只对http://huawei.com/ifm?revision=2019-09-01)interfaces特性进行差异发现。

```
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function

    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True
        feature_msg.features.extend(self.build_feature('ifm', '', '(http://huawei.com/ifm?
revision=2019-09-01)interfaces'))
        self.logger.info('getFeatures end.')
        return feature_msg

    def build_feature(self, name, depends, path):
        feature = Feature()
        feature.name = name
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()
        function.value = path
        function.collectPath = path
        function.preSyncToNe = False
        function.preSyncFromNe = False
        function.postSyncFromNe = False
        feature.functions.extend([function])
        return [feature]
```

----结束

5.2.5.2 开发 NE5000E CLI SND 示例

本示例开发华为NE5000E CLI SND。华为NE5000E 设备使用基础扩展项无法满足YANG/CLI转换，需要自定义开发自己的扩展项语法，并在对应节点使用自定义扩展项。SND需要编写设备纳管定制代码、设备驱动数据代码、设备业务自定义处理、设备特性能力定制。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据**表5-22**设置软件包的相关属性。

版权所有 © 华为技术有限公司

表 5-22 NE5000E SND 属性

属性名称	属性值
名称	SND-CLI-HUAWEI-NE5000E-1
版本	4.0.1
提供商	Huawei
包类型	Specific NE driver
映射类型	python
设备类型	NE5000E
协议类型	CLI

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "SND-CLI-HUAWEI-NE5000E-1",
   "version": "4.0.1",
   "description": "",
   "package-type": "snd",
   "producer": "Huawei",
   "snd-id": "SND_CLI_HUAWEI_NE5000E_1",
   "nce-min-versions":[
       "1.0.0"
   ],
    "devices": [
      {
        "vendor": "Huawei",
        "device-type": "NE5000E.*",
        "device-version": "V800R009C10.*"
      },
      {
        "vendor": "Huawei",
        "device-type": "NE5000E.*",
        "device-version": "V800R011C10.*"
      },
      {
        "vendor": "Huawei",
        "device-type": "NE5000E.*",
        "device-version": "V800R021C00.*"
      }
   ],
   "hooks": [
     {
       "type": "snd",
       "key": "ecs-driver",
       "python-class-name": "com.huawei.ne5000e.cuii.cuanetcli.CUANETCLI"
     }
   ]
}
```

步骤3 编写设备YANG模型

下面以创建evpn相关命令分析其对应的YANG模型。

 版权所有 © 华为技术有限公司

表 5-23 分析 CLI 命令并设计 YANG 模型

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
evpn vpn-instance VPWS-HW-110 vpws	evpn：进入子视图，转换为list节点。vpws 指定EVPN实例的模式，以vpn-instance-name与实例模式为key值可以创建多个EVPN，所以使用vpn-instance与vpws作为list的复合key。	one-step-no-parse表示该节点在处理命令行回显时不通过Yang文件做CLI到YANG的转换，由自定义的cliToYang方法直接定义命令行回显的处理方式。
	vpn-instance：使用*vpn instance-name*作为参数，转换为leaf节点，类型为string。	render-name true 表示leaf节点的名字vpn-instance需要在CLI命令中显示
	vpws：命令参数值，转换为leaf节点。类型为枚举，本例中只使用vpws模式，枚举类型填写为单一的vpws即可。	不显示leaf节点名称，需要annotations定制。
description VPWS-HW-110_TEST	description：命令参数值，转换为leaf节点。	无。
route-distinguisher 100:101	route-distinguisher 100:1：单个命令，转换为leaf节点。	render-undo-value true 表示删除命令需要包含leaf的值。
vpn-target 100:102 export-extcommunity vpn-target 100:102 import-extcommunity	vpn-target：前缀，转换为list节点。	render-view false 表示抑制进入list子视图。
	100:11：命令参数值，转换为leaf节点。类型为string。	无。
	export-extcommunity：命令参数值，转换为leaf节点。类型为枚举，值为export-extcommunity或import-extcommunity。	无。

输出的完整YANG文件如下：

```
/*
Copyright (C) 2021 Huawei Technologies Co., Ltd. All rights reserved.
*/
module huawei-evpn {
   yang-version 1;
   namespace "urn:huawei:yang:huawei-evpn";
   prefix evpn;
   import cli_annotations {
     prefix "cli";
   }
   revision 2021-05-26 {
      description "Initial revision.";
```


版权所有 © 华为技术有限公司

```
    }

    list evpn {
        cli:custom-read 'one-step-no-parse';
        key "vpn-instance vpws";
        leaf vpn-instance {
            cli:render-name true;
            type string {
                length "1..31";
            }
        }
        leaf vpws {
            type enumeration {
                enum vpws {
                    value 0;
                }
            }
        }

        leaf route-distinguisher {
            cli:render-undo-value true;
            type string {
                length "3..21";
            }
        }

        list vpn-target {
            cli:render-view false;
            key "number vt-type";
            leaf number {
                type string {
                    length "3..21";
                }
            }
            leaf vt-type {
                type enumeration {
                    enum "export-extcommunity" {
                        value 0;
                    }
                    enum "import-extcommunity" {
                        value 1;
                    }
                }
            }
        }

        leaf description {
            type string {
                length "1..63";
            }
        }
    }
}
```

步骤4 开发代码

1. 编写设备纳管定制

   a. 注册设备sysoid信息

```
from aoc.snd.clisnd import CliSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class CUANETCLI(CliSND):
    def getSysoidInfo(self, aoccontext, request=None):
        self.logger.info("[CUANET-5000eCLI][getSysoidInfo] request is: %s, context is: %s"
                         % (request, aoccontext))
        sysoid_info = SysoidInfo()
        CUANETCLI.gen_sysoid(sysoid_info, "NE5000E(V8)", "1.3.6.1.4.1.2011.2.62.2.3.1")
        CUANETCLI.gen_sysoid(sysoid_info, "NE5000E", "1.3.6.1.4.1.2011.2.62.2.3")
        CUANETCLI.gen_sysoid(sysoid_info, "NE5000E-X16A",
```

 版权所有 © 华为技术有限公司

```
"1.3.6.1.4.1.2011.20021210.12.561231")
        return sysoid_info

    def gen_sysoid(sysoid_info, device_model, sys_oid):
        sysoid_entity = sysoid_info.sysoidEntity.add()
        sysoid_entity.deviceType = "ROUTER"
        sysoid_entity.deviceModel = device_model
        sysoid_entity.sysoid = sys_oid
        sysoid_entity.deviceVendor = "HUAWEI"
```

b. 配置设备的连接能力信息

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    primaryConnectInfo = connectInfos.connectInfo.add()
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

2. 编写设备类型能力定制

a. 配置设备通用驱动数据

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
        common_driver.checkSync = True
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "false"
    return common_driver
```

b. 配置设备协议类驱动数据

自定义逻辑，读取cli-driver.properties文件中的配置。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo
import os
import codecs

    def getCliDriverInfo(self, aoccontext, request=None):
        self.logger.info('getCliDriverInfo start.')
        cliDriverInfo = CliDriverInfo()
        self.logger.info(self.get_dict())
        for key, value in self.get_dict().items():
            cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
            cliDriverEntity.key = key
            cliDriverEntity.value = value
        self.logger.info('getCliDriverInfo end.')
        return cliDriverInfo


    def get_dict(self):
        try:
            path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
            self.logger.info(self.resourceDir)
            self.logger.info(path)
            pro_file = open(path, 'r')
            properties = {}
            for line in pro_file:
                if line.find('=') > 0:
                    strs = line.replace('\n', '').split('=')
```

 版权所有 © 华为技术有限公司

```
                properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
        except IOError as e:
            self.logger.info('getCliDriverInfo start.')
            self.logger.info(e)
            return {}
        else:
            pro_file.close()
        self.logger.info(properties)
        return properties
```

cli-driver.properties文件中通过配置项定制CLI设备驱动驱动信息。

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2021-2022. All rights reserved.
#
protocol=Ssh
userModeCommand=
defaultParser=basicPatternParser
keepAliveCommandParser=basicPatternParser
privilegedModeCommandAfterAbortCommand=false
userModePrompt=^<.*>$
privilegedModePrompt=^\\[.*\\]$|(.*ontinue\\? \\[Y\\/N\\])
promptSkipRegex=(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}
(@prompt@){1}$
privilegedModeCommand=
userNameResponse=Login:
passwordResponse=Enter password:
initCommand=display clock
keepAliveCommand=dis clock utc
cliTxCreateCommand=return\nsystem-view\n
cliTxCommitCommand=commit \n
cliTxCancelCommand=clear configuration candidate\n
cliTxSaveCommand=return \n
abortCommand=
cliTxCancelRetryTimes=3
connectionTimeout=20000
keepAliveInterval=120000
keepConnectedInterval=20000
cliType=
logData=false
errorMessagePattern=^Error:?[ ]{1}.*
ignoreMessagePattern=randomvalueshould#not#match
warningMessagePattern=^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*
maxNumberReadsChannels=1
maxResponseSize=400
showConfigCommand=return\ndisplay current-configuration
showConfigSkipRegex=^(--|echo|#).*$
resetCommand=return
submodeCommand=system-view ${context}
configModeCommand=\n
operModeCommand=display
partialReadCommand=return\nsystem-view\n${context}\ndisplay this
indentationSequence=\u0020
exitSubmodeCommand=quit
submodeStartSequence=@indent@
submodeEndSequence=@dedent@
negateCommand=undo\u0020
shutdownCommand=\u0020shutdown
operationalDataReadCommand=/display ${context}
contextAccessType=readDatastore
contextSupportsSinglelineCommand=false
defaultDeleteOperationWithValue=false
supportsSinglelineCommand=false
cliInteractionMatchKey=Warning.*(base on this interface, continue|The destination address and
the mask do not match.*will be deleted. Continue.*| BGP process will be deleted. Continue|
Changing the parameter in this command resets the peer session. Continue| BGP will be deleted.
Continue| BGP peer will be deleted. Continue| This operation will reset the peer session.
Continue| Changing the smallest VLAN ID of the interface will temporarily affect services on the
interface. Continue| This operation may cause an E-Trunk service interruption. Continue| All the
configurations related to the BGP peer will be deleted. Continue| If you want the configurations
to be restored, perform reconfigurations. Continue).*;^The password needs to be changed\\.
```


版权所有 © 华为技术有限公司

```
Change now\\? \\[Y\\/N\\].*;Are you sure you want to continue connecting.*
cliInteractionGoOnCommond=Y;N;yes
judgeContinueCondition=.*---- More ----.*
partialReadSubmode=false
continueReadingCommand=\u0020
changePasswordPrompt=^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*
skipChangePasswordCmd=N
Timeout=10000
```

c. 设置业务自定义处理

自定义代码开发扩展项语法

```
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
from aoc.sys import cliproxy

    def yangToCli(self, aoccontext, request):
        """
        yang转成cli
        :param aoc_context: dict, contains
        :param request: XML packet
        """
        self.logger.info("[CUANET-5000eCLI][yangToCli] request is: %s, context is: %s"
                         % (request, aoccontext))
        self.logger.info('yangToCli end.')

    def cliToYang(self, aoccontext, request):
        """
        查询命令行转yang
        自定义代码实现cli转yang逻辑
        :param request: XML packet
        :param aoc_context: dict, contains
        :return: 查询命令行对应的yang结构的xpath
        The aoc reads device data and converts cli packets to yang model data based on the
default conversion rules of the yang model.
        In some special scenarios,commands cannot be converted into yang data according to the
default rules of the yang model structure.
        users can customize the conversion implementation logic.
        """
        self.logger.info("[CUANET-5000eCLI][cliToYang] request is: %s, context is: %s"
                         % (request, aoccontext))
        self.logger.info('yangToCli current path : %s' % request.path)
        out = None
        if request.path == '/huawei-tunnel-path:explicit-path':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['dis current-configuration configuration explicit-path'])
        elif request.path == '/huawei-network-instance:ip':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger,
                ['dis current-configuration configuration vpn-instance'])
        elif request.path == '/huawei-ifm:interface':
            out = cli_to_yang_custom.clitoyang_blank(
                request, self.logger,
                ['dis current-configuration interface | section exclude interface Tunnel'])
        elif request.path == '/huawei-static-route:ip':
            out = cli_to_yang_custom.clitoyang_blank(
                request, self.logger, ['dis current-configuration configuration route-static'])
        elif request.path == '/huawei-bgp:bgp':
            out = cli_to_yang_custom.clitoyang_bgp(
                request, self.logger, ['dis current-configuration configuration bgp'])
        elif request.path == '/huawei-hqos:traffic-classifier':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['dis current-configuration configuration '
                                       '| section include traffic classifier'])
        elif request.path == '/huawei-hqos:traffic-policy':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['dis current-configuration configuration trafficpolicy'])
        elif request.path == '/huawei-hqos:flow-queue':
            out = cli_to_yang_custom.clitoyang_blank(
                request, self.logger, ['dis current-configuration configuration flow-queue'])
        elif request.path == '/huawei-hqos:qos-profile':
```


版权所有 © 华为技术有限公司

```
            out = cli_to_yang_custom.clitoyang_blank(
                request, self.logger, ['dis current-configuration configuration qos-profile'])
        elif request.path == '/huawei-tunnel:interface':
            out = cli_to_yang_custom.clitoyang_tunnel(
                request, self.logger, ['dis current-configuration interface Tunnel'])
        else:
            out = self.additional_cli_to_yang(request, out)

        self.logger.info('cliToYang end: %s' % Util.safe_str(out))
        return out

    def additional_cli_to_yang(self, request, out):
        """
        带查询条件的查询命令行转yang
        :param request: request
        :param out: out
        :return: 查询命令行对应的yang结构的xpath
        """
        if request.path == '/huawei-bfd-cfg:bfd':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger,
                ['display current-configuration | section include bind mpls-te'])
        elif request.path == '/huawei-tunnel-policy:tunnel-policy':
            out = cli_to_yang_custom.clitoyang_tunnel_policy(
                request, self.logger, ['dis current-configuration configuration tunnel-policy'])
        elif request.path == '/huawei-evpn:evpn':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger,
                ['display current-configuration configuration | section include evpn'])
        elif request.path == '/huawei-evpn:evpl':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['display current-configuration configuration evpl'])
        elif request.path == '/huawei-evpn-lacp:lacp':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger,
                ['display current-configuration configuration | include lacp'])
        elif request.path == '/huawei-evpn-ifm:interface':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['display current-configuration interface'])
        elif request.path == '/huawei-evpn-ifm:evpn':
            out = cli_to_yang_custom.clitoyang_evpn_ifm_evpn(
                request, self.logger,
                ['display current-configuration | section include evpn'])
        elif request.path == '/huawei-evpn-ifm:e-trunk':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['display current-configuration configuration e-trunk'])
        elif request.path == '/huawei-evpn-ifm:vlan':
            out = cli_to_yang_custom.clitoyang_evpn_vlan(
                request, self.logger, ['display current-configuration | include vlan'])
        elif request.path == '/huawei-hqos:traffic-behavior':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger,
                ['dis current-configuration configuration | section include traffic behavior'])
        else:
            out = self.additional_cli_to_yang2(request, out)
        return out

    def additional_cli_to_yang2(self, request, out):
        """
        全局场景下带查询条件的查询命令行转yang
        :param request: request
        :param out: out
        :return: 查询命令行对应的yang结构的xpath
        """
        if request.path == '/huawei-ip-prefix:ip':
            out = cli_to_yang_custom.clitoyang_common(
                request, self.logger, ['display current-configuration | include ip ip-prefix'])
        elif request.path == '/huawei-ip-community:ip':
            out = cli_to_yang_custom.clitoyang_ip_community(
```

 版权所有 © 华为技术有限公司

```
                request, self.logger,
                ['display current-configuration | include ip community-filter'])
        elif request.path == '/huawei-route-policy:route-policy':
            out = cli_to_yang_custom.clitoyang_route_policy(
                request, self.logger, ['display current-configuration configuration route-policy'])
        return out


# 自定义实现类
class CommandProxy(object):
    """
    invoke java, send command to device
    """

    def __init__(self, ne_id, logger):
        logger.info("build CommandProxy for " + ne_id)
        self.ne_id = ne_id
        self.logger = logger

    def send_command(self, commands, timeout=20):
        """
        向设备发送命令行
        :param commands: 命令行列表 string
        :param timeout: 超时时间
        :return: NONE
        """
        for command in commands:
            try:
                return cliproxy.sendCli(self.ne_id, command, timeout)
            except Exception as e:
                self.logger.error(e)
                return ""

    def send_command_list(self, commands, timeout=20):
        """
        向设备发送命令行合集并获取回显
        :param commands: string
        :param timeout: 超时时间
        :return: SendCliOutput
        """
        response = []
        for command in commands:
            try:
                response.append(cliproxy.sendCli(self.ne_id, command, timeout))
            except Exception as e:
                self.logger.error(e)
                return ""
        return response[-1]
```

d. 获取syncId

```
from tool.CommandProxy import CommandProxy

    def getSyncId(self, aoccontext, request):
        self.logger.info('[CUANET-5000eCLI] getSyncId start.%s' % aoccontext)
        proxy = CommandProxy(request.deviceId, self.logger)
        response_data = proxy.send_command_list(['return', 'system-view', 'diagnose',
                                                 'display configflowid'], 120)
        response_body = response_data.response if hasattr(response_data, 'response') \
            else response_data
        lines = response_body.splitlines()
        result = lines[4].strip().split(' ')[0]
        sync_id_info_output = SyncIdInfoOutput()
        sync_id_info_output.SyncId = result
        self.logger.info(result)
        self.logger.info('[CUANET-5000eCLI]getSyncId end.%s' % sync_id_info_output.SyncId)
        return sync_id_info_output
```

3. 编写设备特性能力定制

   自定义差异发现，对账和同步使用的特性范围。

 版权所有 © 华为技术有限公司

```
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function

  def getFeatures(self, aoccontext, request=None):
      self.logger.info("[CUANET-5000eCLI][getFeatures] request is: %s, context is: %s"
                       % (request, aoccontext))
      feature_msg = FeatureCfgsMsg()
      feature_msg.replace = True

      feature_msg.features.extend(
          self.build_feature(
              'explicit-path',
              '',
              '(urn:huawei:yang:huawei-tunnel-path?revision=2021-02-23)explicit-path'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'l3vpn',
              '',
              '(urn:huawei:yang:huawei-network-instance?revision=2021-02-24)ip'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'ifm',
              '',
              '(urn:huawei:yang:huawei-ifm?revision=2021-02-24)interface'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'bgp',
              '',
              '(urn:huawei:yang:huawei-bgp?revision=2021-03-08)bgp'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'srt',
              '',
              '(urn:huawei:yang:huawei-static-route?revision=2021-03-24)ip'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'traffic-policy',
              '',
              '(urn:huawei:yang:huawei-hqos?revision=2021-03-04)traffic-policy'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'flow-queue',
              '',
              '(urn:huawei:yang:huawei-hqos?revision=2021-03-04)flow-queue'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'qos-profile',
              '',
              '(urn:huawei:yang:huawei-hqos?revision=2021-03-04)qos-profile'
          )
      )
      feature_msg.features.extend(
          self.build_feature(
              'tunnel',
              '',
              '(urn:huawei:yang:huawei-tunnel?revision=2021-02-24)interface'
```


版权所有 © 华为技术有限公司

```
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'bfd',
            '',
            '(urn:huawei:yang:huawei-bfd-cfg?revision=2021-02-24)bfd'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'tunnel-policy',
            '',
            '(urn:huawei:yang:huawei-tunnel-policy?revision=2021-02-26)tunnel-policy'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'evpn',
            '',
            '(urn:huawei:yang:huawei-evpn?revision=2021-05-26)evpn'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'evpl',
            '',
            '(urn:huawei:yang:huawei-evpn?revision=2021-05-26)evpl'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'lacp',
            '',
            '(urn:huawei:yang:huawei-evpn-lacp?revision=2021-05-31)lacp'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'interface',
            '',
            '(urn:huawei:yang:huawei-evpn-ifm?revision=2021-05-26)interface'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'evpn-ifm',
            '',
            '(urn:huawei:yang:huawei-evpn-ifm?revision=2021-05-26)evpn'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'e-trunk',
            '',
            '(urn:huawei:yang:huawei-evpn-ifm?revision=2021-05-26)e-trunk'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'vlan',
            '',
            '(urn:huawei:yang:huawei-evpn-ifm?revision=2021-05-26)vlan'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'traffic-behavior',
            '',
```


版权所有 © 华为技术有限公司

```
            '(urn:huawei:yang:huawei-hqos?revision=2021-03-04)traffic-behavior'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'traffic-classifier',
            '',
            '(urn:huawei:yang:huawei-hqos?revision=2021-03-04)traffic-classifier'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'ip-prefix',
            '',
            '(urn:huawei:yang:huawei-ip-prefix?revision=2021-08-02)ip'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'ip-community',
            '',
            '(urn:huawei:yang:huawei-ip-community?revision=2021-08-02)ip'
        )
    )
    feature_msg.features.extend(
        self.build_feature(
            'route-policy',
            '',
            '(urn:huawei:yang:huawei-route-policy?revision=2021-08-02)route-policy'
        )
    )
    self.logger.info('getFeatures end:%s' % feature_msg)
    return feature_msg

def build_feature(self, name, depends, path):
    feature = Feature()
    feature.name = name
    feature.operType = Feature.MERGE
    feature.depends.extend([depends])
    function = Function()
    function.value = path
    function.collectPath = path
    function.preSyncToNe = False
    function.preSyncFromNe = False
    function.postSyncFromNe = False
    feature.functions.extend([function])
    return [feature]
```

----结束

5.2.5.3 开发 SG6000 CLI SND 示例

本示例开发山石SG6000 CLI SND。山石SG6000 防火墙设备使用基础扩展项无法满足YANG/CLI转换，需要自定义开发自己的扩展项语法，并在对应节点使用自定义扩展项。SND需要编写设备纳管定制代码、设备驱动数据代码、设备业务自定义处理、设备特性能力定制。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表5-24设置软件包的相关属性。

 版权所有 © 华为技术有限公司

表 5-24 SG6000 SND 属性

属性名称	属性值
名称	HillstoneFire-SND
版本	22.0.53
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	SG6000
协议类型	CLI

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "HillstoneFire-SND",
   "version": "22.0.53",
   "description": "snd",
   "package-type": "snd",
   "producer": "HUAWEI",
   "nce-min-versions": [
      "1.0.0"
   ],
   "snd-id": "HillstoneFire-SND",
   "devices": [
    {
      "vendor": "Hillstone",
      "device-type": "SG6000",
      "device-version": "5.5.0"
    }
   ],
   "hooks": [
    {
      "type": "snd",
      "key": "ecs-driver",
      "python-class-name": "com.huawei.controller.devicetype.Hillstonefiresnd.HillstonefireCLI"
    }
   ]
}
```

步骤3 编写设备YANG模型

下面以配置防火墙策略命令分析其对应的YANG模型。

表 5-25 分析 CLI 命令并设计 YANG 模型

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
address src	address ： 子视图，转换为list节点。	无。

 版权所有 © 华为技术有限公司

CLI命令	分析CLI命令，设计CLI命令字和YANG模型节点转换关系	annotations定制项
address dst	src/dst：子视图参数，转换为key leaf节点。	无。
ip 10.1.1.1/32 ip 10.2.2.2/32	ip：单个命令。 与range功能相似，也转换为list节点	不需要进入视图，通过annotations定制。
	10.1.1.1/32：1个参数，转换为key leaf节点	无。
range 10.125.252.193 10.125.252.198	range：单个命令，带2个参数。 有多个参数，转换为list节点。	不需要进入视图，通过annotations定制。
	10.125.252.193：第1个参数，转换为key leaf节点	无。
	101.125.252.198：第2个参数，转换为key leaf节点	无。
rule id 10	rule：前缀，转换为container 节点。	无。
	id：子视图，转换为list节点。	无。
	10：子视图参数，转换为key leaf节点。	无。
action deny	action deny：单个命令，转换为leaf节点。	无。
src-zone untrust dst-zone trust	src-zone untrust：单个命令，转换为leaf节点。	无。
src-addr src dst-addr dst	src-addr：转换为list节点。	不需要进入视图，通过annotations定制。
	src：转换为key leaf节点。	删除命令需要带上leaf参数值，通过annotations定制。
service any	同src-zone untrust	同src-zone untrust
name policy-test	name policy-test：单个命令，转换为leaf节点。	无。

输出的完整YANG文件如下：

```
module hillstone-policy {
  yang-version 1;
```

 版权所有 © 华为技术有限公司

```
namespace "https://www.hillstonenet.com.cn/hillstone-policy";
prefix access_policy;
import cli_annotations {
    prefix "ss";
}
revision "2022-02-17" {
    description "first version test";
}

container rule {
    ss:cli-custom-read "one-step-no-parse";
    ss:cli-custom-processor 'post-config';
    list id {
        key "id-number";
        leaf id-number {
            type uint16 {
                range "1..12256";
            }
        }
        leaf description {
            ss:allow-multi-token 10;
            type string;
        }
        leaf action {
            type enumeration {
                enum deny;
                enum permit;
            }
            default "deny";
        }
        leaf src-zone {
            type string;
        }
        leaf dst-zone {
            type string;
        }
        list src-addr {
            ss:render-view false;
            key "src-addr-name";
            leaf src-addr-name {
                ss:render-undo-value true;
                type string {
                    length "1..95";
                }
            }
        }
        list dst-addr {
            ss:render-view false;
            key "dst-addr-name";
            leaf dst-addr-name {
                ss:render-undo-value true;
                type string {
                    length "1..95";
                }
            }
        }
        list service {
            ss:render-view false;
            key "service-name";
            leaf service-name {
                ss:render-undo-value true;
                type string;
            }
        }
        leaf name {
            type string {
                length "1..95";
            }
        }
```


版权所有 © 华为技术有限公司

```
    }
  }
}
```

```
module hillstone-pool {
  yang-version 1;
  namespace "https://www.hillstonenet.com.cn/hillstone-pool";
  prefix hillstone-pool;
  import cli_annotations {
    prefix "hw";
  }
  revision "2022-02-17" {
    description "first version test";
  }
  import ietf-inet-types {
    prefix inet;
  }
  typedef quoted-string {
    type string;
  }

  container address-pool {
    hw:render-name false;
    hw:custom-read 'one-step-no-parse';
    hw:cli-custom-processor 'post-config';
    list address {
      key "address-name";
      leaf address-name {
        type quoted-string;
      }
      leaf description {
        hw:allow-multi-token 10;
        type string;
      }
      leaf member {
        hw:render-undo-value true;
        type string;
      }
      list ip {
        hw:render-view false;
        key "addr";
        leaf addr {
          type inet:ipv4-prefix;
        }
      }
      list range {
        hw:render-view false;
        key "address-start address-end";
        leaf address-start {
          type inet:ipv4-address-no-zone;
        }
        leaf address-end {
          type inet:ipv4-address-no-zone;
        }
      }
    }
  }
}
```

步骤4 开发代码

1. 编写设备纳管定制

   a. 注册设备sysoid信息

```
from aoc.snd.clisnd import CliSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class HillstonefireCLI(CliSND):
    def getSysoidInfo(self, aoccontext, request=None):
```

 版权所有 © 华为技术有限公司

```
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.87.27"
    sysoidEntity.deviceType = "FIREWALL"
    sysoidEntity.deviceModel = "SG6000"
    sysoidEntity.deviceVendor = "Hillstone"
    return sysoidInfo
```

b. 配置设备的连接能力信息

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
    self.logger.info('getConnectInfo start.')
    connectInfos = ConnectInfos()
    primaryConnectInfo = connectInfos.connectInfo.add()
    primaryConnectInfo.connectPolicy = DEFAULT_CONNECT
    primaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
    primaryConnectInfo.channelInfo.is_read_share_write = True
    primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli
    primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

2. 编写设备类型能力定制

a. 配置设备通用驱动数据

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

    def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()
    common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
    syncToDel = common_driver.para.add()
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
    return common_driver
```

b. 配置设备协议类驱动数据

自定义逻辑，读取cli-driver.properties文件中的配置。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo
import os
import codecs

    def getCliDriverInfo(self, aoccontext, request=None):
        self.logger.info('getCliDriverInfo start.')
        cliDriverInfo = CliDriverInfo()
        self.logger.info(self.get_dict())
        for key, value in self.get_dict().items():
            cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
            cliDriverEntity.key = key
            cliDriverEntity.value = value
        self.logger.info('getCliDriverInfo end.')
        return cliDriverInfo


    def get_dict(self):
        try:
            path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
            self.logger.info(self.resourceDir)
            self.logger.info(path)
            pro_file = open(path, 'r')
            properties = {}
            for line in pro_file:
                if line.find('=') > 0:
                    strs = line.replace('\n', '').split('=')
                    properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
        except IOError as e:
            self.logger.info('getCliDriverInfo start.')
```


版权所有 © 华为技术有限公司

```
            self.logger.info(e)
            return {}
        else:
            pro_file.close()
        self.logger.info(properties)
        return properties
```

cli-driver.properties文件中通过配置项定制CLI设备驱动驱动信息。

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2019-2019. All rights reserved.
#

protocol=Ssh
userModeCommand=
defaultParser=basicPatternParser
keepAliveCommandParser=basicPatternParser
privilegedModeCommandAfterAbortCommand=false
userModePrompt=.*#.*
privilegedModePrompt=^.*\\(config\\)#$
promptSkipRegex=(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}
(@prompt@){1}$
privilegedModeCommand=
userNameResponse=.*login.*
passwordResponse=.*password.*
initCommand=show clock
keepAliveCommand=show clock utc
cliTxCreateCommand=
cliTxCommitCommand=
cliTxCancelCommand=
cliTxSaveCommand=
abortCommand=
cliTxCancelRetryTimes=3
connectionTimeout=10000
keepAliveInterval=120000
keepConnectedInterval=20000
readMessageTimeout=60000
cliType=
logData=false
errorMessagePattern=^([\\s\\S]*)(Error:|Info: The interval between portswitch|unrecognized
keyword ^end).*
ignoreMessagePattern=randomvalueshould#not#match
warningMessagePattern=^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*
maxNumberReadsChannels=1
maxResponseSize=400
showConfigCommand=end\nshow configuration running
showConfigSkipRegex=^(--|echo|#).*$
resetCommand=end
submodeCommand=configure ${context}
configModeCommand= end\nconfigure
operModeCommand=show
partialReadCommand=end\nshow run section ${context}
indentationSequence=\u0020\u0020
exitSubmodeCommand=exit
submodeStartSequence=@indent@
submodeEndSequence=@dedent@
negateCommand=no\u0020
shutdownCommand=\u0020shutdown
operationalDataReadCommand=/show ${context}
contextAccessType=readDatastore
contextSupportsSinglelineCommand=false
defaultDeleteOperationWithValue=false
supportsSinglelineCommand=false
cliInteractionMatchKey=Warning.*(base on this interface, continue| BGP process will be deleted.
Continue).*
cliInteractionGoOnCommond=Y
judgeContinueCondition=.*--More--.*
partialReadSubmode=false
continueReadingCommand=\u0020
changePasswordPrompt=.*Please choose one:.*
skipChangePasswordCmd=3
```


版权所有 © 华为技术有限公司

c. 设置业务自定义处理

自定义代码开发扩展项语法，实现CLI转YANG逻辑。

```
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
from aoc.sys import cliproxy

    # 自定义代码实现cli转yang逻辑
    def cliToYang(self, aoccontext, request):
        self.logger.info('cliToYang start.')
        self.logger.info('yangToCli current path : %s' % request.path)
        if request.path == '/hillstone-policy:rule':
            return CliToYangCustom.clitoyang_rule(request, self.logger)
        if request.path == '/hillstone-pool:address-pool':
            return CliToYangCustom.clitoyang_address(request, self.logger)
        self.logger.info('cliToYang end.')


    def clitoyang_rule(request, logger):
        out = CliCustomTransformOutput()
        proxy = CommandProxy(request.neId, logger)
        # proxy.send_command(['exit', 'configure'])
        response_data = proxy.send_command(['show config policy'], 120)
        response_body = response_data.response if hasattr(response_data, 'response') else
response_data
        out.data = ConfigUtil.handle_response_from_device(response_body)
        out.endOffset = -1
        logger.info('yangToCli policy out.data %s' % out.data)
        return out


    def clitoyang_address(request, logger):
        out = CliCustomTransformOutput()
        proxy = CommandProxy(request.neId, logger)
        # proxy.send_command(['exit', 'configure'])
        response_data = proxy.send_command(['show configuration address'], 120)
        response_body = response_data.response if hasattr(response_data, 'response') else
response_data
        out.data = ConfigUtil.handle_response_from_device(response_body)
        logger.info('clitoyang address pool out.data: %s' % out.data)
        out.endOffset = -1
        return out


# 自定义实现类
class CommandProxy(object):
    """
    invoke java, send command to device
    """

    def __init__(self, ne_id, logger):
        logger.info("build CommandProxy for " + ne_id)
        self.ne_id = ne_id
        self.logger = logger

    def send_command(self, commands, timeout=20):
        """
        向设备发送命令行
        :param commands: 命令行列表 string
        :param timeout: 超时时间
        :return: NONE
        """
        for command in commands:
            try:
                return cliproxy.sendCli(self.ne_id, command, timeout)
            except Exception as e:
                self.logger.error(e)
                return ""

    def send_command_list(self, commands, timeout=20):
        """
```


版权所有 © 华为技术有限公司

```
        向设备发送命令行合集并获取回显
        :param commands: string
        :param timeout: 超时时间
        :return: SendCliOutput
        """
        response = []
        for command in commands:
            try:
                response.append(cliproxy.sendCli(self.ne_id, command, timeout))
            except Exception as e:
                self.logger.error(e)
                return ""
        return response[-1]
```

d. 设置数据后置处理

当生成的CLI命令与设备上的实际命令不同，使用configCliPostProcessor自定义转换后的命令。

```
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput

    def configCliPostProcessor(self, aoccontext, request):
        self.logger.info('default configCliPostProcessor start')
        out = CliCustomTransformOutput()
        cli_str = request.data
        lines = cli_str.splitlines()
        self.logger.info('default configCliPostProcessor request %s' % request)
        newLines = []
        for line in lines:
            if constants.REQUEST_PATH_HILLSTONE_ADDRESS in request.path:
                if constants.ADDRESS in line:
                    index = 2 if (request.oper_type == OperType.Value(constants.OPERTYPE_CREATE)
or
                                  (request.oper_type == OperType.Value(constants.OPERTYPE_DELETE) and
                                   constants.REQUEST_PATH_HILLSTONE_POOL_ADDRESS != request.path)
                                  or (request.oper_type ==
OperType.Value(constants.OPERTYPE_MERGE))) else 4
                    line = ''.join(re.split(r'(\w+)', line)[:index]) + ' \"' + ''.join(
                        re.split(r'(\w+)', line)[(index + 1):]) + '\"'
                    newLines.append(line)
                elif constants.MEMBER in line or (constants.DESCRIPTION in line and
                                                  request.oper_type !=
OperType.Value(constants.OPERTYPE_DELETE)):
                    index = 2 if request.oper_type == OperType.Value(
                        constants.OPERTYPE_CREATE) or request.oper_type == OperType.Value(
                        constants.OPERTYPE_MERGE) else 4
                    line = ''.join(re.split(r'(\w+)', line)[:index]) + ' \"' + ''.join(
                        re.split(r'(\w+)', line)[(index + 1):]) + '\"'
                    newLines.append(line)
                else:
                    newLines.append(line)
            elif constants.REQUEST_PATH_HILLSTONE_POLICY in request.path and not bool(
                    re.search(r'rule id \d+', line)) and \
                    constants.ACTION not in line and constants.EXIT not in line:
                if request.oper_type == OperType.Value(constants.OPERTYPE_DELETE) and
ConfigUtil.check_key_exists(line):
                    newLines.append(line)
                else:
                    index = 2 if request.oper_type == OperType.Value(
                        constants.OPERTYPE_CREATE) or request.oper_type == OperType.Value(
                        constants.OPERTYPE_MERGE) else 4
                    line = ''.join(re.split(r'(\w+-?\w+)', line)[:index]) + ' \"' \
                           + ''.join(re.split(r'(\w+-?\w+)', line)[index:]).strip() + '\"'
                    newLines.append(line)
            else:
                newLines.append(line)
        out.data = '\n'.join(newLines)
        self.logger.info('default configCliPostProcessor end')
        return out
```

 版权所有 © 华为技术有限公司

3. 编写设备特性能力定制

自定义差异发现，对账和同步使用的特性范围。

```
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function


def getFeatures(self, aoccontext, request=None):
    self.logger.info('getFeatures start.')
    feature_msg = FeatureCfgsMsg()
    feature_msg.replace = True
    feature_msg.features.extend(
        self.build_feature('hillstone-policy', '',
            '(https://www.hillstonenet.com.cn/hillstone-policy?revision=2022-02-17)rule'))
    feature_msg.features.extend(
        self.build_feature('hillstone-pool', '',
            '(https://www.hillstonenet.com.cn/hillstone-pool?revision=2022-02-17)address-pool'))
    self.logger.info('getFeatures end.')
    return feature_msg


def build_feature(self, name, depends, path):
    feature = Feature()
    feature.name = name
    feature.operType = Feature.MERGE
    feature.depends.extend([depends])
    function = Function()
    function.value = path
    function.collectPath = path
    function.preSyncToNe = False
    function.preSyncFromNe = False
    function.postSyncFromNe = False
    feature.functions.extend([function])
    return [feature]
```

----结束

5.3 开发 NETCONF&CLI 双协议设备原子驱动包

本章节介绍NETCONF&CLI双协议设备原子驱动包的开发过程。

用户在开发SND软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

5.3.1 创建原子驱动包模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据表5-26设备软件包的相关属性。

表 5-26 NETCONF&CLI 软件包属性

属性名称	属性值
名称	NETCONF-CLI-SND
版本	1.0.0
提供商	HUAWEI

版权所有 © 华为技术有限公司

属性名称	属性值
包类型	Specific NE driver
映射类型	python
设备类型	NETCONF-CLI
协议类型	NETCONF

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

5.3.2 开发原子驱动包

本节详细介绍开发SND包全过程。

5.3.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 请按照参数表修改对应参数的值。

表 5-27 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
package-type	包类型。	枚举值：ssp（Specific Service plugin）、gnd（Generic NE driver）、snd（Specific NE driver）、mp（SND Migrate plugin）。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。仅支持该字段为“huawei”（不区分大小写）的SND和GND包。	必选
augment-isolation	模型范围标识。	布尔值，只能为 true 或 false。DCN单站场景需要配置为true，其他场景为false。	可选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。 version（必选）字段：版本号三段式规则。	可选
snd-type	snd的类型。	值为“system”，表示为系统内置的snd包。系统内置的snd包，该参数值不建议修改。 该参数未设置时，默认值为“specific”，表示用户自定义包。	可选
snd-id	snd包才需要设置的值。id信息区分不同snd包，不同名的包不能使用相同的snd-id。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。长度不能超过128个字符。 snd包时，必须设置，其他类型软件包时，不需要设置。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
devices	snd, gnd包特有值，驱动匹配的设备信息。	如果配置，则必须包含： ● 驱动设备厂商vendor（必选），不能超过 128个字符。 ● 具体设备类型device-type（必选），不能超过 128个字符。 ● 具体设备的版本device-version（必选），不能超过 128个字符。	可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。只能包含，数字、大小写字母，下划线（“_”），连字符(“-”)，不能超过128个字符。SND包设置为"snd"。 ● key（必选），回调key值，不能超过128个字符。SND包一般只需配置“ecs-driver”，指定在SND包中具体实现类；用户不使用系统默认连接agent时，再补充配置“ecs-connect-agent”，自定义hook中配置的连接agent，要生效需要同时设备的连接能力参数connectPolicy 配置为“COMPATIBLE_CONNECT”。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“device-version”、“python-class-name”，参数含义和修改方法请参见上述表格。最终编辑文件示例如下：

```
{
"name": "NETCONF-CLI-SND",
"version": "1.0.15",
"description": "",
"package-type": "snd",
"producer": "HUAWEI",
"nce-min-versions": [
        "1.0.0"
    ],
"snd-id": "NETCONF-CLI",
"devices": [
```

 版权所有 © 华为技术有限公司

```
    {
      "vendor": "HUAWEI",
      "device-type": "NetEngine",
      "device-version": "V200R005C10SPC800"
    }
  ],
"hooks": [
    {
      "type": "snd",
      "key": "ecs-driver",
      "python-class-name": "com.huawei.controller.devicetype.NETCONFCLI.NE"
    }
  ]
}
```

----结束

5.3.2.2 编写设备 YANG 模型

SND包中的YANG不参与编译，用于描述设备的配置能力、设备的北向接口以及生成南向报文。用户根据SND包类型，获取或编写对应的YANG模型，替换掉软件包中默认的YANG模型文件。

本示例中，请用户将SND模板中默认YANG文件删除，将从设备商获取设备的YANG模型文件放置到yang文件夹下。

如果用户需要使用自定义数据加密处理、高危操作定义、YANG模块黑名单屏蔽、YANG模型免校验、自定义模型权限等功能，请参考下文配置。

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

表 5-28 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d
-	-	type	节点类型，取值范围： • leaf • leaf-list • typedef –自定义类型 默认是leaf

版权所有 © 华为技术有限公司

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
    <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
    <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
    <path>/db-security-test:security-top/security-list/security-list-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
    <rpcs>
        <rpc>huawei-aaa:reactivateUser</rpc>
        <rpc>huawei-aaa:changeMyPassword</rpc>
        <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
    </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

 版权所有 © 华为技术有限公司

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```

5.3.2.3 开发代码

开发Python代码，实现设备的纳管、设备的配置管理、定制差异发现及同步和对账能力。

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码。用户可以重命名py文件名称和路径，将名称改为有意义的名称，需要与pkg.json文件的python-class-name中py文件名称字段保持一致。

5.3.2.3.1 引入必要的头文件

在代码最前面，首先需要引入必要的头文件中的相关接口。

版权所有 © 华为技术有限公司

下文在重写各个方法时描述了需要引用的头文件接口，请用户根据SND包类型、业务逻辑，参考示例代码导入需要的接口。

import语句应当放在文件头部，置于模块说明之后，全局变量之前。

5.3.2.3.2 编写设备纳管定制

编写设备纳管定制，包含两部分：注册设备sysoid信息和配置设备的连接能力信息。

注册设备 sysoid 信息

用户通过getSysoidInfo方法设置设备的sysoid信息，用于纳管此款型设备。

该方法不是必须要重写。纳管时期望对设备的sysoid信息校验，则需要重写方法，并准确配置sysoid字段信息。不对设备的sysoid信息校验，则可以不重写方法，通过pkg.json文件里的三元组来匹配。

注册设备sysoid信息的参数如下表所示。

表 5-29 注册设备 sysoid 参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
sysoid	否	STRING	N/A	N/A	设备唯一标识。 sysoid 可以通过以下方式获取： • 命令行查询，如华为设备一般可以执行display system information查询 • mib browser工具查询 • 通过OID网站查找，**https://oidref.com**
deviceType	否	STRING	N/A	N/A	设备类型。 请根据具体设备，匹配对应的设备类型，常用取值ROUTER、SWITCH、FIREWALL。
deviceModel	否	STRING	N/A	N/A	设备款型。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。
deviceVendor	否	STRING	N/A	N/A	设备厂商。 可通过命令行查询获取，如华为设备一般可以通过display version查询设备厂商、版本和款型。

注册设备信息代码示例如下。

```
# 保留默认代码中导入NetconfSND、SysoidInfo接口，重写getSysoidInfo方法时需要使用。
from aoc.snd.netconfclisnd import NetconfCliSND
```


版权所有 © 华为技术有限公司

```
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo


# 默认定义实现类代码可全部删除，按如下示例定义继承NetconfCliSND父类的子类
# NetconfCliSND是所有Netconf&CLI协议网元驱动包的父类，默认的配置已在父类中实现
class NE(NetconfCliSND):

    # 重写getSysoidInfo方法，注册设备sysoid值、设备类型信息、设备型号信息、设备厂商信息
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()

    # 设置设备的sysoid
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.360.1.10"

    # 设置设备的设备类型
    sysoidEntity.deviceType = "ROUTER"

    # 设置设备的设备款型
    sysoidEntity.deviceModel = "NetEngine"

    # 设置设备的设备厂商
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

配置设备的连接能力信息

NETCONF&CLI SND包由于支持双协议，需要同时配置主协议和辅助协议连接能力信息，其他类型的SND包只需要配置主协议连接能力信息。

设备连接能力信息常用参数如下表所示，详细的参数说明请参考《 插件包API参考 》。继承的父类不同，默认值则可能存在差异，以下表格中不做特殊说明，都是继承NetconfSND父类后的默认值。

表 5-30 设备连接能力常用参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
connectPolicy	否	ENUM	DEFAULT_CONNECT/COMPATIBLE_CONNECT	DEFAULT_CONNECT	连接策略 DEFAULT_CONNECT，由默认agent处理 COMPATIBLE_CONNECT，由compatible agent处理，由hook指定。如果hook中未指定，则需要在methodType、agentImpl、priority参数中配置具体实现类agent。
channelInfo.writeChannel	否	ENUM	SINGLE_CHANNEL/PROTECTED_MODE	PROTECTED_MODE	写通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
channelInfo.readChannel	否	ENUM	SINGLE_CHANNEL/PROTECTED_MODE	SINGLE_CHANNEL	读通道类型 SINGLE_CHANNEL：表示单链路无保护 PROTECTED_MODE：表示双链路主备保护
channelInfo.is_read_share_write	否	bool	True/False	False	是否读写通道共享 True：读写通道共享。只需要配置写通道类型即可，另一种通道无需配置。 False：读写通道不共享。读写通道类型都需要配置。
protocolType	否	ENUM	netconf/cli/restful/restconf	netconf	设置SND包的协议类型
helloType	否	ENUM	defaultType/standardTyp/extendType	defaultType	报文类型 defaultType，建立schema对接的NETCONF通道，能力集置为空。 standardType，建立标准yang能力集的NETCONF通道，能力集置为空。 extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。
extendEntity	否	ARRAY_STRING	N/A	N/A	能力集列表 若报文类型为extendType类型时，需要携带扩展能力集。
connectionPriority	否	ENUM	PRIMARY_CONNECTION/AUXILIARY_CONNECTION	PRIMARY_CONNECTION	连接调用优先级 PRIMARY_CONNECTION：单协议SND包、双协议SND包的主协议需配置为该值。 AUXILIARY_CONNECTION：双协议SND包时，辅助协议需配置为该值。
methodType	否	STRING	Java	N/A	指定实现类agent的编码语言。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
agentImpl	否	STRING	N/A	N/A	指定实现类agent。
priority	否	STRING	N/A	N/A	指定实现类agent连接调用优先级。

配置主协议和辅协议设备连接能力信息代码示例如下。

```
# 导入ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity接口，重写getConnectInfo方法配置设备连接能力时使用，用于配置连接信息，如与设备的建连协议，建立几个通道，握手的hello报文等
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, AUXILIARY_CONNECTION, HelloEntity
# 导入SINGLE_CHANNEL接口，重写getConnectInfo方法配置设备连接通道时使用
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL


    # 重写getConnectInfo方法，配置NETCONF协议类型及连接能力信息，若不重写则按默认值配置
    def getConnectInfo(self, aoccontext, request=None):
        self.logger.info('getConnectInfo start.')
        connectInfos = ConnectInfos()

        # 所有类型的SND包，都需要配置主协议能力信息
        primaryConnectInfo = connectInfos.connectInfo.add()

        # 设置连接策略
        # primaryConnectInfo.connectPolicy = DEFAULT_CONNECT
        primaryConnectInfo.connectPolicy = COMPATIBLE_CONNECT

        """
        设置通道配置。NETCONF协议需要配置不同的通道类型，
        PROTECTED_MODE表示主备保护，SINGLE_CHANNEL表示单链路无保护，
        当is_read_share_write为true表示读写通道共享，readChannel无需配置。
        """
        # primaryConnectInfo.channelInfo.writeChannel = PROTECTED_MODE
        primaryConnectInfo.channelInfo.readChannel = SINGLE_CHANNEL
        primaryConnectInfo.channelInfo.is_read_share_write = True

        # 设置主协议类型
        primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

        # 指定自定义连接agent
        primaryConnectInfo.connectEntity.methodType = 'java'
        primaryConnectInfo.connectEntity.agentImpl =
'com.huawei.enterprise.naas.netconfcliagent.service.NetconfCliAgentImpl'
        primaryConnectInfo.connectEntity.priority = 0

        """
        设置报文类型：defaultType，建立schema对接的NETCONF通道，能力集置为空；
        standardType，建立标准yang能力集的NETCONF通道，能力集置为空；
        extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。
        """
        primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.defaultType

        # 设置连接调用优先级
        primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION

        # NETCONF&CLI双协议SND，还需要配置辅助协议能力信息
        auxiliaryConnectInfo = connectInfos.connectInfo.add()
```


版权所有 © 华为技术有限公司

```
        # 设置连接策略
        auxiliaryConnectInfo.connectPolicy = COMPATIBLE_CONNECT

        """
        设置通道配置，PROTECTED_MODE表示主备保护，
        CLI协议时无需配置readChannel，is_read_share_write为False表示读写通道不共享
        """
        auxiliaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
        auxiliaryConnectInfo.channelInfo.is_read_share_write = True

        # 设置协议类型，NETCONF或CLI
        auxiliaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli

        """
        设置报文类型：defaultType，建立schema对接的NETCONF通道，能力集置为空；
        standardType，建立标准yang能力集的NETCONF通道，能力集置为空；
        extendType，建立扩展yang能力集的NETCONF通道，需要添加能力集列表。
        """

        # 指定自定义连接agent
        auxiliaryConnectInfo.connectEntity.methodType = 'java'
        auxiliaryConnectInfo.connectEntity.agentImpl =
'com.huawei.enterprise.naas.netconfcliagent.service.NetconfCliAgentImpl'
        auxiliaryConnectInfo.connectEntity.priority = 0

        # 设置连接调用优先级
        auxiliaryConnectInfo.connectionPriority = AUXILIARY_CONNECTION
        self.logger.info('getConnectInfo end.')
        return connectInfos
```

5.3.2.3.3 编写设备类型能力定制

编写设备类型能力主要包括：配置设备驱动数据、业务自定义处理、数据前置后置处理、数据联动处理。

5.3.2.3.3.1 配置设备驱动数据

提供了两种方法，编写Python配置设备驱动数据，或者通过编写xml文件配置设备驱动数据，下面分别进行说明。

通过代码配置设备驱动信息

- **基于通用能力定制示例，定义与设备的交互策略**

  编写基于通用能力的交互策略。常用参数请参见下表，详细的参数说明请参考《 插件包API参考 》。

  表 5-31 设备通用驱动参数

参数名称	是否必选	参数类型	参数值域	默认值	说明
checkSync	否	bool	true/false	false	在向设备下发业务数据过程中检查设备和控制器的数据是否一致。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
unsupportedOperations	否	STRING	"create"/"delete"/"create,delete"	N/A	设备不支持的操作类型，自动进行转换。如将 "create,delete"转换为 "merge,remove" 下发配置时可能会修改设备已有配置，请谨慎配置，以免引发业务安全风险。
deleteStrategy	否	DeleteStrategy	PATH_ADD/CONTAINER_WITH_PRESENCE	CONTAINER_WITH_PRESENCE	对于delete、remove操作的报文，是否删除报文中的具体信息 PATH_ADD时，指定PATH的报文内容删除。 CONTAINER_WITH_PRESENCE时，下发报文不做内容删除。
syncToDel	否	N/A	syncToDel.key = "sync-to-del-enable" syncToDel.value 可以为 "true"或“false”	false	设置数据对账时是否支持删除南向设备配置实例。 当此参数设置为true时，会在数据对账时删除仅设备侧存在的配置数据，可能引发业务安全风险。建议将此参数设置为false。
pathNeedPreProcess	否	STRING	N/A	N/A	该字段填写需要进行报文前置处理的特性QName。若该字段填写了对应QName，对于设备返回的查询结果报文，会在处理之前先调用netconfPreprocess进行加工，加工之后再进行报文转换。

版权所有 © 华为技术有限公司

参数名称	是否必选	参数类型	参数值域	默认值	说明
pathNeedPostProcess	否	STRING	N/A	N/A	该字段填写需要进行报文后置处理的特性QName。若该字段填写了对应QName，对于即将下发设备的报文，会在下发之前先调用netconfPostprocess进行加工，加工之后再下发设备。
pathNeedEcsMapping	否	STRING	N/A	N/A	该字段填写需要进行业务报文定制处理的特性QName。若该字段填写了对应QName，对于下发设备的报文和设备返回的都统一调用toDocument、toDataObject进行转换。
pathNeedEcsRpcMapping	否	STRING	N/A	N/A	该字段填写需要进行rpc报文定制处理的rpc QName。若该字段填写了对应QName，对于下发设备的rpc报文和设备返回的都统一调用toRpcDocument、toRpcDataObject进行转换。
pathNeedEcsDBHook	否	STRING	N/A	N/A	下发配置到设备后，某些配置会联动生成其他配置，而控制器只会存储联动前的配置，会导致控制器和转发器配置不一致。因此，控制器端也需要在配置下发前进行联动处理。

通用能力定制代码示例如下。

```
# 导入CommonDriverInfo接口，重写getCommonDriverInfo方法编写设备类型时使用，基于通用能力定制，用于定制通用操作类型转换等
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo


  # 重写getCommonDriverInfo方法，配置设备通用驱动，不重写则按默认值配置
  def getCommonDriverInfo(self, aoccontext, request=None):
  common_driver = CommonDriverInfo()

      # 在向设备下发业务数据过程中检查设备和控制器的数据是否一致。建议设置此参数为true，并实现getSyncId方法获取SyncId。
      common_driver.checkSync = True

  # 设备不支持的操作类型，自动进行转换，如将 "create,delete"转换为"merge,remove"。
  common_driver.unsupportedOperations = "create,delete"
      common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
  syncToDel = common_driver.para.add()

  # 定义key值，同步删除
  syncToDel.key = "sync-to-del-enable"

  # 设置数据对账时是否支持删除南向设备配置实例。取值：true（支持）。
  syncToDel.value = "true"
  return common_driver
```


版权所有 © 华为技术有限公司

- **基于协议类型定制示例，定义Netconf协议参数**

  支持NETCONF协议，需要定义NETCONF协议参数。常用参数请参见下表，详细的参数说明请参考《 插件包API参考 》。

  表 5-32 设备协议类参数

参数	是否必选	参数类型	参数值域	默认值	说明
classification	否	STRING	"huawei-v5"/"huawei-v8"/"cisco"	huawei-v5	设备原型。 huawei-v5:报文转换规则为match huawei-v8:报文转换规则为same cisco:报文转换规则为match
phase	否	STRING	"one"/"two"	one	设备支持的几阶段下发。
maxPkgLength	否	INT32	>0	N/A	设备支持的最大单个报文的大小。
netconfLock	否	bool	true/false	false	下发设备时是否需要先获取设备锁。若设备为juniper的设备，该字段须为true。
errorOption	否	STRING	"rollback-on-error"/"stop-on-error"/"continue-on-error"	"rollback-on-error"	设备配置数据出错之后的处理策略。
testOption	否	STRING	"set"	N/A	下发设备的报文是否需要加上<test-option>标签，当该字段为“set”时，下发报文会加上<test-option>set</test-option>。

版权所有 © 华为技术有限公司

参数	是否必选	参数类型	参数值域	默认值	说明
netconfHelloEntity	否	NetconfHelloEntity	standardType/ extendType/ defaultType	defaultType	设备管理hello报文定制： standardType：标准报文 extendType：自定义报文 defaultType：默认报文（YANG对接）
modelDiff	否	STRING	"same"/"match"	match	netconf报文转换类型。当为same时，下发的报文的namespace会统一转换为"https://www.huawei.com/netconf/vrp"。

协议类型定制代码示例如下。

```
# 导入NetconfDriverInfo接口，重写getNetconfDriverInfo方法编写设备类型时使用，基于Netconf协议类型能力定制，用于配置netconf协议参数
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo


    # 重写getNetconfDriverInfo方法，配置设备NETCONF驱动，不重写则按默认值配置
    def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()

    # 设置neconf下发阶段，一阶段还是二阶段。如果是支持二阶段的设备，设置此参数为two。
    netconf_driver.phase = "two"

    # 定义驱动设备原型类别，大部分华为设备是huawei-v5
    netconf_driver.classification = "huawei-v5"

    """
    定义报文转换类型。当为same时，下发报文为Schema标准报文；
    当为match时，下发报文为YANG标准报文
        netconf_driver.modelDiff = "match"
    """

    # 定义报文校验策略，当该字段为set时，直接根据报文处理不用校验。
    netconf_driver.testOption = "set"
    return netconf_driver
```

- **基于协议类型定制示例，定义CLI协议参数**

  CLI SND包，支持CLI协议，必须要定义CLI协议参数。

  协议类型定制代码示例如下。

```
# 导入CLIDriverInfo接口，重写getCliDriverInfo方法时使用
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo
import os
import codecs


    # 重写getCliDriverInfo方法，配置设备Cli驱动，不重写则按默认值配置
    # 这里是通过自定义读取cli-driver.properties文件进行配置的，Cli驱动参数较多，通过文件配置较方便
```


版权所有 © 华为技术有限公司

```
def getCliDriverInfo(self, aoccontext, request=None):
    self.logger.info('getCliDriverInfo start.')
    cliDriverInfo = CliDriverInfo()
    self.logger.info(self.get_dict())

# 将字典中的键值对赋值给配置项
    for key, value in self.get_dict().items():
        cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
        cliDriverEntity.key = key
        cliDriverEntity.value = value
    self.logger.info('getCliDriverInfo end.')
    return cliDriverInfo


# 将resources/cli-driver.properties设备驱动文件中的信息读入到字典中
def get_dict(self):
    try:
        path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
        self.logger.info(self.resourceDir)
        self.logger.info(path)
        pro_file = open(path, 'r')
        properties = {}
        for line in pro_file:
            if line.find('=') > 0:
                strs = line.replace('\n', '').split('=')
                properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
    except IOError as e:
        self.logger.info('getCliDriverInfo start.')
        self.logger.info(e)
        return {}
    else:
        pro_file.close()
    self.logger.info(properties)
    return properties
```

通过文件配置设备驱动信息

通过文件配置设备驱动信息，系统自动解析文件或用户自定义解析文件逻辑。用户按照要求命名和编写xml配置文件，不需要编码系统可自动解析xml文件配置设备驱动信息。

用户通过“resources/cli-driver.properties”文件，可以定制CLI设备驱动信息，用于解析和识别设备的CLI回显。配置项详细信息如下表所示。

表 5-33 配置项

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
userModePrompt	^<HUAWEI.*>$	-	是	用户模式下设备命令行提示符正则表达式，用于解析返回报文是否为用户模式提示符。	<HUAWEI123>
privilegedModePrompt	^\\[[~*]HUAWEI.*\\]$	-	是	特权模式下设备命令行提示符正则表达式，用于解析返回报文是否为特权模式的提示符。	[~HUAWEI123]

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
promptSkipRegex	(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}(@prompt@){1}$	-	否	跳过解析的提示符正则表达式，用于跳过解析返回报文中的提示符场景。 正则表达式中的@prompt@将被替换为userModePrompt或privilegedModePrompt配置值。	-
userNameResponse	Login:	-	否	登录时提示输入用户名的命令行提示符匹配规则，用于解析返回报文是否为输入用户名提示符。	Login:
passwordResponse	Password:	-	否	登录时提示输入密码的命令行提示符匹配规则，用于解析返回报文是否为输入密码提示符。	Password:
InitCommand	display users	-	否	初始连接设备后发送的初始化命令，用于系统识别到成功登录设备后自动执行和显示的初始化命令。	<HUAWEI123>display users User-Intf Delay Type Network Address AuthenStatus AuthorcmdFlag + 34 VTY 0 00:00:00 TEL 10.183.45.110 pass yes Username : huawei <HUAWEI123>

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
keepAliveCommand	display users	-	否	保活连接命令，用于定期向设备发送该命令以保持连接。	<HUAWEI123>display users User-Intf Delay Type Network Address AuthenStatus AuthorcmdFlag + 34 VTY 0 00:00:00 TEL 10.183.45.110 pass yes Username : huawei <HUAWEI123>
cliTxCreateCommand	return\n	-	是	事务创建命令，用于发送配置前先下发创建事务。	<HUAWEI123>return <HUAWEI123>
cliTxCommitCommand	commit\n	-	是	事务提交命令，用于配置提交后再下发提交事务。	[~HUAWEI123]screen-length 24 [*HUAWEI123]commit [~HUAWEI123]
cliTxSaveCommand	return\n	-	是	事务保存命令，用于配置提交成功后再下发事务保存。	-
cliTxCancelCommand	clear configuration candidate \n	-	是	事务取消命令，用于取消配置等情况时下发事务取消。	[~HUAWEI123]screen-length 0 [*HUAWEI123]clear configuration candidate [~HUAWEI123]
cliTxCancelInterval	2000	2000	否	设备事务异常时，取消事务间隔时间，单位毫秒。用于多次取消事务操作时间间隔。	-
cliTxCancelRetryTimes	3	3	否	取消事务重试次数，用于设备事务异常时，取消事务可重试的次数。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
abortCommand	quit	-	否	异常停止命令，用于CLI协议出现异常时，通过该命令强制终止当前命令的执行，最好配置成Ctrl+C的unicode码。 未配置值时，CLI协议异常时处理策略是先断连，再建连。	[~HUAWEI123]quit <HUAWEI123>
connectionTimeout	10000	10000	否	连接超时时间，单位ms，用于判断连接超时。	-
keepAliveInterval	120000	20000	否	发送保活命令进行保活的间隔时间，单位为ms，用于保持设备连接。	-
errorMessagePattern	^Error:?[ ]{1}.*	-	否	配置Error正则表达式，用于解析返回报文中是否为配置失败的信息。	Error: Unrecognized command found at '^' position.
ignoreMessagePattern	randomvalueshould#not#match	-	否	忽略配置Error匹配正则表达式，用于当解析返回报文中为配置失败情况时但该场景又需要被忽略时。	-
warningMessagePattern	^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*	-	否	回显信息告警正则表达式，用于解析返回报文是否为告警信息。	Info: Removing directory cfcard:/1.txt/.....Done!
maxResponseSize	400	400	否	执行一条命令后，能够允许的最大返回报文大小，单位为KB。用于限制超大报文等异常场景。	-
showConfigCommand	return \ndisplay current-configuration	-	否	配置全量读的设备命令，用于查询设备配置。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
showConfigSkipRegex	^(--|echo|#).*$	-	否	不需要解析的关键字正则表达式，用于忽略解析含有该关键字正则表达式的返回报文。	-
configModeCommand	system-view	-	否	配置模式进入命令，用于下发实际配置命令（创建、删除、更新等）前下发该命令进入配置模式。	<HUAWEI123>system-view
partialReadCommand	return \nsystem-view\n${context}\ndisplay this	-	否	配置部分读的设备命令，用于查询设备部分配置。提前构造部分读命令，然后替换${context}生成具体的部分读命令。	<HUAWEI123>return <HUAWEI123>system-view [~HUAWEI123]interface Ethernet0/1/2 [~HUAWEI123-Ethernet0/1/2]display this
indentationSequence	\u0020\u0020\u0020\u0020	\u0020\u0020\u0020\u0020	否	命令行缩进序列，用于解析设备命令行的缩进序列。HUAWEI设备通常左缩进，四个空格键的缩进序列。	-
exitSubmodeCommand	quit	-	否	返回上级视图的命令，用于返回上级视图下发该命令。通常为quit、exit。	[~HUAWEI123]quit <HUAWEI123>
submodeStartSequence	@indent@	-	否	子模式开始序列，用于解析设备命令行子模式开始的缩进序列。@indent@表示缩进一级。	acl number 10001 rule 2 name xxx deny
submodeEndSequence	@dedent@	-	否	子模式结束序列，用于解析设备命令行子模式结束的缩进序列。@dedent@表示反缩进一级。	acl number 10001 rule 2 name xxx deny #

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
negateCommand	undo \u0020	-	否	逆命令表达式，用于解析设备命令行的no、undo等关键字，识别是否为逆命令。	[~HUAWEI123]undo
shutdownCommand	\u0020shutdown	\u0020shutdown	否	shutdown命令表达式，用于解析当下发delete命令时设备返回报文中含有shutdown等关键字，需要用户输入后才能继续的情况。	[~HUAWEI123-Ethernet0/1/5]dis this # interface Ethernet0/1/5 shutdown
contextSupportsSinglelineCommand	false	-	否	表示删除命令是否长命令模式，用于配置是否支持解析长命令模式。HUAWEI设备不支持，所以配置成false。 配置值包括true和false。	-
defaultDeleteOperationWithValue	false	-	否	表示删除叶子节点时，生成的命令是否需要带上叶子的值。用于其他厂家设备的特殊情况，HUAWEI设备不需要，所以配成false。 配置值包括true和false。	-
supportsSinglelineCommand	false	-	否	表示创建、修改命令是否支持长命令模式。用于其他厂家设备的特殊情况，HUAWEI设备不支持，所以配置成false。 配置值包括true和false。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
cliInteractionMatchKey	Warning.*(base on this interface, continue|BGP process will be deleted. Continue).* y(es)/n(o)	-	否	交互匹配信息正则表达式，用于下发设备命令时，解析设备返回提示信息，需要用户通过命令做出交互选择。支持分号分隔多个正则表达式。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:
cliInteractionGoOnCommond	y	-	否	交互匹配信息响应，用于解析出设备交互匹配信息时，对设备进行响应，与cliInteractionMatchKey配置项组合使用。支持分号分隔多个正则表达式，分号数量需要跟cliInteractionMatchKey对应。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:y Info: Removing directory cfcard:/1.txt/.....Done! <HUAWEI123>
judgeContinueCondition	.*--- More ---*	-	否	判断继续条件的正则表达式，用于解析返回报文是否包含继续条件正则表达式。常用于查询的配置没有显示完整的情况。	[~HUAWEI123]dis current-configuration !Software Version V800R020C00SPC200B336 ---- More ----
partialReadSubmode	false	-	否	部分读取子模式，用于配置是否支持通过进入子视图，然后输入一条命令，查询部分配置。例如，HUAWEI支持进入相应视图，display this查询当前配置，就配置为true。 配置值包括true和false。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
continueReadingCommand	\u0020	-	否	继续读取命令，用于查询设备配置时，设备没有全部返回查询的配置，通过该命令，让设备继续返回配置。	[~HUAWEI123]dis current-configuration !Software Version V800R020C00SPC200B336 !Last configuration was updated at 2021-09-02 07:30:54+00:00 by huawei [~HUAWEI123]
changePasswordPrompt	^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*	-	否	修改密码提示正则表达式，用于解析与设备建连时设备要求修改密码的提示信息。	Password: Warning: The initial password poses security risks. The password needs to be changed. Change now? [Y/N]:
skipChangePasswordCmd	N	-	否	连接提示更改密码时下发的命令，用于解析出changePasswordPrompt提示信息时，下发设备的命令。	The password needs to be changed. Change now? [Y/N]:n <HUAWEI123>
cliNoPrintCommandPattern	cliNoPrintCommandPattern=change password.*;set key.*	-	否	非打印命令模式，用于打印日志时过滤敏感命令，支持以分号分隔，默认空不过滤敏感命令。	-
channelOpeningTimeout	10000	10000	否	通道建连超时时间，用于防止通道建连异常长时间空耗资源。单位ms。	-

版权所有 © 华为技术有限公司

配置项名称	配置值示例	默认值	是否必选	含义与作用	示例说明
readMessageTimeout	10000	10000	否	通道读取返回报文超时时间，用于防止通道异常长时间读取报文。单位ms。	-
cliInteractionCommand	-	-	否	批量下发配置时可能出现交互的命令，用于提前识别批量配置需要交互的命令。以分号分隔，可配置多条命令。	<HUAWEI123>rm 1.txt Info: Remove directory cfcard:/1.txt/? [Y/N]:
ifBatchSendCommand	off	off	否	是否启用拼接命令方法的批量下发配置，用于其他厂家设备支持启用拼接命令方法批量下发配置时。 配置值包括on和off。	-
cliCmdAppender	0	0	否	命令回车符，用于解析命令行中的回车符。通常回车符为\r 或\n 。 配置为0表示\n ； 配置为1表示\r。	-
cliIndentSpace	\u0020\u0020\u0020\u0020	\u0020\u0020\u0020\u0020\u0020\u0020\u0020\u0020	否	子命令行缩进，用于下发进入视图命令时，子命令行缩进量。	-
rangeCommand	batch	-	否	批量命令行的范围命令，用于render-batch-list-syntax扩展字使用的配置项。	-
rangeIndicator	\u0020to\u0020	-	否	批量命令行的范围标识，用于render-batch-list-syntax扩展字使用的配置项。	-
rangeSeparator	\u0020	-	否	批量命令行的分隔符，用于render-batch-list-syntax扩展字使用的配置项。	-

版权所有 © 华为技术有限公司

5.3.2.3.3.2 设置业务自定义处理（可选）

当开放可编程提供的设备报文转换通用能力不能满足交互时，用户可以使用该方法自定义设备报文转换处理。

业务自定义处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfTransformer方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


  # 重写netconfTransformer方法
  def netconfTransformer(self, aoccontext, request=None):
      netconf_msg = NetconfMsg()
      if input.data.find('xmlns="urn:huawei:yang:huawei-ac-ne-snmp"') > -1:
          netconf_msg.msg =
              /input.data.replace('xmlns="urn:huawei:yang:huawei-ac-ne-snmp"',
              /'xmlns="http://www.huawei.com/netconf/vrp" content-version="1.0"  '
              /'format-version="1.0"')
      else:
          netconf_msg.msg = input.data.replace('http://www.huawei.com/netconf/vrp',
                        /'urn:huawei:yang:huawei-ac-ne-snmp')
```

5.3.2.3.3.3 设置数据前置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异，如网络和设备枚举值定义可能不一致。设置前置处理后，当数据从设备同步NCE的时候，可以对数据进行处理，从而使NCE能够识别数据。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPreProcess字段。

数据前置处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfPreprocess方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


  # 重写netconfPreprocess方法
  def netconfPreprocess(self, aoccontext, request=None):
  self.logger.info('netconfPreprocess start.')
  doc = request.data

  # 对数据进行前置处理调整
  new_doc = doc.replace("<viewName>", "<viewName>pre")
  netconfAtomicConfig = NetconfAtomicConfig()
  netconfAtomicConfig.data = new_doc
  self.logger.info('netconfPreprocess end.')
  return netconfAtomicConfig
```

5.3.2.3.3.4 设置数据后置处理（可选）

由于各厂家实现的不同，数据格式上可能存在差异。设置后置处理后，当数据从NCE同步到设备的时候，可以对数据进行处理，从而使设备能够识别数据。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedPostProcess字段。

数据后置处理代码示例如下。

```
# 导入NetconfAtomicConfig接口，重写netconfPostprocess方法时需要使用
from aoc.snd.snd_model_pb2.netconfProcess_pb2 import  NetconfAtomicConfig


  # 重写netconfPostprocess方法
  def netconfPostprocess(self, aoccontext, request=None):
```


版权所有 © 华为技术有限公司

```
    self.logger.info('netconfPostprocess start.')
    doc = request.data

    # 对数据进行后置处理调整
    new_doc = doc.replace("<viewName>", "<viewName>post")
    netconfAtomicConfig = NetconfAtomicConfig()
    netconfAtomicConfig.data = new_doc
    self.logger.info('netconfPostprocess end.')
    return netconfAtomicConfig
```

5.3.2.3.3.5 设置数据联动（可选）

给设备下发数据的时候，设备可能会进行数据的联动处理，此时需要NCE也做数据联动处理，保持NCE与设备的同步。

使用该方法时，需要先在设备驱动CommonDriverInfo中设置pathNeedEcsDBHook字段。

数据联动代码示例如下。

```
# 导入EcsDbConfigOut, EcsDbConfig接口，重写dbPostProcess方法配置数据联动时使用
from aoc.snd.snd_model_pb2.ecsDbProcess_pb2 import EcsDbConfigOut, EcsDbConfig


    # 重写dbPostProcess方法
    def dbPostProcess(self, aoccontext, request):
        ecsData = request.data
        ecsPath = request.path
        ecsopType = request.opType
        ecsDbConfigOut = EcsDbConfigOut()
        if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/huawei-ifm:service-name"
        elif "huawei-ifm:l2vc/secondary" in ecsPath and ecsopType == "DELETE":
            ifmKey = self.get_key(ecsPath)
            ecsDbconfig1 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig2 = ecsDbConfigOut.ecsDbConfig.add()
            ecsDbconfig1.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/reroute"
            ecsDbconfig2.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + "/huawei-ifm:mpls/
huawei-ifm:l2vpn/redundancy"
        return ecsDbConfigOut
```

5.3.2.3.3.6 获取 syncId（可选）

syncId是数据更改的记录ID。通过比较设备上syncId和NCE上的syncId，可以了解设备上数据是否与NCE一致。例如，用户通过命令行对设备下发过业务，就可能会造成数据不一致。当数据不一致时，可以通过同步或对账对齐数据。

在向设备下发业务数据过程中检查设备和控制器的数据是否一致时，则需要在设备驱动配置common_driver.checkSync = True，并重写getSyncId方法获取syncId。

获取syncId代码示例如下。

```
# netconf协议报文方式查询获取syncId示例
# 导入devicemgr接口，重写getSyncId方法使用
from aoc import devicemgr


    # netconf协议报文方式查询获取syncId示例
    def getSyncId(self, aoccontext, request):
        """
        获取设备configFlowId
        :param aoccontext: 上下文环境
```


版权所有 © 华为技术有限公司

```
    :param request: 方法携带的参数
    :return: sync_id_info_output
    """
    self.logger.info('getSyncId start. aoccontext is %s,'
                     ' request is %s' % (aoccontext, request))
    device_id = request.deviceId
    # 设备yang path
    path = '/huawei-cfg:cfg'
    # 构造查询设备netconf subtree报文
    properties = {
        'subtree': self.CFG_SUBTREE.strip()
    }
    # 查询设备get报文
    data = devicemgr.query_data_from_device(device_id, 'OPERATIONAL', path, properties, 60000)
    self.logger.info('query data rsp is %s' % data)
    data_dict = {}
    if data.isSuccessful == "success" and data.result is not None:
        data_dict = NcsService.xmltodictnode(data.result)
        self.logger.info('data_dict is %s' % data_dict)
    sync_id_info_output = SyncIdInfoOutput()
    if 'cfg' not in data_dict or 'checkpoints' not in data_dict['cfg'] \
            or 'checkpoint' not in data_dict['cfg']['checkpoints']:
        return sync_id_info_output
    commit_ids = []
    for checkpoint in data_dict['cfg']['checkpoints']['checkpoint']:
        if checkpoint['commit-id']:
            commit_ids.append(checkpoint['commit-id'])
    # 取最新commit_id作为sync_id
    sync_id_info_output.SyncId = max(commit_ids)
    self.logger.info('getSyncId end. SyncId is: %s' % sync_id_info_output.SyncId)
    return sync_id_info_output
```

5.3.2.3.4 编写设备特性能力定制（可选）

当用户由于业务需要，可能会对差异发现范围、对账数据、同步数据进行定制处理，则需要编写设备特性能力定制代码。如果没有定制，则会对所有的特性进行全量差异发现。

特性能力定制代码示例如下。

```
# 导入FeatureCfgsMsg, Feature, Function接口，重写编写设备特性能力方法时使用
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function


    # 重写getFeatures方法，新增特性定制
    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True

         # 新增一个特性定制，编写差异发现的路径
        feature_msg.features.extend(self.build_feature('huawei-l3vpn', 'huawei-rtp', '(http://www.huawei.com/
netconf/vrp/huawei-l3vpn?revision=2018-06-11)l3vpn'))
        self.logger.info('getFeatures end.')
        return feature_msg


    def build_feature(self, name, depends, path):
        feature = Feature()

        # 特性名称
        feature.name = name

        # 操作类型
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()
```


版权所有 © 华为技术有限公司

```
    # 设置差异比较路径
    function.value = path

    # 设置从南向设备采集数据的路径
    function.collectPath = path
    feature_name = name.replace('huawei-', '')

    # 设置对账前定制，设置为“True”，则调用重写的preSyncToNe过程，设置为“False”，则不定制
    function.preSyncToNe = True

    # 设置同步前定制，设置为“True”，则调用重写的preSyncFromNe过程，设置为“False”，则不定制
    function.preSyncFromNe = False

    # 设置同步后定制，设置为“True”，则调用重写的postSyncFromNe过程，设置为“False”，则不定制
    function.postSyncFromNe = False
    feature.functions.extend([function])
    return [feature]


# 重写对账前定制过程
def preSyncToNe(self, request, aoccontext=None):
    self.logger.info('preSyncToNe start.')
    dataIn = request

    # 获得业务注册的数据路径
    regPath = dataIn.regPath

    # 获得特性名称
    featureName = dataIn.feature

    # 获得事务id
    transId = dataIn.transId
    self.logger.info("preSyncToNe begin regPath ={}, feature={}, transId={}", regPath, featureName, transId)

    # 获得业务处理后返回的差异数据
    diffDatas = dataIn.diffDatas

    # 获得差异数据列表的容量
    diffSize = len(diffDatas)

    # 如果差异为null或者差异数据容量为0，表明没有差异，函数直接返回null
    if diffDatas is None or diffSize == 0:
        return None

    # 定制数据一致性对账使用的数据
    dataOut = DiffDataOutMsg()

    # 遍历差异数据列表，处理差异数据
    for data in diffDatas:
        # 获得xml格式的差异数据
        diffXml = data

        # TODO 处理diffXml
        diffDataOut = diffData()

        # 该差异数据是否需要单独提交到南向设备。false表示不需要单独提交到南向设备
        diffDataOut.singleMsg = False
        diffDataOut.diffData = diffXml
        dataOut.diffDatas.extend([diffDataOut])
    self.logger.info('preSyncToNe end.')
    return dataOut
```

说明

新老设备被同一款SND包管理，不同设备支持的特性范围不一致时，则会导致同步出现异常。例如：SND包支持管理A、B两个特性，老设备仅支持A特性，新设备支持A和B特性，系统发起同步时，会尝试同时同步两个特性，系统会报老设备同步异常。

版权所有 © 华为技术有限公司

5.3.3 验证原子驱动包

5.3.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

5.3.4 编译原子驱动包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件复制到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

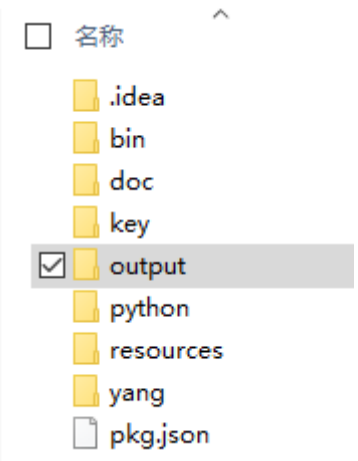
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。



版权所有 © 华为技术有限公司

```
2019-11-07 14:18:00,812 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Execute success
2019-11-07 14:18:00,819 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Zip: Clean dir success
2019-11-07 14:18:01,127 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Key length:3072
2019-11-07 14:18:01,180 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.common.FileUtil] -
[Sign]Generate signature file success.
2019-11-07 14:18:01,181 INFO
[com.huawei.ncecommon.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

5.3.5 开发 NETCONF&CLI SND 示例

5.3.5.1 开发 CE5855 NETCONF&CLI SND 示例

本示例开发华为CE5855 NETCONF&CLI SND。SND需要编写设备纳管定制代码、设备驱动数据代码、设备特性能力定制。实现NETCONF协议设备配置下发，CLI协议查询设备信息 。

步骤1 创建原子驱动包模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表5-34设置软件包的相关属性。

表 5-34 CE5855 SND 属性

属性名称	属性值
名称	CE5855-48T4S2Q-EI-V200R002C50SPC800
版本	1.0.0
提供商	HUAWEI
包类型	Specific NE driver
映射类型	python
设备类型	CE5855-48T4S2Q-EI
协议类型	NETCONF

版权所有 © 华为技术有限公司

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
"name": "CE5855-48T4S2Q-EI",
"version": "1.0.15",
"description": "",
"package-type": "snd",
"producer": "HUAWEI",
"nce-min-versions": [
        "1.0.0"
    ],
"snd-id": "CE5855-48T4S2Q-EI",
"devices": [
     {
        "vendor": "HUAWEI",
        "device-type": "CE5855-48T4S2Q-EI",
        "device-version": "V200R005C10SPC800"
     }
    ],
"hooks": [
     {
        "type": "snd",
        "key": "ecs-driver",
        "python-class-name": "com.huawei.controller.devicetype.CE5855_48T4S2Q_EI.CE585548T4S2QEI"
     }
    ]
}
```

步骤3 编写设备YANG模型

从厂商获取CE5855设备的YANG模型，替换掉软件包中默认的YANG模型文件。

步骤4 开发代码

1. 编写设备纳管定制
   a. 注册设备sysoid信息

```
from aoc.snd.netconfclisnd import NetconfCliSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

class CE585548T4S2QEI(NetconfCliSND):
    def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.239.22"
    sysoidEntity.deviceType = "ROUTER"
    sysoidEntity.deviceModel = "CE5855-48T4S2Q-EI"
    sysoidEntity.deviceVendor = "HUAWEI"
    return sysoidInfo
```

   b. 配置设备的连接能力信息

   连接策略配置为compatible agent处理，在agentImpl参数中配置实现类agent。

```
from aoc.snd.snd_model_pb2.connectinfo_pb2 import ConnectInfos, ProtocolEntity,
DEFAULT_CONNECT, PRIMARY_CONNECTION, AUXILIARY_CONNECTION, HelloEntity
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

    def getConnectInfo(self, aoccontext, request=None):
        self.logger.info('getConnectInfo start.')
        connectInfos = ConnectInfos()
        primaryConnectInfo = connectInfos.connectInfo.add()
        primaryConnectInfo.connectPolicy = COMPATIBLE_CONNECT
        primaryConnectInfo.channelInfo.readChannel = SINGLE_CHANNEL
        primaryConnectInfo.channelInfo.is_read_share_write = True
```

 版权所有 © 华为技术有限公司

```
        primaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.netconf
        primaryConnectInfo.connectEntity.methodType = 'java'
        primaryConnectInfo.connectEntity.agentImpl =
'com.huawei.enterprise.naas.netconfcliagent.service.NetconfCliAgentImpl'
        primaryConnectInfo.connectEntity.priority = 0
        primaryConnectInfo.protocolEntity.helloEntity.helloType = HelloEntity.defaultType
        primaryConnectInfo.connectionPriority = PRIMARY_CONNECTION

        auxiliaryConnectInfo = connectInfos.connectInfo.add()
        auxiliaryConnectInfo.connectPolicy = COMPATIBLE_CONNECT
        auxiliaryConnectInfo.channelInfo.writeChannel = SINGLE_CHANNEL
        auxiliaryConnectInfo.channelInfo.is_read_share_write = True
        auxiliaryConnectInfo.protocolEntity.protocolType = ProtocolEntity.cli
        auxiliaryConnectInfo.connectEntity.methodType = 'java'
        auxiliaryConnectInfo.connectEntity.agentImpl =
'com.huawei.enterprise.naas.netconfcliagent.service.NetconfCliAgentImpl'
        auxiliaryConnectInfo.connectEntity.priority = 0
        auxiliaryConnectInfo.connectionPriority = AUXILIARY_CONNECTION
        self.logger.info('getConnectInfo end.')
        return connectInfos
```

2. 编写设备类型能力定制

   a. 配置设备通用驱动数据

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

  def getCommonDriverInfo(self, aoccontext, request=None):
  common_driver = CommonDriverInfo()
  common_driver.unsupportedOperations = "create,delete"
        common_driver.deleteStrategy = CONTAINER_WITH_PRESENCE
  syncToDel = common_driver.para.add()
  syncToDel.key = "sync-to-del-enable"
  syncToDel.value = "true"
  return common_driver
```

   b. 配置设备协议类驱动数据

      i. 基于协议类型定制示例，定义Netconf协议参数。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo

  def getNetconfDriverInfo(self, aoccontext, request=None):
  netconf_driver = NetconfDriverInfo()
  netconf_driver.phase = "one"
  netconf_driver.classification = "huawei-v8"
  netconf_driver.testOption = "set"
  return netconf_driver
```

      ii. 基于协议类型定制示例，定义CLI协议参数。

      自定义逻辑，读取cli-driver.properties文件中的配置。

```
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
import os
import codecs

  def getNetconfDriverInfo(self, aoccontext, request=None):
  netconf_driver = NetconfDriverInfo()
  netconf_driver.phase = "one"
  netconf_driver.classification = "huawei-v8"
  netconf_driver.testOption = "set"
  return netconf_driver

from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CLIDriverInfo

  def getCliDriverInfo(self, aoccontext, request=None):
      self.logger.info('getCliDriverInfo start.')
      cliDriverInfo = CliDriverInfo()
      self.logger.info(self.get_dict())
      for key, value in self.get_dict().items():
          cliDriverEntity = cliDriverInfo.cliDriverEntity.add()
          cliDriverEntity.key = key
          cliDriverEntity.value = value
```


版权所有 © 华为技术有限公司

```
        self.logger.info('getCliDriverInfo end.')
        return cliDriverInfo


    def get_dict(self):
        try:
            path = os.path.join(self.resourceDir, "resources/cli-driver.properties")
            self.logger.info(self.resourceDir)
            self.logger.info(path)
            pro_file = open(path, 'r')
            properties = {}
            for line in pro_file:
                if line.find('=') > 0:
                    strs = line.replace('\n', '').split('=')
                    properties[strs[0]] = codecs.getdecoder("unicode_escape")(strs[1])[0]
        except IOError as e:
            self.logger.info('getCliDriverInfo start.')
            self.logger.info(e)
            return {}
        else:
            pro_file.close()
        self.logger.info(properties)
        return properties
```

cli-driver.properties文件中通过配置项定制CLI设备驱动信息。

```
#
# Copyright (c) Huawei Technologies Co., Ltd. 2019. All rights reserved.
#

protocol=Ssh
userModeCommand=
defaultParser=basicPatternParser
keepAliveCommandParser=basicPatternParser
privilegedModeCommandAfterAbortCommand=false
userModePrompt=^<.*>$
privilegedModePrompt=^\\[.*\\]$|.*ontinue\\? \\[Y\\/N\\]
promptSkipRegex=(?s).*((Error)|(WARNING)|(MINOR)|(INFO)):[ ]{1}[^@prompt@.]{1,}
(@prompt@){1}$
privilegedModeCommand=
userNameResponse=Username:
passwordResponse=Password:
initCommand=display clock
keepAliveCommand=dis clock utc
cliTxCreateCommand=
cliTxCommitCommand=
cliTxCancelCommand=
cliTxSaveCommand=
abortCommand=
cliTxCancelRetryTimes=3
connectionTimeout=10000
keepAliveInterval=120000
keepConnectedInterval=20000
cliType=
logData=false
errorMessagePattern=1111111111111
ignoreMessagePattern=randomvalueshould#not#match
warningMessagePattern=^((WARNING)|(MINOR)|(INFO)):[ ]{1}.*
maxNumberReadsChannels=1
maxResponseSize=400
showConfigCommand=return\ndisplay current-configuration
showConfigSkipRegex=^(--|echo|#).*$
resetCommand=return
submodeCommand=system-view ${context}
configModeCommand= return\nsystem-view
operModeCommand=display
partialReadCommand=return\nsystem-view\n${context}\ndisplay this
indentationSequence=\u0020\u0020\u0020\u0020
exitSubmodeCommand=quit
submodeStartSequence=@indent@
submodeEndSequence=@dedent@
```


版权所有 © 华为技术有限公司

```
negateCommand=undo\u0020
shutdownCommand=\u0020shutdown
operationalDataReadCommand=/display ${context}
contextAccessType=readDatastore
contextSupportsSinglelineCommand=false
defaultDeleteOperationWithValue=false
supportsSinglelineCommand=false
cliInteractionMatchKey=Warning.*(base on this interface, continue| BGP process will be
deleted. Continue).*;^The password needs to be changed\\. Change now\\? \\[Y\\/N\
\].*;^Are you sure you want to continue connecting \\(yes\\/no\\)\\?
cliInteractionGoOnCommond=Y;N;yes
judgeContinueCondition=.*--- More ---*
partialReadSubmode=false
continueReadingCommand=\u0020
interMatherPatterns=^The password needs to be changed\\. Change now\\? \\[Y\\/N\
\];^Are you sure you want to continue connecting \\(yes\\/no\\)\\?
interMatcherCmds=N;yes
```

3. 编写设备特性能力定制

   自定义差异发现，对账和同步使用的特性范围。

```
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function

    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True
        feature_msg.features.extend(self.build_feature('ifm', '', '(http://www.huawei.com/netconf/vrp/
huawei-ifm?revision=2018-11-23)ifm'))
        self.logger.info('getFeatures end.')
        return feature_msg

    def build_feature(self, name, depends, path):
        feature = Feature()
        feature.name = name
        feature.operType = Feature.MERGE
        feature.depends.extend([depends])
        function = Function()
        function.value = path
        function.collectPath = path
        function.preSyncToNe = False
        function.preSyncFromNe = False
        function.postSyncFromNe = False
        feature.functions.extend([function])
        return [feature]
```

----结束

 版权所有 © 华为技术有限公司

6 开发 GND

6.1 开发GND业务包
6.2 开发GND示例

6.1 开发 GND 业务包

GND包分为两种包，模型包和配置实现包：

- GND模型包：存放通用网元层的YANG模型。需要开发YANG模型和对应的权限等功能的配置。
- GND配置实现包：定义通用网元模型层到网元层业务逻辑，并定义网元层业务还原到通用网元模型层的逻辑。需要开发Python代码和Jinja2模板。

开放可编程开发者社区中已经上传了一些典型场景的软件包，用户可以登录社区查询并下载适合的代码样例包。通过对代码样例包进行定制开发，比较高效，开发难度也比较小。如果没有找到适合的代码样例包，用户也可以从开放可编程环境中下载默认GND软件包进行开发。

用户在开发GND软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

6.1.1 创建 GND 模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据**表6-1**和**表6-2**填写软件包的相关属性值，分别导出GND模型包和GND配置实现包模板。

表 6-1 模型包属性

属性名称	属性值
名称	FireWallModel
版本	1.1.11

 版权所有 © 华为技术有限公司

属性名称	属性值
提供商	HUAWEI
描述	A GND Model For FireWall
包类型	Generic NE driver
GND类型	model

表 6-2 配置实现包属性

属性名称	属性值
名称	FireWallMapping
版本	1.0.20
提供商	HUAWEI
描述	A GND Mapping For HW_FireWall
包类型	Generic NE driver
GND类型	configuration

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

6.1.2 开发 GND

请分别开发GND模型包和GND配置实现包。

6.1.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

版权所有 © 华为技术有限公司

表 6-3 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：Specific Service plugin、Generic NE driver、Specific NE driver、SND Migrate plugin。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。仅支持该字段为“huawei”（不区分大小写）的SND和GND包。	必选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，长度不能超过64个字符。 version（必选）字段：版本号三段式规则。	可选
sub-type	GND包类型。	模型包取值为“model”，配置实现包该属性不需要配置。	可选
devices	snd, gnd包特有值，驱动匹配的设备信息。用于软件包的描述，不用于设备纳管。	如果配置，则必须包含： ● 驱动设备厂商vendor（必选），不能超过 128个字符。 ● 具体设备类型device-type（必选），不能超过 128个字符。 ● 具体设备的版本device-version（必选），不能超过 128个字符。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息。相同包里hooks组合唯一，但在不同包里可以相同。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型，EasyMap，type设置为mapping；RPC，type设置为service-rpc；Discover，type设置为discover；GND包特有的值，可以设置为gnd-rpc、gnd-action、port、link等。如需要支持填充在YANG模型中定义参数默认值，type设置为service_settings。data-with-default可以设置为all或explicit，all表示填充定义的参数默认值，explicit表示默认不填充定义的参数默认值。 ● key（必选），回调key值，不能超过128个字符。YANG模型中的“app:application-definition”的值对应key值，即业务定义点。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

GND模型包示例如下，导出后一般不用修改。

```
{
   "name": "FireWallModel",
   "version": "1.1.11",
   "description": "A GND Model For FireWall OF NT",
   "package-type": "gnd",
   "producer": "HUAWEI",
   "nce-min-versions": [
      "1.0.0"
   ],
   "sub-type":"model",
   "package-dependencies": [

   ],
   "devices": [

   ],
   "hooks": [
   ]
}
```

 版权所有 © 华为技术有限公司

GND配置实现包示例如下，“package-dependencies”参数需要配置依赖的GND模型包和SND包的“name”和“version”，“devices”参数配置适配的设备类型和版本，“hooks”参数配置业务回调映射信息。

```
{
   "name": "FireWallMapping",
   "version": "1.0.20",
   "description": "A GND Mapping For HW_FireWall OF NT",
   "package-type": "gnd",
   "producer": "HUAWEI",
   "nce-min-versions": [
      "1.0.0"
   ],
   "package-dependencies": [
    {
      "name": "HillstoneFire-SND",
      "version": "22.0.53"
    },{
        "name": "FireWallModel",
        "version": "1.1.11"
      }
   ],
   "devices": [
    {
      "vendor": "Hillstone",
      "device-type": "SG6000",
      "device-version": "5.5.0"
    }
   ],
   "hooks": [
    {
      "type": "mapping",
      "key": "firewall_address_pool",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRule"
    },
    {
      "type": "discover",
      "key": "firewall_address_pool",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRule"
    },
    {
      "type": "service-rpc",
      "key": "firewall_address_pool",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.RPC"
    },
    {
      "type": "gnd-rpc",
      "key": "firewall_address_pool",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRPC"
    }
   ]
}
```

----结束

6.1.2.2 开发业务 YANG 模型

在集成开发环境左侧导航树中，双击打开yang目录下的.yang文件，编写YANG模型。YANG模型中的“app:application-definition”的值一般对应pkg.json中的hooks下的key，即业务定义点。

当前系统仅支持在application的根节点下挂接list和存在类型container的业务定义点。在父节点上定义业务定义点，则在子节点上不允许再定义业务定义点。

当用户希望对list进行自定义排序，如安全规则list、静态路由list，需要在YANG模型中对应的list打上ordered-by user标记，并在代码中进行处理。

版权所有 © 华为技术有限公司

根据业务模型的北向输入，构建YANG模型。YANG模型中的每一个模块和节点会生成北向UI，由用户输入参数。

开发YANG模型时，分析梳理出配置下发涉及的参数，如表6-4和表6-5所示，并在YANG模型中进行定义。

⚠ 注意

YANG文件命名时需要注意：AOC系统中加载的GND与SSP中，不能存在名字相同的YANG文件。

表 6-4 防火墙规则参数

参数名	参数值	含义
id	10	规则id
action	deny	禁止访问
description	-	-
src_zone	untrust	不可信任区域
dst_zone	trust	信任区域
addr_id	-	地址id
src_addr	src	不可信任区域地址
dst_addr	dst	信任区域地址
service	any	业务
name	policy-test	规则名称

表 6-5 地址池参数

参数名	参数值1	参数值2	含义
address_name	src	dst	地址池名称
description	-	-	描述
member	-	-	-
ip_id	1	2	IP地址ID
ip_address	10.0.0.1 255.255.255.255	10.0.0.2 255.255.255.255	IP地址
range_id	1	-	IP范围ID
address_from	10.0.0.193	-	IP范围起始IP
address_to	10.0.0.198	-	IP范围结尾IP

版权所有 © 华为技术有限公司

编辑完成的YANG模型示例如下。

firewall_rule.yang模型：

```
module firewall_rule {
    yang-version 1;
    namespace "https://www.hillstonenet.com.cn/firewall_rule";
    prefix "firewall_rule";
    revision "2022-03-08" {
        description
          "A Generate NE Driver For Hillstonenet FireWall That contribute to rule_design by LYL";
    }
     import huawei-ac-nes-device {
        prefix nepre;
    }
    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    augment "/nepre:nes/nepre:ne"{
        list policy_rule {
            app:ne-application-definition "firewall_rule";
            key "id_number";
            //ordered-by user;
            //可选功能，当用户希望对list进行自定义排序，如安全规则list、静态路由list等场景时，需要在YANG模
型中对应的list打上ordered-by user标记。
            leaf id_number {
                type uint16 {
                    range "1..12256";
                }
            }
            leaf action {
                type enumeration {
                    enum deny;
                    enum permit;
                }
            }
            leaf description {
                type string {
                    length "1..255";
                }
            }
            leaf src_zone {
                type string;
            }
            leaf dst_zone {
                type string;
            }
            list addr_group{
                key "addr_id";
                leaf addr_id{
                    type uint16;
                }
                leaf src_addr_name{
                    type string {
                        length "1..95";
                    }
                }
                leaf dst_addr_name{
                    type string {
                        length "1..95";
                    }
                }
                leaf service{
```

 版权所有 © 华为技术有限公司

```
                type string;
            }
        }
        leaf name {
            type string {
                length "1..95";
            }
        }
    }
  }
}
```

firewall_address_pool.yang模型：

```
module firewall_address_pool {
    yang-version 1;
    namespace "https://www.hillstonenet.com.cn/firewall_address_pool";
    prefix "firewall_address_pool";
    revision "2022-03-01" {
        description
          "A Generate NE Driver For Hillstonenet FireWall by LYL
          leaf:address-type is used for describing ip_type which has ipv4 and ipv6. In this Model，
          the address_type will be limited in ipv4 and ipv6.";
    }
     import huawei-ac-nes-device {
        prefix nepre;
    }
    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    augment "/nepre:nes/nepre:ne"{
        list address_pool {
            app:ne-application-definition "firewall_address_pool";
            description "For cli command of address test";
            key "address_name";
            leaf address_name {
                type string;
            }
            leaf description {
                type string;
            }
            leaf member {
                type string;
            }
            list ip {
                key "ip_id";
                leaf ip_id {
                    type string;
                }
                leaf ip_address{
                    type inet:ipv4-prefix;
                    mandatory true;
                }
            }

            list range {
                key "range_id";
                leaf range_id {
                    type uint16;
                }
                leaf address_from {
                    type inet:ipv4-address-no-zone;
                    mandatory true;
                }
                leaf address_to {
                    type inet:ipv4-address-no-zone;
```

 版权所有 © 华为技术有限公司

```
                mandatory true;
            }
        }
    }
  }
}
```

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

表 6-6 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d
-	-	type	节点类型，取值范围： ● leaf ● leaf-list ● typedef –自定义类型 默认是leaf

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
   <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
   <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
   <path>/db-security-test:security-top/security-list/security-list-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
   <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

 版权所有 © 华为技术有限公司

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
   <rpcs>
      <rpc>huawei-aaa:reactivateUser</rpc>
      <rpc>huawei-aaa:changeMyPassword</rpc>
      <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
   </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文

 版权所有 © 华为技术有限公司

件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```

6.1.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开GND配置实现包中Python目录下的.py文件，编写Python代码，实现功能逻辑。

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

另外，当通过服务注入包提供了通用的SDK能力或开放内部服务接口时，用户可以直接调用服务注入包提供的接口快速实现功能，提高开发效率。

可以将默认代码全部删除，按照如下示例重新命名py文件和编写代码。需要与pkg.json文件的python-class-name中py文件名称字段保持一致。

说明

实现异步发现逻辑时，需要注意

- 先处理设备同步逻辑，再处理业务同步逻辑。
- 建议限制分批次发现，每批次发现的业务数量在200以内，否则可能导致功能异常。

EasyMap

EasyMap是将GND层业务分解成网元层业务，hook中type需设置为mapping，编写代码时需继承NcsService类并重写ncs_map方法，定义业务映射逻辑。

版权所有 © 华为技术有限公司

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService


class GNDRule(NcsService):

    # 继承NcsService类并重写ncs_map方法，定义业务映射逻辑
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入policy_rule信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['policy_rule']

        # 将neid和policy填充到jinja模板
        res = self.render('policy_rule.j2', {
            'neid': aoccontext.neid,
            'policy': request_dic_node
        })
        self.logger.info('return is %s' % res)
        return res
```

Discover

Discover即业务还原，将网元层业务重新组织成GND层业务，即EasyMap的逆过程。hook中type需设置为discover，编写代码时需继承NcsService类并重写discover方法，定义业务还原逻辑。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService
from aoc.sys import datastore
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverOutput


class GNDRule(NcsService):

    # 继承NcsService类并重写discover方法，定义业务还原逻辑
    def discover(self, discoverInput, aoc_context):
        self.logger.info('The Discover Begin...')

        # 1.从库中读取需要的设备配置数据，准备构造GND yang数据
        ne_len = len(discoverInput.deviceInfo)
        self.logger.info('The deviceInfo is %s' % discoverInput.deviceInfo)
        result = DiscoverOutput()

        for i in range(0, ne_len):
            # 这里的device_id就是neid
            device_id = discoverInput.deviceInfo[i].deviceId
            # SND的yang模型路径
            rule_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + device_id + \
                        '/hillstone-policy:rule'

            # 从SND同步结束后的数据库读取当前路径下的所有数据
            interfaces_configs = datastore.read_datastore_rdb(aoc_context, rule_path)

            self.logger.info("ne device_id is %s" % device_id)
            self.logger.info("ne data is %s" % str(interfaces_configs))

            data_dict = self.xmltodictnode(interfaces_configs)
            # ifm_data = data_dict.get("list-root").get("interface")

            self.logger.info('The data_dict is %s ' % data_dict)

            rule_data = data_dict.get("rule").get('id')

            # self.logger.info("The ifm_data is %s" % ifm_data)

            if not isinstance(rule_data, list):
                rule_data = [rule_data]
```

 版权所有 © 华为技术有限公司

```
        # 2.将SND yang数据转换为 -> GND yang数据

        self.logger.info("The rule_data_list is %s" % rule_data)

        # xml->xml

        for item in rule_data:
            ne_data_res = self.render('discover/policy_rule_discover.j2', {'item': item})

            # 3.输出，将构造好的GND配置数据返回
            # 构造输出
            data = result.serviceConfig.add()
            neid = device_id
            data.servicePath = "/huawei-ac-nes-device:nes/ne/" \
                               + neid + "/policy_rule/" + item.get('id-number')
            data.serviceData = ne_data_res
    return result
```

Action

Action是用户自定义功能，主要用于标准的NETCONF操作无法表达的动作，或不需要保存的一次性操作场景，用户可以灵活自定义功能。

```
# 必选，导入NcsService类，NcsService是aoc_api提供的父类。
from aoc.ncs.ncsservice import NcsService

class AocNcsActionAaaService(NcsAction):

    # 继承NcsService类并重写action方法
    def action(self, aoc_context, action_input):
        # get isp service input data
        self.logger.info("do action")
        self.logger.info(action_input)
        param = action_input.reset-at
        output = {"time": param}
        result = self.render('action_Aaa.j2', output)
        return result
```

RPC

RPC是用户自定义功能，主要用于标准的NETCONF操作无法表达的动作，或不需要保存的一次性操作场景，用户可以灵活自定义功能。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService

class RPC(NcsService):
    rpc_input.inputData = '<input><userName>xxx</userName></input>'
    rpc_input.rpcName = '(nrn:com:huawei)activeUser'
    # 继承NcsService类并重写ncs_rpc方法，自定义功能逻辑
    def ncs_rpc(self, rpc_input, aoc_context, template=None):
        sndinputData = convert(rpc_input.inputData)
        neid = getneid(sndinputData)
        sndrpcName = convert(rpc_input.rpcName)
        sndresult = devicemgr.device_rpc(neid, sndinputData, sndrpcName)
        sspresult = convert(sndresult.result)
        return sspresult
```

gnd-rpc

gnd-rpc和RPC的功能是一样的。用户灵活自定义功能，如一些查询类操作，数据不需要存入数据库。

```
# 导入GndRpc类，该模块提供GND rpc挂钩。用户定义的GND YANG模型包含RPC。当调用GND的RPC时，机制会自动回调用户python脚本实现的RPC处理回调。
from aoc.gnd.GndRpc import GndRpc
```

 版权所有 © 华为技术有限公司

```
from aoc.sys import devicemgr

class GNDRPC(GndRpc):
def rpc(self, aoccontext, request=None):
    sndinputData = convert(request.inputData)
    sndrpcName = convert(request.rpcName)
    sndresult = devicemgr.device_rpc(request.neid, sndinputData,sndrpcName)
    gndresult = convert(sndresult.result)
    output = RpcResult()
    output.outputData = gndresult
    return output
```

ordered-by user

当用户希望对list进行自定义排序，如安全规则list、静态路由list，需要在YANG模型中对应的list打上ordered-by user标记，并在代码中进行处理。

ordered-by user场景，如果是根节点下的list，需要用户编写排序逻辑，如果非根节点下的list，系统自动处理排序逻辑，用户无需编码。

```
    # 自定义排序逻辑时，建议before&last/first、after&last/first组合使用。注意不要before和after混用，否则可能造成排序逻辑矛盾，导致顺序混乱。
    def build_orderedby_user_key(self, aoccontext, template_context):
        ordered_paths = aoccontext.orderedPaths
        path = aoccontext.serviceInst
        # get position of current path instance
        index = ordered_paths.index(path, 0, len(ordered_paths))
        if index is not None:
            if index == (len(ordered_paths) - 1):
                template_context['last'] = True
            else:
                next_path = ordered_paths[index + 1]
                next_ins = next_path.split('/')[len(split_path) - 1]
                template_context['next_ins'] = next_ins
        return template_context
```

6.1.2.4 开发 Jinja2 模板

Jinja2模板开发思路：在纳管设备后导出设备yang模型的j2数据模板。再将对应内容复制到默认j2中，然后编辑j2逻辑，填入参数名变量。

为了方便编码，除了支持Jinja本身的过滤器之外，系统还新增了一些可能用到的过滤器及方法。具体请参考**13.3 Jinja2模板过滤器及方法**。

正向推导方法：

步骤1 在主菜单中选择“设备配置 > 设备配置”。在“设备配置”界面，单击对应设备操作列中的“编辑>特定设备配置”。

步骤2 在中间工具栏中单击“导出”按钮，弹出导出页面。

步骤3 在导出页面中，勾选模型节点，勾选“合并模板”，勾选“带结构不带数据”导出方式，然后单击“确定”。

步骤4 打开导出的模板，保留下发部分，然后把输入参数替换为参数名称变量，可以补充判断逻辑等。保留默认j2文件中“<neid>”标签及之前内容，然后把编辑的内容放到<neid>”标签下面。j2文件重新命名并移动到“template”目录下。

“policy_rule.j2”示例如下：

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
    <nes>
        <ne>
            <neid>{{ neid }}</neid>
```


版权所有 © 华为技术有限公司

```
        {%- for rule in policy %}
        <rule xmlns="https://www.hillstonenet.com.cn/hillstone-policy">
          <id>
            <id-number>{{ rule['id_number'] }}</id-number>
            {%- if 'action' in rule %}
            <action>{{ rule.get('action') }}</action>
            {%- endif %}
            {%- if 'description' in rule %}
            <description>{{ rule.get('description') }}</description>
            {%- endif %}
            {%- if 'src_zone' in rule %}
            <src-zone>{{ rule.get('src_zone') }}</src-zone>
            {%- endif %}
            {%- if 'dst_zone' in rule %}
            <dst-zone>{{ rule.get('dst_zone') }}</dst-zone>
            {%- endif %}
            {%- if rule.get('addr_group') %}
            {%- for item in rule.get('addr_group') %}
              {%- if 'src_addr_name' in item %}
              <src-addr>
                <src-addr-name>{{ item.src_addr_name }}</src-addr-name>
              </src-addr>
              {%- endif %}
              {%- if 'dst_addr_name' in item %}
              <dst-addr>
                <dst-addr-name>{{ item.dst_addr_name }}</dst-addr-name>
              </dst-addr>
              {%- endif %}
              {%- if 'service' in item %}
              <service>
                <service-name>{{ item.service }}</service-name>
              </service>
              {%- endif %}
            {%- endfor %}
            {%- endif %}
            {%- if 'name' in rule %}
            <name>{{ rule.get('name') }}</name>
            {%- endif %}
          </id>
        </rule>
        {%- endfor %}
      </ne>
    </nes>
</inventory-cfg>
```

----结束

6.1.3 验证业务包

6.1.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

版权所有 © 华为技术有限公司

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

6.1.3.2 执行单元测试

步骤1 使用“yang-offline-util.zip”工具，根据YANG模型生成空的NETCONF报文。请执行如下命令。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

当运行结果为空时，表明“subtree.xml”文件成功生成。

步骤2 打开“subtree.xml”文件，并在<application>标签下各标签中填上适当的值。

policy_rule：

```
<policy_rule xmlns="https://www.hillstonenet.com.cn/firewall_rule">
  <id_number>10</id_number>
  <action>deny</action>
  <description></description>
  <src_zone>untrust</src_zone>
  <dst_zone>trust</dst_zone>
  <addr_group>
    <addr_id>111</addr_id>
    <src_addr>src</src_addr>
    <dst_addr>dst</dst_addr>
    <service>any</service>
  </addr_group>
  <name>policy-test</name>
</policy_rule>
```

address_pool：

```
<address_pool xmlns="https://www.hillstonenet.com.cn/firewall_address_pool">
  <address_name>src</address_name>
  <description></description>
  <member></member>
  <ip>
    <ip_id>1</ip_id>
    <ip_address>10.0.0.1 255.255.255.255</ip_address>
  </ip>
  <range>
    <range_id>1</range_id>
    <address_from>10.0.0.193</address_from>
    <address_to>10.0.0.198</address_to>
  </range>
</address_pool>
```

步骤3 双击打开test目录下的.py文件，将前面步骤修改好的<application>标签内的代码拷贝粘贴到如下xml语句后面，并补充模拟调用代码。示例如下。

test_gnd_rule_service.py代码：

```
import unittest
import sys
from aoc.ncs.ncsservice import DictNode, MapRequest

sys.path.insert(0, "../../python")
from com.huawei.controller.devicetype.gnd_rule import GNDRule

class Test(unittest.TestCase):
  xml = '''
  <policy_rule xmlns="https://www.hillstonenet.com.cn/firewall_rule">
    <id_number>10</id_number>
```

 版权所有 © 华为技术有限公司

```
    <action>deny</action>
    <description></description>
    <src_zone>untrust</src_zone>
    <dst_zone>trust</dst_zone>
    <addr_group>
      <addr_id>111</addr_id>
      <src_addr>src</src_addr>
      <dst_addr>dst</dst_addr>
      <service>any</service>
    </addr_group>
    <name>policy-test</name>
  </policy_rule>
  '''

  def test_case1(self):
      #input = MapRequest(self.xml, None)
      #request_dic_node = request.xmldictnode(self.xml,None)
      aoccontex = DictNode({"neid": "45454e3f-5ceb-4209-9f7d-605d83ce9328"})
      result = GNDRule().ncs_map(MapRequest(self.xml, None), aoccontex)
      print(result)

if __name__ == "__main__":
  unittest.main()
```

test_pool_ipv4_service.py代码

```
import unittest
import sys
from aoc.ncs.ncsservice import DictNode, MapRequest
sys.path.insert(0, "../../python")
from com.huawei.controller.devicetype.gnd_rule import GNDRule
from com.huawei.controller.devicetype.pool_ipv4 import AddressPool


class Test(unittest.TestCase):
  xml = '''
  <address_pool xmlns="https://www.hillstonenet.com.cn/firewall_address_pool">
    <address_name>src</address_name>
    <description></description>
    <member></member>
    <ip>
      <ip_id>1</ip_id>
      <ip_address>10.0.0.1 255.255.255.255</ip_address>
    </ip>
    <range>
      <range_id>1</range_id>
      <address_from>10.0.0.193</address_from>
      <address_to>10.0.0.198</address_to>
    </range>
  </address_pool>
  '''

  def test_case1(self):
      #input = MapRequest(self.xml, None)
      #request_dic_node = request.xmldictnode(self.xml,None)
      aoccontex = DictNode({"neid": "45454e3f-5ceb-4209-9f7d-605d83ce9328"})

      result = AddressPool().ncs_map(MapRequest(self.xml, None), aoccontex)
      print(result)

if __name__ == "__main__":
  unittest.main()
```

步骤4 运行该测试代码，可看到生成的报文，检查输出是否正确。

回显提示“Process finished with exit code 0”，表明单元测试成功。否则请仔细检查Jinja模板文件和YANG模型文件中各属性是否一一对应。

----结束

 版权所有 © 华为技术有限公司

6.1.4 编译业务包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件移动到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

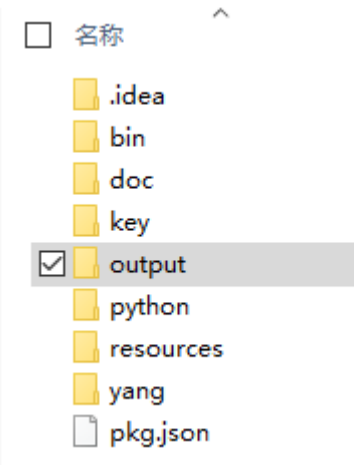
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2021-11-04 18:10:12,291 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Generate signature file success.
2021-11-04 18:10:12,542 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

6.2 开发 GND 示例

6.2.1 开发 AAAmini GND 示例

本示例介绍如何通过下载的默认GND包，开发一个AAAmini GND包，创建一个用户，并设置密码。GND需要开发两个子包，GND模型包和GND配置实现包，下面分别介绍。

版权所有 © 华为技术有限公司

开发 GND 模型包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。
   请根据表6-7填写软件包的相关属性值。

   表 6-7 模型包属性

属性名称	属性值
名称	aaamini
版本	1.0.1
提供商	HUAWEI
描述	GND Model
包类型	Generic NE driver
GND类型	model

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "aaaminigndmodel",
   "version": "1.0.1",
   "description": "gnd model",
   "package-type": "gnd",
   "producer": "huawei",
   "nce-min-versions": [
      "1.0.0"
   ],
   "sub-type":"model",
   "package-dependencies": [
   ],
   "devices": [

   ],
   "hooks": [

   ]
}
```

步骤3 编写设备YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表6-8所示，并在YANG模型中进行定义。

版权所有 © 华为技术有限公司

表 6-8 输入参数

类别	描述	参数名称	参数值
AAA用户	创建AAA用户用于认证。	username	aaaminignd
		password	***

最终编辑完成的YANG模型示例如下。

```
module aaaminignd {
    namespace "http://swisscom.com/aaaminignd";
    prefix aaaminignd;

    import ietf-yang-types {
        prefix yang;
    }

    import huawei-pub-type {
        prefix pub-type;
    }

    import huawei-ac-nes-device {
        prefix nepre;
    }

    import huawei-ac-applications {
        prefix app;
    }

    description "This module can configure and manage Huawei NE40-X3";

    revision 2019-05-30 {
        description "Initial revision.";
    }

    augment "/nepre:nes/nepre:ne"{
        list aaaminignd{
            app:ne-application-definition "aaaminignd";
            uses app:application-optional-para;
            description "aaa parameters";
            key "username";
            leaf username{
                type string;
            }
            leaf password {
                type pub-type:passwordExtend {
                }
            }
        }
    }
/*end of augment /nepre:nes/nepre:ne*/
}
```

----结束

开发 GND 配置实现包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

 版权所有 © 华为技术有限公司

2. 并在弹出的对话框中设置软件包的相关属性。
   请根据表6-9填写软件包的相关属性值。

   表 6-9 配置实现包属性

属性名称	属性值
名称	aaamini
版本	1.0.1
提供商	HUAWEI
描述	GND Mapping
包类型	Generic NE driver
GND类型	configuration

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

GND配置实现包中，“package-dependencies”参数需要配置依赖的GND模型包和SND包的“name”和“version”，“devices”参数配置适配的设备类型和版本，“hooks”参数配置业务回调映射信息。

```
{
  "name": "aaaminigndservice",
  "version": "1.0.1",
  "description": "GND Mapping",
  "package-type": "gnd",
  "producer": "huawei",
  "nce-min-versions": [
     "1.0.0"
  ],
  "package-dependencies": [
     {
        "name": "aaaminigndmodel",
        "version": "1.0.1"
     }
  ],
  "devices": [
   {
     "vendor": "HUAWEI",
     "device-type": "NE40E-X8A(V8)",
     "device-version": "V800R010C10SPC500"
   },
   {
     "vendor": "HUAWEI",
     "device-type": "NE40E-X3A(V8)",
     "device-version": "V800R010C10SPC500"
   }
  ],
  "hooks": [
   {
     "type": "mapping",
     "key": "aaaminignd",
     "python-class-name": "com.huawei.aaa.AocNcsAaa.AocNcsAaaService"
   },{
     "type": "validate",
```

 版权所有 © 华为技术有限公司

```
      "key": "aaaminignd",
      "python-class-name": "com.huawei.aaa.AocNcsAaa.AocNcsAaaService"
    },
    {
      "type": "mapping",
      "key": "profile",
      "python-class-name": "com.huawei.aaa.AocNcsAaa.AocNcsAaaService"
    },
    {
      "type": "mapping",
      "key": "user-group-queue",
      "python-class-name": "com.huawei.aaa.AocNcsAaa.AocNcsAaaService"
    },
    {
      "type": "discover",
      "key": "aaaminignd",
      "python-class-name": "com.huawei.aaa.AocNcsAaa.AocNcsAaaService"
    }
   ]
}
```

步骤3 开发代码

- 开发gnd_rule.py代码，将基于yang输入policy_rule信息填充到对应jinja模板中。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService
# 导入DiscoverInput，DiscoverOutput接口，重写discover方法时需要使用
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverInput, DiscoverOutput
# 导入datastore接口，使用read_datastore_rdb方法时需要使用
from aoc.sys import datastore
# 导入sleep,time接口，使用sleep等方法时需要使用
from time import sleep,time


class AocNcsAaaService(NcsService):

    # 继承NcsService类并重写ncs_map方法，定义业务映射逻辑
    def ncs_map(self, request, aoccontext=None, template=None):

        # 获取业务输入数据
        request_dic_node = self.xmltodictnode(request.xml)
        self.logger.info("bngAaaPairs request_dic_node: %s" % (request_dic_node))

        # 查询业务数据
        template_context = self.fill_miniaaapair_context(request_dic_node, aoccontext)
        self.logger.info("bngAaaPairs template_syspair_context: %s" % (template_context))

        # ordered-by user场景,自定义排序逻辑
        template_context = build_orderedby_user_key

        # 用template_context数据渲染j2模板
        strtttt = self.render('template_Aaa.j2', template_context)
        self.logger.info("bngAaaPairs template_nepair%s" % (strtttt))
        return strtttt


    # 自定义排序逻辑时，建议before&last/first、after&last/first组合使用。注意不要before和after混用，
否则可能造成排序逻辑矛盾，导致顺序混乱。
    def build_orderedby_user_key(self, aoccontext, template_context):
        ordered_paths = aoccontext.orderedPaths
        path = aoccontext.serviceInst
        # get position of current path instance
        index = ordered_paths.index(path, 0, len(ordered_paths))
        if index is not None:
            if index == (len(ordered_paths) - 1):
                template_context['last'] = True
            else:
                next_path = ordered_paths[index + 1]
                next_ins = next_path.split('/')[len(split_path) - 1]
                template_context['next_ins'] = next_ins
```

 版权所有 © 华为技术有限公司

```
        return template_context


    # 查询neid
    def fill_miniaaapair_context(self, request_dic_node, aoccontext):
        self.aaamini = request_dic_node.aaaminignd
        servicepath = aoccontext.serviceInst
        nerootpath = aoccontext.serviceInst[servicepath.find("ne/")+3:]
        neid = nerootpath[:nerootpath.find("/")]
        return {'users':self.aaamini,'neid':neid}


    def ncs_validate(self,aoc_context,validate_input):
        self.logger.info('ncs_validate:%s' % validate_input)
        if '222' in validate_input.validateConfig.data:
            self.logger.info("delete validate failed")
            raise ValidateException("validate failed")


    # 定义业务还原逻辑
    def discover(self, discoverInput, aoc_context):
        self.logger.info(discoverInput)
        self.logger.info(aoc_context)

        discoveryId = discoverInput.discoveryId
        cacheContext = discoverInput.cacheContext

        # 从discover_input中获取设备信息
        if discoverInput.specificInstance is not None:
            inst = discoverInput.specificInstance
            self.logger.info("specificInstance: %s" % inst)
            if len(inst) != 0 :
                cont = inst.split('/')
                if len(cont) > 0:
                    name = cont[-1]
                    self.logger.info("specific name: %s" % name)
                    for device in discoverInput.deviceInfo:
                        deviceId = device.deviceId
                        deviceName = device.deviceName

                        # 构造env
                        env = dict()
                        env['device'] = deviceName
                        path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + '/huawei-aaa:aaa/lam/
users/user/' + name
                        users = datastore.read_datastore_rdb(aoc_context, path)
                        root = self.xmltodictnode(users)
                        self.logger.info("specific name users: %s" % root)
                        for user in root.user:
                            result = DiscoverOutput()
                            self.parse_rdb_user(env, user, result)
                            result.discoveryId = discoveryId
                            cacheContext[user.userName] = "exist"
                            for key in cacheContext:
                                self.logger.info("user %s, time  %s" % (cacheContext[key], time()))
                                result.cacheContext[key] = cacheContext[key]

                            result.goOn = False
                            self.logger.info("specific result: %s" % result)
                            return result

        self.logger.info("start real discovery: %s---%s" % (discoveryId, time()))
        try:
            for device in discoverInput.deviceInfo:
                deviceId = device.deviceId
                deviceName = device.deviceName

                # 构造env
                env = dict()
```


版权所有 © 华为技术有限公司

```
            env['device'] = deviceName

            # 读取设备RDB
            path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + '/huawei-aaa:aaa/lam/users'
            users = datastore.read_datastore_rdb(aoc_context, path)
            root = self.xmltodictnode(users)

            # result = DiscoverOutput()
            # 解析RDB结果
            for user in root.users.user:
                if user.userName == 'test5':
                    continue
                if cacheContext is not None and cacheContext[user.userName] is not None and
cacheContext[user.userName] == 'exist':
                    self.logger.info("user %s exist, time  %s" % (user.userName, time()))
                    continue
                #单个用户暂停3s
                result = DiscoverOutput()
                self.parse_rdb_user(env, user, result)
                result.discoveryId = discoveryId
                cacheContext[user.userName] =  "exist"
                for key in cacheContext:
                    self.logger.info("user %s, time  %s" % (cacheContext[key], time()))
                    result.cacheContext[key] = cacheContext[key]

                result.goOn = True
                self.logger.info(result)
                self.logger.info("time  %s" % time())
                sleep(1)
                return result
        except Exception as ex:
            self.logger.exception('xxx, Exception occur. %s' % ex)
            self.logger.info("xxxx end real discovery: %s---%s" % (discoveryId, time()))
            result = DiscoverOutput()
            result.discoveryId = discoveryId
            result.goOn = False
            # result.cacheContext = None
            return result

    def parse_rdb_user(self, env, user, result):
        aaamin = result.serviceConfig.add()
        aaamin.servicePath = "/huawei-ac-nes-device:nes/ne/%s/aaaminignd:aaaminignd/%s" %
(self.to_ne_id(env['device']), user.userName)
        env['username'] = user.userName
        if user.password is None or len(user.password) == 0:
            env['password'] = '123456'
        else:
            env['password'] = user.password
        aaamin.serviceData = self.render('aaamingnd-discover.j2', env)
```

步骤4 开发Jinja2模板

分为两个Jinja2文件，“template_Aaa.j2”循环查找user，“hbng-aaa.j2”处理模板内容。

“template_Aaa.j2”

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
   <nes>
{%- for user in users %}
      <ne>
        <neid>{{neid}}</neid>
        {%- include 'com/huawei/aaa/hbng-aaa.j2' %}
      </ne>
{%- endfor %}
   </nes>
</inventory-cfg>
```

“hbng-aaa.j2”

 版权所有 © 华为技术有限公司

```
<aaa xmlns="http://www.huawei.com/netconf/vrp/huawei-aaa">
 <lam>
  <users>
 {# ordered-by user场景，用户需要在Jinja2模板中编写排序处理逻辑。非ordered-by user场景，用户只需要else代码。#}
    {% if last %}
      <user xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="last">
    {% elif next_ins %}
      <user xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="before" ns0:key="[username='{{ next_ins }}']">
    {% else %}
       <user>
    {% endif %}
         <userName>{{user.username}}</userName>
         <password>{{user.password}}</password>
       </user>
  </users>
 </lam>
</aaa>
```

----结束

6.2.2 开发 Hillstone FireWall GND 示例

本示例介绍如何通过下载的默认GND包，开发一个Hillstone FireWall GND包，创建防火墙策略。GND需要开发两个子包，GND模型包和GND配置实现包，下面分别介绍。

开发 GND 模型包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表6-10填写软件包的相关属性值。

   表 6-10 模型包属性

属性名称	属性值
名称	FireWallModel
版本	1.1.11
提供商	HUAWEI
描述	A GND Model For FireWall
包类型	Generic NE driver
GND类型	model

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "FireWallModel",
```


版权所有 © 华为技术有限公司

```
  "version": "1.1.11",
  "description": "A GND Model For FireWall",
  "package-type": "gnd",
  "producer": "HUAWEI",
  "nce-min-versions": [
      "1.0.0"
  ],
  "sub-type":"model",
  "package-dependencies": [

  ],
  "devices": [

  ],
  "hooks": [
  ]
}
```

步骤3 编写设备YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表6-11和表6-12所示，并在YANG模型中进行定义。

表 6-11 防火墙规则参数

参数名	参数值	含义
id	10	规则id
action	deny	禁止访问
description	-	-
src_zone	untrust	不可信任区域
dst_zone	trust	信任区域
addr_id	-	地址id
src_addr	src	不可信任区域地址
dst_addr	dst	信任区域地址
service	any	业务
name	policy-test	规则名称

表 6-12 地址池参数

参数名	参数值1	参数值2	含义
address_name	src	dst	地址池名称
description	-	-	描述
member	-	-	-
ip_id	1	2	IP地址ID

版权所有 © 华为技术有限公司

参数名	参数值1	参数值2	含义
ip_address	10.1.1.1 255.255.255.255	10.2.2.2 255.255.255.255	IP地址
range_id	1	-	IP范围ID
address_from	10.125.252.193	-	IP范围起始IP
address_to	10.125.252.198	-	IP范围结尾IP

最终编辑完成的YANG模型示例如下。

firewall_rule.yang模型：

```
module firewall_rule {
    yang-version 1;
    namespace "https://www.hillstonenet.com.cn/firewall_rule";
    prefix "firewall_rule";
    revision "2022-03-08" {
        description
          "A Generate NE Driver For Hillstonenet FireWall That contribute to rule_design by LYL";
    }
     import huawei-ac-nes-device {
        prefix nepre;
    }
    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    augment "/nepre:nes/nepre:ne"{
        list policy_rule {
            app:ne-application-definition "firewall_rule";
            key "id_number";
            leaf id_number {
                type uint16 {
                    range "1..12256";
                }
            }
            leaf action {
                type enumeration {
                    enum deny;
                    enum permit;
                }
            }
            leaf description {
                type string {
                    length "1..255";
                }
            }
            leaf src_zone {
                type string;
            }
            leaf dst_zone {
                type string;
            }
            list addr_group{
                key "addr_id";
                leaf addr_id{
                    type uint16;
                }
                leaf src_addr_name{
```


版权所有 © 华为技术有限公司

```
                type string {
                    length "1..95";
                }
            }
            leaf dst_addr_name{
                type string {
                    length "1..95";
                }
            }
            leaf service{
                type string;
            }
        }
        leaf name {
            type string {
                length "1..95";
            }
        }
    }
  }
}
```

firewall_address_pool.yang模型：

```
module firewall_address_pool {
  yang-version 1;
  namespace "https://www.hillstonenet.com.cn/firewall_address_pool";
  prefix "firewall_address_pool";
  revision "2022-03-01" {
    description
      "A Generate NE Driver For Hillstonenet FireWall by LYL
      leaf:address-type is used for describing ip_type which has ipv4 and ipv6. In this Model,
      the address_type will be limited in ipv4 and ipv6.";
  }
   import huawei-ac-nes-device {
    prefix nepre;
  }
  import huawei-ac-applications {
    prefix app;
  }
  import ietf-inet-types {
    prefix inet;
  }

  augment "/nepre:nes/nepre:ne"{
    list address_pool {
      app:ne-application-definition "firewall_address_pool";
      description "For cli command of address test";
      key "address_name";
      leaf address_name {
        type string;
      }
      leaf description {
        type string;
      }
      leaf member {
        type string;
      }
      list ip {
        key "ip_id";
        leaf ip_id {
          type string;
        }
        leaf ip_address{
          type inet:ipv4-prefix;
          mandatory true;
        }
      }
```

 版权所有 © 华为技术有限公司

```
        list range {
            key "range_id";
            leaf range_id {
                type uint16;
            }
            leaf address_from {
                type inet:ipv4-address-no-zone;
                mandatory true;
            }
            leaf address_to {
                type inet:ipv4-address-no-zone;
                mandatory true;
            }
        }
    }
  }
}
```

----结束

开发 GND 配置实现包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表6-13填写软件包的相关属性值。

   表 6-13 配置实现包属性

属性名称	属性值
名称	FireWallMapping
版本	1.0.20
提供商	HUAWEI
描述	A GND Mapping For HW_FireWall
包类型	Generic NE driver
GND类型	configuration

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

GND配置实现包中，“package-dependencies”参数需要配置依赖的GND模型包和SND包的“name”和“version”，“devices”参数配置适配的设备类型和版本，“hooks”参数配置业务回调映射信息。

```
{
  "name": "FireWallMapping",
  "version": "1.0.20",
  "description": "A GND Mapping For HW_FireWall",
  "package-type": "gnd",
```

 版权所有 © 华为技术有限公司

```
  "producer": "HUAWEI",
  "nce-min-versions": [
      "1.0.0"
  ],
  "package-dependencies": [
    {
      "name": "HillstoneFire-SND",
      "version": "22.0.53"
    },{
        "name": "FireWallModel",
        "version": "1.1.11"
      }
  ],
  "devices": [
    {
      "vendor": "Hillstone",
      "device-type": "SG6000",
      "device-version": "5.5.0"
    }
  ],
  "hooks": [
    {
      "type": "mapping",
      "key": "firewall_address_pool",
      "python-class-name": "com.huawei.controller.devicetype.pool_ipv4.AddressPool"
    },
    {
      "type": "mapping",
      "key": "firewall_rule",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRule"
    }
  ]
}
```

步骤3 开发代码

- 开发gnd_rule.py代码，将基于yang输入policy_rule信息填充到对应jinja模板中。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService


class GNDRule(NcsService):

    # 继承NcsService类并重写ncs_map方法，定义业务映射逻辑
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入policy_rule信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['policy_rule']

        # 将neid和policy填充到jinja模板
        res = self.render('policy_rule.j2', {
            'neid': aoccontext.neid,
            'policy': request_dic_node
        })
        self.logger.info('return is %s' % res)
        return res
```

- 开发pool_ipv4.py代码，将基于yang输入address_pool信息填充到对应jinja模板中。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService


class AddressPool(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入address_pool信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['address_pool']

        # 将neid和address_pool填充到jinja模板
```

 版权所有 © 华为技术有限公司

```
        res = self.render('pool_ipv4.j2', {"address_pool": request_dic_node,
                                           'neid': aoccontext.neid})
        self.logger.info('return is %s' % res)
        return res
```

步骤4 开发Jinja2模板

policy_rule.j2

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
    <ne>
      <neid>{{ neid }}</neid>
      {%- for rule in policy %}
      <rule xmlns="https://www.hillstonenet.com.cn/hillstone-policy">
        <id>
          <id-number>{{ rule['id_number'] }}</id-number>
          {%- if 'action' in rule %}
          <action>{{ rule.get('action') }}</action>
          {%- endif %}
          {%- if 'description' in rule %}
          <description>{{ rule.get('description') }}</description>
          {%- endif %}
          {%- if 'src_zone' in rule %}
          <src-zone>{{ rule.get('src_zone') }}</src-zone>
          {%- endif %}
          {%- if 'dst_zone' in rule %}
          <dst-zone>{{ rule.get('dst_zone') }}</dst-zone>
          {%- endif %}
          {%- if rule.get('addr_group') %}
          {%- for item in rule.get('addr_group') %}
            {%- if 'src_addr_name' in item %}
            <src-addr>
              <src-addr-name>{{ item.src_addr_name }}</src-addr-name>
            </src-addr>
            {%- endif %}
            {%- if 'dst_addr_name' in item %}
            <dst-addr>
              <dst-addr-name>{{ item.dst_addr_name }}</dst-addr-name>
            </dst-addr>
            {%- endif %}
            {%- if 'service' in item %}
            <service>
              <service-name>{{ item.service }}</service-name>
            </service>
            {%- endif %}
          {%- endfor %}
          {%- endif %}
          {%- if 'name' in rule %}
          <name>{{ rule.get('name') }}</name>
          {%- endif %}
        </id>
      </rule>
      {%- endfor %}
    </ne>
  </nes>
</inventory-cfg>
```

pool_ipv4.j2

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
    <ne>
      <neid>{{ neid }}</neid>
      {%- for pool in address_pool %}
        <address-pool xmlns="https://www.hillstonenet.com.cn/hillstone-pool">
          <address>
          {%- if 'description' in pool %}
            <description>{{ pool.get('description') }}</description>
```

 版权所有 © 华为技术有限公司

```
            {%- endif %}
            {%- if 'member' in pool %}
               <member>{{ pool.get('member') }}</member>
            {%- endif %}
               <address-name>{{ pool.get('address_name') }}</address-name>
            {%- if pool.ip %}
            {%- for item in pool.ip %}
               <ip>
                  <addr>{{ item.get('ip_address') }}</addr>
               </ip>
            {%- endfor %}
            {%- endif %}
            {%- if pool.range %}
            {%- for item in pool.range %}
               <range>
                 <address-start>{{ item.get('address_from')}}</address-start>
                 <address-end>{{ item.get('address_to')}}</address-end>
               </range>
            {%- endfor %}
            {%- endif %}
            </address>
         </address-pool>
      {%- endfor %}
   </ne>
 </nes>
</inventory-cfg>
```

----结束

6.2.3 开发 Hillstone FireWall 业务还原 GND 示例

本示例介绍如何通过下载的默认GND包，开发一个Hillstone FireWall GND业务还原包，将设备上创建的防火墙策略配置还原到GND业务配置上。GND需要开发两个子包，GND模型包和GND配置实现包，下面分别介绍。

开发 GND 模型包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表6-14填写软件包的相关属性值。

表 6-14 模型包属性

属性名称	属性值
名称	FireWallModel
版本	1.1.11
提供商	HUAWEI
描述	A GND Model For FireWall
包类型	Generic NE driver
GND类型	model

版权所有 © 华为技术有限公司

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "FireWallModel",
   "version": "1.1.11",
   "description": "A GND Model For FireWall",
   "package-type": "gnd",
   "producer": "HUAWEI",
   "nce-min-versions": [
       "1.0.0"
   ],
   "sub-type":"model",
   "package-dependencies": [

   ],
   "devices": [

   ],
   "hooks": [
   ]
}
```

步骤3 编写设备YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表6-15和表6-15所示，并在YANG模型中进行定义。

表 6-15 防火墙规则参数

参数名	参数值	含义
id	10	规则id
action	deny	禁止访问
description	-	-
src_zone	untrust	不可信任区域
dst_zone	trust	信任区域
addr_id	-	地址id
src_addr	src	不可信任区域地址
dst_addr	dst	信任区域地址
service	any	业务
name	policy-test	规则名称

表 6-16 地址池参数

参数名	参数值1	参数值2	含义
address_name	src	dst	地址池名称

版权所有 © 华为技术有限公司

参数名	参数值1	参数值2	含义
description	-	-	描述
member	-	-	-
ip_id	1	2	IP地址ID
ip_address	10.1.1.1 255.255.255.255	10.2.2.2 255.255.255.255	IP地址
range_id	1	-	IP范围ID
address_from	10.125.252.193	-	IP范围起始IP
address_to	10.125.252.198	-	IP范围结尾IP

最终编辑完成的YANG模型示例如下。

firewall_rule.yang模型：

```
module firewall_rule {
    yang-version 1;
    namespace "https://www.hillstonenet.com.cn/firewall_rule";
    prefix "firewall_rule";
    revision "2022-03-08" {
        description
          "A Generate NE Driver For Hillstonenet FireWall That contribute to rule_design by LYL";
    }
     import huawei-ac-nes-device {
        prefix nepre;
    }
    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    augment "/nepre:nes/nepre:ne"{
        list policy_rule {
            app:ne-application-definition "firewall_rule";
            key "id_number";
            leaf id_number {
                type uint16 {
                    range "1..12256";
                }
            }
            leaf action {
                type enumeration {
                    enum deny;
                    enum permit;
                }
            }
            leaf description {
                type string {
                    length "1..255";
                }
            }
            leaf src_zone {
                type string;
            }
            leaf dst_zone {
                type string;
```


版权所有 © 华为技术有限公司

```
        }
        list addr_group{
            key "addr_id";
            leaf addr_id{
                type uint16;
            }
            leaf src_addr_name{
                type string {
                    length "1..95";
                }
            }
            leaf dst_addr_name{
                type string {
                    length "1..95";
                }
            }
            leaf service{
                type string;
            }
        }
        leaf name {
            type string {
                length "1..95";
            }
        }
    }
  }
}
```

firewall_address_pool.yang模型：

```
module firewall_address_pool {
    yang-version 1;
    namespace "https://www.hillstonenet.com.cn/firewall_address_pool";
    prefix "firewall_address_pool";
    revision "2022-03-01" {
        description
          "A Generate NE Driver For Hillstonenet FireWall by LYL
          leaf:address-type is used for describing ip_type which has ipv4 and ipv6. In this Model，
          the address_type will be limited in ipv4 and ipv6.";
    }
     import huawei-ac-nes-device {
        prefix nepre;
    }
    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    augment "/nepre:nes/nepre:ne"{
        list address_pool {
            app:ne-application-definition "firewall_address_pool";
            description "For cli command of address test";
            key "address_name";
            leaf address_name {
                type string;
            }
            leaf description {
                type string;
            }
            leaf member {
                type string;
            }
            list ip {
                key "ip_id";
                leaf ip_id {
                    type string;
```

 版权所有 © 华为技术有限公司

```
            }
            leaf ip_address{
                type inet:ipv4-prefix;
                mandatory true;
            }
        }

        list range {
            key "range_id";
            leaf range_id {
                type uint16;
            }
            leaf address_from {
                type inet:ipv4-address-no-zone;
                mandatory true;
            }
            leaf address_to {
                type inet:ipv4-address-no-zone;
                mandatory true;
            }
        }
    }
  }
}
```

----结束

开发 GND 配置实现包

步骤1 创建GND模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表6-17填写软件包的相关属性值。

   表 6-17 配置实现包属性

属性名称	属性值
名称	FireWallMapping
版本	1.0.20
提供商	HUAWEI
描述	A GND Mapping For HW_FireWall
包类型	Generic NE driver
GND类型	configuration

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

 版权所有 © 华为技术有限公司

GND配置实现包中，“package-dependencies”参数需要配置依赖的GND模型包和SND包的“name”和“version”，“devices”参数配置适配的设备类型和版本，“hooks”参数配置业务回调映射信息。

```
{
   "name": "FireWallMapping",
   "version": "1.0.20",
   "description": "A GND Mapping For HW_FireWall",
   "package-type": "gnd",
   "producer": "HUAWEI",
   "nce-min-versions": [
      "1.0.0"
   ],
   "package-dependencies": [
    {
      "name": "HillstoneFire-SND",
      "version": "22.0.53"
    },{
        "name": "FireWallModel",
        "version": "1.1.11"
      }
   ],
   "devices": [
    {
      "vendor": "Hillstone",
      "device-type": "SG6000",
      "device-version": "5.5.0"
    }
   ],
   "hooks": [
    {
      "type": "mapping",
      "key": "firewall_rule",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRule"
    },
    {
      "type": "discover",
      "key": "firewall_rule",
      "python-class-name": "com.huawei.controller.devicetype.gnd_rule.GNDRule"
    }
   ]
}
```

步骤3 开发代码

- 开发gnd_rule.py代码，将基于yang输入policy_rule信息填充到对应jinja模板中。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService
from aoc.sys import datastore
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverOutput


class GNDRule(NcsService):

    # 继承NcsService类并重写discover方法，定义业务还原逻辑
    def discover(self, discoverInput, aoc_context):
        self.logger.info('The Discover Begin...')

        # 1.从库中读取需要的设备配置数据，准备构造GND yang数据
        ne_len = len(discoverInput.deviceInfo)
        self.logger.info('The deviceInfo is %s' % discoverInput.deviceInfo)
        result = DiscoverOutput()

        for i in range(0, ne_len):
            # 这里的device_id就是neid
            device_id = discoverInput.deviceInfo[i].deviceId
            # SND的yang模型路径
            rule_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + device_id + \
                        '/hillstone-policy:rule'
```

 版权所有 © 华为技术有限公司

```
        # 从SND同步结束后的数据库读取当前路径下的所有数据
        interfaces_configs = datastore.read_datastore_rdb(aoc_context, rule_path)

        self.logger.info("ne device_id is %s" % device_id)
        self.logger.info("ne data is %s" % str(interfaces_configs))

        data_dict = self.xmltodictnode(interfaces_configs)
        # ifm_data = data_dict.get("list-root").get("interface")

        self.logger.info('The data_dict is %s ' % data_dict)

        rule_data = data_dict.get("rule").get('id')

        # self.logger.info("The ifm_data is %s" % ifm_data)

        if not isinstance(rule_data, list):
            rule_data = [rule_data]

        # 2.将SND yang数据转换为 -> GND yang数据

        self.logger.info("The rule_data_list is %s" % rule_data)

        # xml->xml

        for item in rule_data:
            ne_data_res = self.render('discover/policy_rule_discover.j2', {'item': item})

            # 3.输出，将构造好的GND配置数据返回
            # 构造输出
            data = result.serviceConfig.add()
            neid = device_id
            data.servicePath = "/huawei-ac-nes-device:nes/ne/" \
                               + neid + "/policy_rule/" + item.get('id-number')
            data.serviceData = ne_data_res
    return result
```

- 开发pool_ipv4.py代码，将基于yang输入address_pool信息填充到对应jinja模板中。

```
# 导入NcsService类，NcsService是AOC提供的基类，需要重写以实现业务YANG数据到Jinja模板的映射
from aoc.ncs.ncsservice import NcsService
from aoc.sys import datastore
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverOutput


class AddressPool(NcsService):
    def discover(self, discoverInput, aoc_context):
        self.logger.info('The Discover Begin...')

        # 1.从库中读取需要的设备配置数据，准备构造GND yang数据
        ne_len = len(discoverInput.deviceInfo)
        self.logger.info('The deviceInfo is %s' % discoverInput.deviceInfo)
        result = DiscoverOutput()

        for i in range(0, ne_len):
            # 这里的device_id就是neid
            device_id = discoverInput.deviceInfo[i].deviceId
            # SND的yang模型路径
            rule_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + device_id + \
                        '/hillstone-pool:address-pool'

            # 从SND同步结束后的数据库读取当前路径下的所有数据
            interfaces_configs = datastore.read_datastore_rdb(aoc_context, rule_path)

            self.logger.info("ne device_id is %s" % device_id)
            self.logger.info("ne data is %s" % str(interfaces_configs))

            data_dict = self.xmltodictnode(interfaces_configs)
            # ifm_data = data_dict.get("list-root").get("interface")

            self.logger.info('The data_dict is %s ' % data_dict)
```

 版权所有 © 华为技术有限公司

```
        rule_data = data_dict.get("address-pool").get('address')

        # self.logger.info("The ifm_data is %s" % ifm_data)

        if not isinstance(rule_data, list):
            rule_data = [rule_data]

        # 2.将SND yang数据转换为 -> GND yang数据
        self.logger.info("The rule_data_list is %s" % rule_data)

        # xml->xml
        for item in rule_data:
            ne_data_res = self.render('discover/pool_ipv4_discover.j2', {'item': item})

            # 3.输出，将构造好的GND配置数据返回
            # 构造输出
            data = result.serviceConfig.add()
            neid = device_id
            data.servicePath = "/huawei-ac-nes-device:nes/ne/" \
                               + neid + "/address_pool/" + str(item.get('address-name'))
            data.serviceData = ne_data_res
    return result
```

步骤4 开发Jinja2模板

policy_rule.j2

```
<policy_rule xmlns="https://www.hillstonenet.com.cn/firewall_rule">
  <id_number>{{ item.get('id-number') }}</id_number>
  {%- if item.get('action') %}
  <action>{{ item.get('action') }}</action>
  {%- endif %}
  {%- if item.get('src-zone') %}
    <src_zone>{{ item.get('src-zone') }}</src_zone>
  {%- endif %}

  {%- if item.get('dst-zone') %}
    <dst_zone>{{ item.get('dst-zone') }}</dst_zone>
  {%- endif %}
  {%- if item.get('name') %}
    <name>{{ item.get('name') }}</name>
  {%- endif %}

  {%- if item.get('description') %}
    <description>{{ item.get('description') }}</description>
  {%- endif %}

  {%- if item.get('src-ip') %}
  {%- for src_ip_address in item.get('src-ip') %}
    <src_ip>
      <src_ip_address>{{ src_ip_address.get('src-ip-address') }}</src_ip_address>
    </src_ip>
  {%- endfor %}
  {%- endif %}

  {%- if item.get('dst-ip') %}
  {%- for dst_ip_address in item.get('dst-ip') %}
    <dst_ip>
      <dst_ip_address>{{ dst_ip_address.get('dst-ip-address') }}</dst_ip_address>
    </dst_ip>
  {%- endfor %}
  {%- endif %}

  {%- if item.get('src-addr') %}
    {%- for src_address_item in item.get('src-addr') %}
    <src_addr_group>
      <src_addr_name>{{ src_address_item.get('src-addr-name') }}</src_addr_name>
    </src_addr_group>
```


版权所有 © 华为技术有限公司

```
    {%- endfor %}
  {%- endif %}
  {%- if item.get('dst-addr') %}
    {%- for dst_address_item in item.get('dst-addr') %}
    <dst_addr_group>
      <dst_addr_name>{{ dst_address_item.get('dst-addr-name') }}</dst_addr_name>
    </dst_addr_group>
    {%- endfor %}
  {%- endif %}
  {%- if item.get('service') %}
    {%- for service_item in item.get('service') %}
    <service_group>
      <service_name>{{ service_item.get('service-name') }}</service_name>
    </service_group>
    {%- endfor %}
  {%- endif %}
</policy_rule>
```

pool_ipv4.j2

```
<address_pool xmlns="https://www.hillstonenet.com.cn/firewall_address_pool">
  <address_name>{{ item.get('address-name') }}</address_name>
  {%- if item.get('address-name') %}
    <description>{{ item.get('description') }}</description>
  {%- endif %}
  {%- if item.get('member') %}
    <member>{{ item.get('member') }}</member>
  {%- endif %}
  {%- if item.get('ip') %}
  {%- for ip_item in item.get('ip') %}
    <ip>
      <ip_address>{{ ip_item.get('addr') }}</ip_address>
    </ip>
  {%- endfor %}
  {%- endif %}

  {%- if item.get('range') %}
  {%- for range_item in item.get('range') %}
    <range>
      <address_from>{{ range_item.get('address-start') }}</address_from>
      <address_to>{{ range_item.get('address-end') }}</address_to>
    </range>
  {%- endfor %}
  {%- endif %}
</address_pool>
```

----结束

 版权所有 © 华为技术有限公司

7 开发 SSP

SSP业务插件包可以实现三种功能：EasyMap、Discover、RPC。

- EasyMap：将网络层业务分解成网元层业务，实现设备纳管后由网络层向设备直接下发业务。
- Discover：业务还原，将网元层业务重新组织成网络层业务，即EasyMap的逆过程。设备纳管前已存在网元层业务，那么纳管后可以将网元层业务还原到网络层。
- RPC：用户自定义功能。标准的NETCONF/YANG配置类模型不能满足要求，用户可以灵活自定义功能，如一些查询类操作。

7.1 开发网络业务插件包

7.2 开发RPC业务插件包

7.3 开发SSP示例

7.1 开发网络业务插件包

开放可编程开发者社区中已经上传了一些典型场景的软件包，用户可以登录社区查询并下载适合的代码样例包。通过对代码样例包进行定制开发，比较高效，开发难度也比较小。如果没有找到适合的代码样例包，用户也可以从开放可编程环境中下载默认SSP软件包进行开发。

用户在开发SSP软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

7.1.1 创建业务包模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据**表7-1**填写软件包的相关属性值。

 版权所有 © 华为技术有限公司

表 7-1 L3VPN 软件包属性

属性名称	属性值
名称	L3VPN
版本	1.0.0
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	l3vpn
业务定义点	l3vpn
映射类型	python+template

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

7.1.2 开发业务包

7.1.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

表 7-2 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：Specific Service plugin、Generic NE driver、Specific NE driver、SND Migrate plugin。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，长度不能超过64个字符。 version（必选）字段：版本号三段式规则。	可选
service-name	ssp包特有值。服务名称。	自定义，不能超过 64个字符。 ssp包时，必须设置，不同名的包不能使用相同的service-name。 其他类型软件包时，不需要设置。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。EasyMap，type设置为mapping。RPC，type设置为service-rpc。Discover，type设置为discover。网络业务联动，type设置为post-modify-action。告警包功能设置为service-alarm。服务注入功能设置为serviceprovider。如需要支持填充在YANG模型中定义参数默认值，type设置为service_settings。data-with-default可以设置为all或explicit，all表示填充定义的参数默认值，explicit表示默认不填充定义的参数默认值。 ● key（必选），回调key值，不能超过128个字符。一般情况下，YANG模型中的“app:application-definition”的值对应key值，即业务定义点。type设置为service-rpc时，key值规则为“（namespace:revision)localname”，localname指YANG模型rpc节点名称，例如"(http://example.com/RPC?revision=2018-12-09)read_device_vrf"。type设置为serviceprovider时，key值自定义。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“type”、"key"、“python-class-name”，参数含义和修改方法请参见上述表格。最终编辑文件示例如下：

```
{
    "name": "L3VPN",
    "version": "1.0.0",
    "description": "l3vpn",
    "package-type": "ssp",
```

 版权所有 © 华为技术有限公司

```
  "producer": "huawei",
  "service-name":"l3vpn",
  "nce-min-versions":[
      "1.0.0"
  ],

  "hooks": [
    {
      "type": "service_settings",
      "key": "l3vpn",
      "data-with-default": "all"
    },
    {
      "type": "mapping",
      "key": "l3vpn",
      "python-class-name": "l3vpn.l3vpn.L3VPNService"
    },
  {
    "type": "mapping",
    "key": "neTunnels",
    "python-class-name": "com.huawei.cmnet.srte.aoc_ncs_srte.AocNcsSrteService"
  },
  {
    "type": "discover",
    "key": "neTunnels",
    "python-class-name": "com.huawei.cmnet.srte.srte_restore.SrteRestoreService"
  }
  ]
}
```

----结束

7.1.2.2 开发业务 YANG 模型

在集成开发环境左侧导航树中，双击打开yang目录下的.yang文件，编写YANG模型。YANG模型中的“app:application-definition”的值一般对应pkg.json中的hooks下的key，即业务定义点。

当前系统仅支持在application的根节点下挂接list和存在类型container的业务定义点。在父节点上定义业务定义点，则在子节点上不允许再定义业务定义点。

当用户希望对list进行自定义排序，如安全规则list、静态路由list，需要在YANG模型中对应的list打上ordered-by user标记，并在代码中进行处理。

开发YANG模型时，分析梳理出配置下发涉及的参数，如表7-3所示，并在YANG模型中进行定义。

⚠ 注意

YANG文件命名时需要注意：AOC系统中加载的GND与SSP中，不能存在名字相同的YANG文件。

表 7-3 整理的输入参数

类别	描述	参数名称	参数值
VPN业务标识	标识哪个VPN业务下发到哪台设备，提取设备名称和VPN名称。	deviceName	PE1
		vrfName	5G-RAN

版权所有 © 华为技术有限公司

类别	描述	参数名称	参数值
VPN配置参数	VPN需要配置RD和RT值，过滤路由信息，提取RD和RT值。	rd	100:01:00
		rt	100:11:00
接口配置参数	绑定VPN的子接口需配置IPv4地址，提取子接口名称、子接口描述、子接口IP和子接口掩码。	ifName	GigabitEthernet0/5/0.1
		description	connect to pe2
		ip	10.0.2.9
		Subnet mask	255.255.255.0
BGP对等体配置参数	配置BGP对等体的IP地址，提取对等体IP。	peerAddress	10.2.2.2

最终编辑完成的L3VPN软件包YANG模型如下。

```
module l3vpn {
    //模块名称对应服务名称
    namespace "http://example.com/l3vpn";
    prefix "l3vpn";

    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    description
        "The module for L3VPN example.";

    revision 2020-11-05 {
        description "Initial revision.";
    }

    augment "/app:applications"{
        list l3vpn {
            app:application-definition "l3vpn";
            /*app:application-definition 表示定义一个系统可识别的servicepoint，
           这里的service-point必须要和pkg.json文件hooks参数中的key字段值一致。*/

            ordered-by user;
            //可选功能，当用户希望对list进行自定义排序，如安全规则list、静态路由list等场景时，需要在YANG模型中对应的list打上ordered-by user标记。

           key "deviceName vrfName";
            //定义业务标识，本例中使用设备名称和VPN名称作为组合key
            leaf deviceName {
              type string {
                length "1..512";
              }
            }

            leaf vrfName {
              description "Defines a type of service component identifier.";
              type string {
                length 1..255 {
                  description "VPN Routing/Forwarding instance name, support 1-255 characters.";
                }
              }
```

 版权所有 © 华为技术有限公司

```
            default "vrf-test";
            mandatory true;
          }
          //在yang模型中定义VPN业务的RD和RT值
          leaf rd {
            description "BGP route distinguisher";
            type string;
          }

          leaf rt {
            description "Route target extended community as per RFC4360";
            type string;
          }

          //在yang模型中定义子接口名称
          leaf ifName {
            type string {
              length "0..255";
            }
          }
          //在yang模型中定义子接口描述
          leaf description {
            type string {
              length "0..242";
            }
          }
          //在yang模型中定义子接口IP
          leaf ip {
            type inet:ipv4-address;
          }
          //在yang模型中定义子接口掩码
          leaf subnet {
            type inet:ipv4-address;
          }
          //在yang模型中定义对等体IP信息
          leaf peerAddress {
            type inet:ipv4-address;
          }
        }
      }
}
```

如果用户需要使用自定义数据加密处理、高危操作定义、YANG模块黑名单屏蔽、YANG模型免校验、自定义模型权限等功能，请参考下文配置。示例软件包不涉及这些配置。

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

 版权所有 © 华为技术有限公司

表 7-4 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d
-	-	type	节点类型，取值范围： ● leaf ● leaf-list ● typedef –自定义类型 默认是leaf

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
    <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
    <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
    <path>/db-security-test:security-top/security-list/security-list-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
    <rpcs>
        <rpc>huawei-aaa:reactivateUser</rpc>
        <rpc>huawei-aaa:changeMyPassword</rpc>
        <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
    </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

 版权所有 © 华为技术有限公司

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
     <name>huawei-cfg</name>
     <revision>2020-03-18</revision>
  </module>
  <module>
     <name>huawei-dhcp</name>
     <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
   <module>ietf-yang-types</module>
   <module>ietf-inet-types</module>
 </modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
```

 版权所有 © 华为技术有限公司

```
            "method":"exec"
        }
        ]
    }
    ]
}
```

7.1.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开软件包Python目录下的.py文件，编写Python代码，实现功能逻辑。

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

另外，当通过服务注入包提供了通用的SDK能力或开放内部服务接口时，用户可以直接调用服务注入包提供的接口快速实现功能，提高开发效率。

说明

实现异步发现逻辑时，需要注意

- 先处理设备同步逻辑，再处理业务同步逻辑。
- 建议限制分批次发现，每批次发现的业务数量在200以内，否则可能导致功能异常。

EasyMap

EasyMap是将网络层业务分解成网元层业务，hook中type需设置为mapping实现EasyMap功能，需要继承NcsService并重写ncs_map方法，实现业务逻辑。

```
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射。
from aoc.ncs.ncsservice import NcsService


class GNDRule(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入的policy_rule信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['policy_rule']

        # 将neid和policy填充到jinja模板
        res = self.render('policy_rule.j2', {
            'neid': aoccontext.neid,
            'policy': request_dic_node
        })
        self.logger.info('return is %s' % res)
        return res
```

Discover

Discover即业务业务还原，将网元层业务重新组织成网络层业务，即EasyMap的逆过程。hook中type需设置为discover，编写代码时需继承NcsService类并重写discover方法，定义业务还原逻辑。

```
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
# devicemgr提供了一些查询设备信息的接口。
from aoc import NcsService, devicemgr
# IPv4Network提供了IP地址是否公网、转换二进制等接口。
from ipaddress import IPv4Network
from aoc.ncs.filter import ipaddr
```

 版权所有 © 华为技术有限公司

```
class SrteRestoreService (NcsService):
    def discover(self, discover_input, aoc_context):
        """
        设备发现函数
        :param discover_input: 设备发现入参
        :param aoc_context: 上下文
        :return:转换后的xml报文
        """
        time_begin = time.time()
        self.logger.info("[SrTeRestoreService][discover] discoverInput is: %s,"
                         " aoc_context is: %s" % (discover_input, aoc_context))

        # 从discover_input中获取设备信息
        device_info_list = []
        if discover_input.deviceInfo is not None:
            device_info_list = discover_input.deviceInfo
        self.logger.info("[SrTeRestoreService][discover] device_info_list is: %s"
                         % device_info_list)

        # 读取上下文信息 接着上一轮处理的进度继续处理
        discovery_id = discover_input.discoveryId
        self.cache_context = discover_input.cacheContext
        self.logger.info("start real discovery: %s---%s" % (discovery_id, time.time()))

        # 介绍一下DiscoverOutput
        result = DiscoverOutput()
        # 遍历输入信息列表中的设备信息
        for device_info in device_info_list:
            device_id = device_info.deviceId

            self.logger.info("[SrTeRestoreService][discover] device_info.attributes is: %s" %
device_info.attributes)
            # 判断设备中需要处理的业务个数，如果大于0小于等于最大值，则将储存的最大值刷新
            if device_info.attributes and self.handler_max_size >= int(device_info.attributes):
                self.handler_max_size = int(device_info.attributes)

            # 读取设备RDB,查询对应设备ifm的总数
            self.read_ifm_count_from_rdb(aoc_context, device_id)
            # get the template by device info
            template = self.find_ne_template(device_id)

            if template is None:
                raise ValueError("The device doesn't support [deviceId: %s]" % device_id)

            # 读取设备RDB,查询对应设备ifm，每次查询100条
            ifm_dicts = self.read_ifm_from_rdb(aoc_context, device_id)
            # build ne template
            if ifm_dicts and ifm_dicts['interface']:
                # 处理SrTe隧道模板内容，并返回模板内容集合
                ifm_list = self.handler_template_content(ifm_dicts, aoc_context, device_id)
                self.logger.info("[SrTeRestoreService][discover] ifm_list is %s" % ifm_list)

                # 组建需要还原的数据
                sr_tes_restore_template_content = {
                    "ifm_list": ifm_list,
                    "ne_id": device_id
                }

                # 根据对应网元的模板对数据进行反向映射完成业务还原，template为业务还原的jinja
                tunnels = DictNode(xmltodict.parse)
                    self.render(template, sr_tes_restore_template_content),
                    encoding="utf-8", strip_whitespace=True))
                self.logger.info("[SrTeRestoreService][discover] tunnels is %s" % tunnels)
                if tunnels['items']:
                    # 处理返回给AOC的响应response
                    self.handler_response(tunnels, device_id, result)

            # 更新上下文信息，为下一轮业务还原做标记。若记录个数超过上限或数据处理完则结束整个流程。
            if int(self.cache_context["offset"]) < int(self.cache_context["ifm_count"]):
```

 版权所有 © 华为技术有限公司

```
            result.goOn = True
        for key in self.cache_context:
            self.logger.debug("interface %s" % self.cache_context[key])
            result.cacheContext[key] = self.cache_context[key]

    self.logger.info("[SrTeRestoreService][discover] result %s" % result)
    time_end = time.time()
    self.logger.info("[SrTeRestoreService][discover] cost time is %s seconds" % (time_end - time_begin))
    return result


    # 代码已经自适应全量业务还原与增量业务还原，无需其他编码
    def handler_template_content(self, ifm_dicts, aoc_context, device_id):
    """
    处理SrTe隧道模板内容，并返回模板内容集合
    :param ifm_dicts: ifm_dicts.
    :param cache_context: discover上下文.
    :param aoc_context: 上下文.
    :param device_id: 设备id.
    :return: ifm_list
    """
    ifm_list = []
    # 查询network-instance
    ipv4s_and_ipv6s = self.read_ipv4_and_ipv6_from_data_store(device_id, aoc_context)
    ipv4s = ipv4s_and_ipv6s['ipv4s']
    ipv6s = ipv4s_and_ipv6s['ipv6s']
    # 通过device_id查询AOC页面中已经存在的隧道
    ifm_name_dict = self.valid_ifm_exist(aoc_context, device_id)
    for ifm in ifm_dicts['interface']:
        self.logger.info("[SrTeRestoreService][handler_template_content]:1")
        # signal-protocol为segment-routing时是SR-TE隧道
        if self.is_segment_routing(ifm):
            self.logger.info("[SrTeRestoreService][handler_template_content]:is_segment_routing")
            # 如果在AOC页面中已经存在则不需要再进行还原
            if ifm_name_dict and ifm['name'] in ifm_name_dict:
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:in ifm_name_dict")
            # 根据cacheContext判断接口是否已经还原
            if self.cache_context is not None and self.cache_context[ifm['name']] is not None and \
                    self.cache_context[ifm.name] == 'exist':
                self.cache_context[ifm['name']] = 'exist'
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:not exist")
            # 解析RDB结果
            if ifm['name'] in ipv4s:
                ifm['ipv4'] = ipv4s[ifm['name']]
            if ifm['name'] in ipv6s:
                ifm['ipv6'] = ipv6s[ifm['name']]

            # 更新上下文中ifm信息 记录其已经被还原
            self.cache_context[ifm['name']] = 'exist'
            self.logger.info("[SrTeRestoreService][handler_template_content] ifm:%s" % ifm['name'])
            sr_te_restore_template_content = {
                "interface": ifm
            }
            self.init_size = self.init_size + 1
            # 查看是否超出单次处理的最大次数
            if self.init_size > self.handler_max_size:
                break
            ifm_list.append(sr_te_restore_template_content)
    return ifm_list
```

ordered-by user

当用户希望对list进行自定义排序，如安全规则list、静态路由list，需要在YANG模型中对应的list打上ordered-by user标记，并在代码中进行处理。

 版权所有 © 华为技术有限公司

ordered-by user场景，如果是根节点下的list，需要用户编写排序逻辑，如果非根节点下的list，系统自动处理排序逻辑，用户无需编码。

```
    # 自定义排序逻辑时，建议before&last/first、after&last/first组合使用。注意不要before和after混用，否则可能造成排序逻辑矛盾，导致顺序混乱。
    def build_orderedby_user_key(self, aoccontext, template_context):
        ordered_paths = aoccontext.orderedPaths
        path = aoccontext.serviceInst
        # get position of current path instance
        index = ordered_paths.index(path, 0, len(ordered_paths))
        if index is not None:
            if index == (len(ordered_paths) - 1):
                template_context['last'] = True
            else:
                next_path = ordered_paths[index + 1]
                next_ins = next_path.split('/')[len(split_path) - 1]
                template_context['next_ins'] = next_ins
        return template_context
```

网络业务联动

在业务层，系统可以自动识别业务间的依赖关系，当业务A变更时，与业务A有依赖关系的业务B会同时变化，从而实现业务变更稳定不出错。例如：已知eva业务依赖evs业务。在evs业务中修改demosync实例的mac-limit-value参数值，eva业务中mac-limit-value参数值会被同步修改。

```
import re
import os
import logging
import json

from aoc import NcsService
from aoc import devicemgr
from aoc.sys import service

# 导入网络联动相关接口
from aoc.ncs.ncs_model_pb2.post_modify_pb2 import PostModifyInput, PostModifyResult, \
    TransactionAttachItems, TransactionAttachItem


class AocNcsEvsServicePoint(NcsService):
    """
    The module where the eas interface is located.
    Generate configuration messages include:
        Eth-Trunk interface, physical port, etc.
    """
    def __init__(self, logger=None, resource_dir=''):
        self.template_adapter = None
        devicemgr.logger = logger
        if logger is None:
            devicemgr.logger = logging
        NcsService.__init__(self, logger, resource_dir)

    def post_modify(self, aoc_context, post_modify_request):

        # 当业务A变更时，与业务A有依赖关系的业务B同步修改实现
        for data in post_modify_request.change_data:
            path = data.path
            instance_name = path.split('/')[-1]
            attached_path = "/huawei-ac-applications:applications/huawei-ncs-eva:eva/" \
                            + instance_name
            if data.operation_type == 1:
                if "async" in instance_name:
                    service.redeploy_async(attached_path)
                else:
                    service.redeploy(attached_path)
            else:
```

 版权所有 © 华为技术有限公司

```
            if "async" in instance_name:
                service.undeploy_async(attached_path)
            else:
                service.undeploy(attached_path)
```

服务注入

服务注入指将系统中没有内置的接口或能力通过服务注入SSP注入系统中。hook中type需设置为serviceprovider，表示是服务提供者。

服务注入后，服务使用者用户可以调用服务注入接口。

- 通过self.serviceinject.action(self.service_name, input, aoc_context)调用服务提供者的action方法，service_name是服务提供者的钩子键（即hook中的key值），input需要根据服务提供者构造。
- 通过self.serviceinject.run(self.service_name, key, input, aoc_context)调用服务提供者的run方法，key是用户定义的关键字，用于区分其他调用。
- 通过ncs_context.setdefault('key', 'value')设置上下文，该上下文将在下次执行时打开。可编程框架将通过request.context引入最后提交的上下文。

```
# 服务提供者
import ipaddress
import random
from aoc.sys.inject.serviceprovider import ServiceProvider


# 定义实现类IpPool，作为服务提供者，实现run方法、action方法、revert方法
class IpPool(ServiceProvider):
    ip_pool = set()
    def run(self, input):
        self.log.info('run %s' % input)
        output = {}
        if input.get("operation") == "apply":
            begin = ipaddress.ip_address('192.168.1.1')._ip
            end = ipaddress.ip_address('192.168.1.100')._ip
            value = random.randint(begin, end)
            output.setdefault("result", value)
            self.ip_pool.add(str(value))
            self.log.info('apply %s' % value)
        self.log.info('ip_pool %s' % self.ip_pool)
        return output


    def revert(self, input):
        self.log.info('revert %s' % input)
        value = input.get("result")
        self.ip_pool.discard(value)
        output = {}
        output.setdefault("result", "success")
        self.log.info('ip_pool %s' % self.ip_pool)
        return output

    def action(self, input):
        self.log.info('action %s' % input)
        if input.get("operation")  == "create":
            request = input.get("request")
            self.log.info('%s' % request.get("url"))
            self.log.info('%s' % request.get("begin"))
            self.log.info('%s' % request.get("end"))
        output = {}
        output.setdefault("result", "success")
        self.log.info('output %s' % output)
        return output
```

```
# 服务使用者
import ipaddress
```

 版权所有 © 华为技术有限公司

```
from aoc.ncs.ncsservice import NcsService
import traceback

class ServiceInjectConsumer(NcsService):
    service_name = 'ippool'
    def gen_request(self):
        begin = ipaddress.ip_address('192.168.1.1')._ip
        end = ipaddress.ip_address('192.168.1.100')._ip
        url = "http://aoc.serviceinject.test"
        request = {}
        request.setdefault("url", url)
        request.setdefault("begin", str(begin))
        request.setdefault("end", str(end))
        return request

    def apply_pool(self):
        input = {}
        input.setdefault("operation", "create")
        input.setdefault("request", self.gen_request())
        self.serviceinject.action(self.service_name, input)

    def apply_ip(self, key, aoccontext):
        input = {}
        input.setdefault("operation", "apply")
        input.setdefault("request", self.gen_request())
        result = self.serviceinject.run(self.service_name, key, input, aoccontext)
        ip_addr = result.get("result")
        return ip_addr

    def ncs_map(self, request, aoccontext=None, template=None):
        self.logger.info('%s' % request)
        try:
            ncs_context = dict(request.context)
            self.logger.info('%s' % ncs_context)

            if "ip_pool" not in ncs_context.keys():
                self.apply_pool();
                ncs_context.setdefault('ip_pool', 'created')

            if "vlan1" not in ncs_context.keys():
                result = self.apply_ip("vlan1", aoccontext);
                ncs_context.setdefault('vlan1', result)

        except Exception as e:
            self.logger.info("%s" % traceback.format_exc())

        return "<inventory-cfg xmlns=\"urn:huawei:yang:huawei-ac-nes\"><nes></nes></inventory-cfg>",
ncs_context
```

7.1.2.4 开发 Jinja2 模板

开发Jinja2模板的思路主要有两个：

- 逆向推导：适用于已纳管真实设备的场景，有下面两种常用的方法。
  - 配置设备试运行获取报文法：通过网元管理试运行获取对应操作NETCONF报文，再将报文复制粘贴到Jinja2模板中，最后把输入参数值替换为参数名。
  - 差异发现导出数据模板法：在纳管设备后同步设备配置，然后在设备上配置所需命令行，再通过网元配置的差异发现能力，导出下发命令行对应的XML数据模板。再在设备配置界面导出需要的YANG模板，对比XML数据模板，在YANG模板中只保留要下发的部分，其他可以删掉，最后把输入参数值替换为参数名。
- 正向推导：适用于没有真实设备的场景，根据设备命令行和YANG模型直接推导编写Jinja2模板。

版权所有 © 华为技术有限公司

为了方便编码，除了支持Jinja本身的过滤器之外，系统还新增了一些可能用到的过滤器及方法。具体请参考**13.3 Jinja2模板过滤器及方法**。

下面主要介绍逆向推导两种方法的具体操作过程。

差异发现导出数据模板法

具体步骤如下：

步骤1 在主菜单中选择“配置一致性 > 设备配置一致性”。进入“设备配置一致性”界面，选择对应的设备，在“操作”列单击“同步”。先保证设备与控制器数据一致。

步骤2 在设备上配置命令后，在“设备配置一致性”界面，选择对应的设备，在“操作”列单击“更多 > 差异发现”。

步骤3 差异发现结束后，“同步状态”显示为“差异发现完毕”，单击“查看差异”图标，打开差异呈现页面，选择“导出模板”。

步骤4 在“设备配置”界面，单击对应设备操作列中的“编辑>特定设备配置”。

步骤5 在工具栏单击导出按钮，弹出导出页面。

步骤6 在导出页面中，勾选“huawei-ifm、Huawei-network-instance和huawei-bgp”三个模型节点，勾选“合并模板”，勾选“带结构不带数据”导出方式，然后单击“确定”。

步骤7 打开导出的模板，对比差异发现导出的数据模板，删除多余部分，仅保留下发部分，然后把输入参数替换为参数名称变量。最后把编辑的内容放入到模板文件“template>l3vpn>NE8000M8.j2”中。

```
<bgp xmlns:x="urn:huawei:yang:huawei-bgp"  x:tag="no_delete">
    <global>
        <yang-enable>true</yang-enable>
    </global>
</bgp>
<network-instance xmlns="urn:huawei:yang:huawei-network-instance">
    <instances>
        <instance>
            <name>{{l3vpn.vrfName}}</name>
            {%- if l3vpn.vrfDes %}
            <description>{{vrfDes}}</description>
            {%- endif %}
            <afs xmlns="urn:huawei:yang:huawei-l3vpn">
                <af>
                    <type>ipv4-unicast</type>
                    <route-distinguisher>{{ l3vpn.rd }}</route-distinguisher>
                    <tunnel-policy>LDP</tunnel-policy>
                    <vpn-targets>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>export-extcommunity</type>
                        </vpn-target>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>import-extcommunity</type>
                        </vpn-target>
                    </vpn-targets>
                </af>
            </afs>
            <bgp xmlns="urn:huawei:yang:huawei-bgp">
                <base-process>
                    <afs>
                        <af>
                            <type>ipv4uni</type>
                            <ipv4-unicast>
```

 版权所有 © 华为技术有限公司

```
                        <import-routes>
                            <import-route>
                                <protocol>direct</protocol>
                                <process-id>0</process-id>
                            </import-route>
                            <import-route>
                                <protocol>static</protocol>
                                <process-id>0</process-id>
                            </import-route>
                        </import-routes>

                    </ipv4-unicast>
                </af>
            </afs>
            <peers>
                <peer>
                    <address>{{ l3vpn.peerAddress }}</address>
                    <remote-as>100</remote-as>
                </peer>
            </peers>
        </base-process>
    </bgp>
  </instance>
 </instances>
</network-instance>
<ifm xmlns="urn:huawei:yang:huawei-ifm">
 <interfaces>
  <interface>
   <name>{{ l3vpn.ifName }}</name>
   <type>GigabitEthernet</type>
   <description>{{ l3vpn.description }}</description>
   <vrf-name>{{ l3vpn.vrfName }}</vrf-name>
   <ipv4 xmlns="urn:huawei:yang:huawei-ip">
    <addresses>
     <address>
      <ip>{{ l3vpn.ip }}</ip>
      <type>main</type>
      <mask>{{ l3vpn.subnet }}</mask>
     </address>
    </addresses>
   </ipv4>
  </interface>
 </interfaces>
</ifm>
```

步骤8 将业务扩展映射到不同款型的设备，编写调用对应款型设备Jinja2模板的逻辑，将编辑好的内容放入到模板文件“template>l3vpn>servicepoint.j2”中。

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
 <nes>
  <ne>
   <neid>{{l3vpn.deviceName| to_ne_id}}</neid>
   {%- if l3vpn.deviceName | is_snd('NE8000M8SPC300_SND')  %}
    {% include 'l3vpn/NE8000M8.j2' %}
   {% elif l3vpn.deviceName | is_snd('NE8000M8_SND') %}
    {% include 'l3vpn/NE8000M8.j2' %}
   {% endif %}
  </ne>
 </nes>
</inventory-cfg>
```

步骤9 （可选）ordered-by user场景，用户需要在Jinja2模板中编写排序处理逻辑。示例如下。

```
<l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
    <l3vpncomm>
     <l3vpnInstances>
      {% if last %}
       <l3vpnInstance xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="last">
      {% elif next_ins %}
```

 版权所有 © 华为技术有限公司

```
            <l3vpnInstance xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="before"
ns0:key="[vrfName='{{ next_ins }}']">
          {% else %}
          <l3vpnInstance>
          {% endif %}
            <vrfName>{{l3vpn.vrfName}}</vrfName>
            <vrfDescription>{{l3vpn.vrfDescription}}</vrfDescription>
            ...
          </l3vpnInstance>
          {% endif %}
        </l3vpnInstances>
      </l3vpncomm>
</l3vpn>
```

----结束

配置设备试运行获取报文法

具体步骤如下：

步骤1 在主菜单中选择“设备配置 > 设备配置”。进入“设备管理”界面，在需要配置的设备上单击“编辑”。

步骤2 在当前设备的“设备管理”左树中单击“huawei-ifm”，打开接口相关的全局操作配置页面。在“Interface”区域单击添加，然后输入接口名称并单击“创建”。

步骤3 在当前设备的“设备管理”左树中单击“huawei-l3vpn”，打开vpn相关的操作配置页面。在“l3vpnInstance”区域单击增加，然后输入vrfName名称并单击“创建”。

步骤4 单击“全局操作”区域上方的“试运行”，查看并复制待下发设备的NETCONF报文。

步骤5 粘贴复制好的NETCONF报文到软件包目录下的“template>l3vpn>NE8000M8.j2”文件中。

步骤6 根据业务需要，修改Jinja模板。如果参数来自于北向输入，则将参数改为变量；如果参数采取固定值，则直接写为常量即可。

步骤7 再将业务扩展映射到不同款型的设备，编写调用对应款型设备Jinja2模板的逻辑，将编辑好的内容放入到模板文件“template>l3vpn>servicepoint.j2”中。最终成型的Jinja模板示例如差异发现导出数据模板法所示。

SSP分解的配置涉及到关键的配置，例如主接口相关。如果在业务实例删除后，删除掉关键配置会有业务安全问题。用户可以基于业务逻辑打上“no_delete”或“no_delete_nested”属性，标记该数据是不会随着业务实例的删除而删除掉。

- no_delete：表示打上标记的本层节点只会被删除数据源，不会被删除数据，子节点是可以被删除数据的。
- no_delete_nested：表示打上标记的本层节点以及子孙节点都只会被删除数据源，不会被删除数据。

  示例如下：

```
<ifm  xmlns:ns0="urn:huawei:yang:huawei-ifm" ns0:tag="no_delete_nested">
    <interfaces>
        <interface>
            <type>GigabitEthernet</type>
            <ipv4 xmlns="urn:huawei:yang:huawei-ip">
                <addresses>
                    <address>
                        <type>main</type>
                    </address>
                </addresses>
            </ipv4>
        </interface>
```

 版权所有 © 华为技术有限公司

```
    </interfaces>
</ifm>
<ntpAuthKeyCfg xmlns:x="urn:huawei:ac" x:tag="no_delete">
```

⚠ 注意

随业务下发的全局配置建议配置“no_delete”标签。否则，在删除业务时会同时删除全局配置，可能导致业务中断。

----结束

7.1.3 验证业务包

7.1.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

7.1.3.2 执行单元测试

步骤1 使用“yang-offline-util.zip”工具，根据YANG模型生成空的NETCONF报文。请执行如下命令。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

当运行结果为空时，表明“subtree.xml”文件成功生成。

步骤2 打开“subtree.xml”文件，并在<application>下各标签中填上适当的值。

```
<l3vpn xmlns="http://example.com/L3VPN">
  <deviceName>NE8000</deviceName>
  <vrfName>5G_test</vrfName>
  <rd>100:1</rd>
  <rt>100:11</rt>
  <ifName>GigabitEthernet0/5/0.1</ifName>
  <description>wireless base</description>
  <ip>10.0.2.9</ip>
  <subnet>255.255.255.0</subnet>
  <peerAddress>10.2.2.2</peerAddress>
</l3vpn>
```


版权所有 © 华为技术有限公司

步骤3 双击打开test目录下的.py文件，将前面步骤修改好的<application>标签内的代码拷贝粘贴到如下<l3vpn>标签位置。示例如下。

```
import unittest
import sys
from aoc.sys import devicemgr, datastore
from mock import Mock
 sys.path.insert(0, "../../python")
from l3vpn.l3vpn import L3VPNService

class Test(unittest.TestCase):
    xml = '''

 # 下面l3vpn标签的代码根据实际情况替换
 <l3vpn xmlns="http://example.com/L3VPN">
    <deviceName>NE8000</deviceName>
    <vrfName>4G_test</vrfName>
    <rd>100:1</rd>
    <rt>100:11</rt>
    <ifName>GigabitEthernet0/5/0.1</ifName>
    <description>wireless base</description>
    <ip>10.0.2.9</ip>
    <subnet>255.255.255.0</subnet>
    <peerAddress>10.2.2.2</peerAddress>
 </l3vpn>
  '''


   def mock_get_id(self, neName):
       if neName == 'CX600':
           return 'mock_neid:CX600'
       if neName == 'NE8000':
           return 'mock_neid:NE8000'
       return 'mock_neid:NE8000'


    def mock_sys_func(self):
       devicemgr.query_neid = self.mock_get_id
       mock_read_datastore_rdb = Mock(return_value='GigabitEthernet0/5/0')
       datastore.read_datastore_rdb = mock_read_datastore_rdb
       mock_write_datastore = Mock(return_value='success')
       datastore.write_datastore = mock_write_datastore
       devicemgr.is_snd = self.mock_is_snd


    def mock_is_snd(self, nename, sndid):
       if nename == 'mock_neid:NE8000' and sndid == 'NE8000M8_SND':
           return True
       if nename == 'mock_neid:NE8000' and sndid == 'NE8000M8SPC300_SND':
           return True
       return False


    def setUp(self):
       self.mock_sys_func()


    def test_l3vpn(self):
       result = L3VPNService().ncs_map_test(self.xml)
       print(result)
if __name__ == "__main__":
    unittest.main()
```

步骤4 运行该测试代码，可看到生成的报文，检查输出是否正确。

回显提示“Process finished with exit code 0”，表明单元测试成功。否则请仔细检查Jinija模板文件和业务YANG模型文件中各属性是否一一对应。

----结束

 版权所有 © 华为技术有限公司

7.1.4 编译业务包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件移动到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

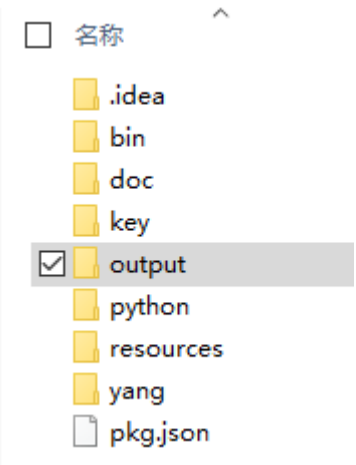
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2021-11-04 18:10:12,291 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Generate signature file success.
2021-11-04 18:10:12,542 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

7.2 开发 RPC 业务插件包

开放可编程开发者社区中已经上传了一些典型场景的软件包，用户可以登录社区查询并下载适合的代码样例包。通过对代码样例包进行定制开发，比较高效，开发难度也比较小。如果没有找到适合的代码样例包，用户也可以从开放可编程环境中下载默认SSP软件包进行开发。

用户在开发SSP软件包时，需要注意关键信息的隐私保护，比如用户密码、登录Token、Session及其他个人数据不要通过日志输出。

版权所有 © 华为技术有限公司

7.2.1 创建业务包模板

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据表7-5填写软件包的相关属性值。

表 7-5 RPC 软件包属性

属性名称	属性值
名称	RPC
版本	1.0.0
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	RPC
业务定义点	RPC
映射类型	python+template

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

7.2.2 开发业务包

7.2.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

版权所有 © 华为技术有限公司

表 7-6 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：Specific Service plugin、Generic NE driver、Specific NE driver、SND Migrate plugin。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，长度不能超过64个字符。 version（必选）字段：版本号三段式规则。	可选
service-name	ssp包特有值。服务名称。	自定义，不能超过 64个字符。 ssp包时，必须设置，不同名的包不能使用相同的service-name。 其他类型软件包时，不需要设置。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。EasyMap，type设置为mapping。RPC，type设置为service-rpc。Discover，type设置为discover。网络业务联动，type设置为post-modify-action。告警包功能设置为service-alarm。服务注入功能设置为serviceprovider。如需要支持填充在YANG模型中定义参数默认值，type设置为service_settings。data-with-default可以设置为all或explicit，all表示填充定义的参数默认值，explicit表示默认不填充定义的参数默认值。 ● key（必选），回调key值，不能超过128个字符。一般情况下，YANG模型中的“app:application-definition”的值对应key值，即业务定义点。type设置为service-rpc时，key值规则为“（namespace:revision)localname”，localname指YANG模型rpc节点名称，例如"(http://example.com/RPC?revision=2018-12-09)read_device_vrf"。type设置为serviceprovider时，key值自定义。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“type”、"key"、“python-class-name”，参数含义和修改方法请参见上述表格。最终编辑文件示例如下：

```
{
   "name": "RPC",
   "version": "1.0.15",
   "description": "This is a RPC service",
   "package-type": "ssp",
```

 版权所有 © 华为技术有限公司

```
  "service-name":"RPC",
  "group": "xxxx",
  "producer": "Huawei",
  "nce-min-versions":[

 ],

  "hooks": [
   {
     "type": "service-rpc",
     "key": "(http://example.com/RPC?revision=2018-12-09)read_device_vrf",
     "python-class-name": "RPCSSP.RpcService.AocNcsAaaService"
   }
  ]
}
```

----结束

7.2.2.2 开发业务 YANG 模型

在集成开发环境左侧导航树中，双击打开yang目录下的.yang文件，编写YANG模型。YANG模型中的“app:application-definition”的值一般对应pkg.json中的hooks下的key，即业务定义点。

当前系统仅支持在application的根节点下挂接list和存在类型container的业务定义点。在父节点上定义业务定义点，则在子节点上不允许再定义业务定义点。

当用户希望对list进行自定义排序，如安全规则list、静态路由list，需要在YANG模型中对应的list打上ordered-by user标记，并在代码中进行处理。

当标准的NETCONF/YANG配置类模型不能满足要求，用户可以自定义RPC功能，定义功能的输入和输出。RPC功能主要用于定义不需要保存数据的一次性操作，如ping命令。

这里定义设备接口状态查询功能，定义输入device-id、interface ，定义输出interface、ifV4State。

```
module RpcService {
   namespace "http://example.com/RPC";
   prefix "RPC";

   import huawei-ac-applications {
       prefix app;
   }
   description
       "The module for RpcService example.";

   revision 2018-12-09 {
       description "Initial revision.";
   }

   augment "/app:applications"{
       list RPC {
           app:application-definition "aaa";
           key "instance_name";
           leaf instance_name {
               type string;
           }
       }
   }
   //定义输入：device-id、interface
   rpc read_device_vrf {
   input {
           leaf device-id {
               mandatory true;
               type string;
```


版权所有 © 华为技术有限公司

```
        }
        container interfaces {
            list interface {
                key name;
                leaf name {
                    type string {
                      length "1..max";
                    }
                 }
            }
        }
     }
     //定义输出：interface、ifV4State
      output {
        container interfaces {
            list interface {
                key name;
                leaf name {
                    type string {
                      length "1..max";
                    }
                }
                leaf ifV4State {
                    type string;
                }
            }
         }
     }
   }
}
```

加密处理

通过编辑yang目录下的security.xml 文件，用户可以自定义YANG模型中节点对应数据是否加密处理。

加密处理时，存入数据库时会进行加密，取出数据库时再进行解密；配置下发时，加密数据字段的差异在页面呈现为******；北向restconf/v1查询加密数据字段时返回密文；数据一致性同步时加密数据字段不会进行同步，对账时加密数据字段不进行下发。

说明：存入数据库时，不支持对list的key字段进行加密处理。并且加密的数据类型仅支持string类型。

表 7-7 security.xml 文件配置说明

节点名称	父节点	属性	说明
security-node	-	无	根节点
path	security-node	-	安全节点的路径 示例：/a:b/c/d
-	-	type	节点类型，取值范围： ● leaf ● leaf-list ● typedef –自定义类型 默认是leaf

版权所有 © 华为技术有限公司

节点类型是leaf，则只定义<path>标签，一般不需要填写属性，每一个<path>对应的数据都会被加密处理。path必须以顶级节点<moduleName>:<nodeName>开头，其中<moduleName>是yang文件的module名称，<nodeName>是此module中的顶级节点的名字。path顶级节点可以连接子节点，中间以”/”隔开。示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<security-node>
    <path>/db-security-test:security-top/security-sec/security-sec-leaf-list</path>
    <path type="typedef">/db-security-test:security-len-typedef-leaf</path>
    <path>/db-security-test:security-top/security-list/security-list-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-cont/augment-cont-security-leaf</path>
    <path>/dbtest:nodes/dbtest:node/db-security-test:augment-security-leaf</path>
</security-node>
```

高危操作

通过编辑yang目录下的risk_operation.xml文件，用户可以自定义YANG模型中RPC操作是否为高危操作。当在界面执行高危操作时，会弹出确认操作的提示框；执行非高危操作时，直接执行。

RPC的操作全部都定义在标签<rpcs>下，每一个RPC高危操作对应一个<rpc>标签，RPC标签内的书写格式为moduleName:rpcName，示例如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<risk-operation>
    <rpcs>
        <rpc>huawei-aaa:reactivateUser</rpc>
        <rpc>huawei-aaa:changeMyPassword</rpc>
        <rpc>huawei-aaa:changeMyIdleTimeout</rpc>
    </rpcs>
 </risk-operation>
```

黑名单功能

通过编辑yang目录下的blacklist.xml文件，用户可以自定义YANG模型中模块是否在界面屏蔽。

需要屏蔽的模块定义在标签<module>下，在标签内定义module name和module revision，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<blacklist>
  <module>
      <name>huawei-cfg</name>
      <revision>2020-03-18</revision>
  </module>
  <module>
      <name>huawei-dhcp</name>
      <revision>2020-04-29</revision>
  </module>
</blacklist>
```

YANG 模型免校验

通过编辑yang目录下的multiRevisionModule.xml文件，用户可以自定义YANG模型中模块在升级时是否参与校验。

需要免校验的模块定义在标签<modules>下，每一个免校验模块对应一个<module>标签，module标签内的书写格式为moduleName，示例如下：

```
<?xml version="1.0" encoding="utf-8"?>
<modules>
    <module>ietf-yang-types</module>
```

 版权所有 © 华为技术有限公司

```
  <module>ietf-inet-types</module>
</modules>
```

自定义模型权限

通过编写权限控制文件，用户可以自定义YANG模型节点的操作权限。软件包安装成功后，这些权限信息会动态注入到系统中。通过用户管理，可以给不同职责的用户授予合理的权限，避免越权操作和非安全操作。默认角色系统会自动绑定软件包模型权限，自定义角色需要用户自助管理分权。

用户通过在resources目录下新增permission.json文件，按照如下代码示例自定义模型节点权限。如果用户未编写permission.json文件，软件包加载时系统自动根据YANG文件生成module粒度的0层权限控制的文件，包含create/delete/read/update/exec权限项。

开放可编程支持container、list节点的操作分权，用户根据需要定义模型节点的操作权限。支持0层、1层、2层深度的权限自定义，0层即module粒度的权限控制，用户未定义时系统默认生成0层权限控制；1层即container或list节点都只是1级，没有子节点；2层即container或list节点存在2级，有子节点。系统进行权限控制时，会根据最长匹配原则进行匹配，即优先匹配层级深的权限。

```
{
   "modules": [
   {
      "module-name":"hbng",
      "operations": [
      {
         "uri-pattern":"containerA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":"listA",
         "method":" create/delete/read/update "
      },
      {
         "uri-pattern":" listA/listC",
         "method":" create/delete/read/update"
      },
      {
         "uri-pattern":"resetStatictic",
         "method":"exec"
      }
      ]
   }
   ]
}
```

7.2.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开软件包Python目录下的.py文件，编写Python代码，实现功能逻辑。

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

另外，当通过服务注入包提供了通用的SDK能力或开放内部服务接口时，用户可以直接调用服务注入包提供的接口快速实现功能，提高开发效率。

当用户自定义RPC功能时，需要继承NcsService并重写ncs_rpc方法，按照业务需求实现自定义功能的逻辑，最终返回处理结果。

 版权所有 © 华为技术有限公司

```
# 必选，导入NcsService类，NcsService是aoc_api提供的供SSP包继承的父类。devicemgr可用于从设备上实时查询数据。
from aoc import NcsService, devicemgr


class AocNcsAaaService(NcsService):
    # 查询srte状态
    INTERFACE_PATH = "/huawei-ifm:ifm"

    # 设置设备查询路径
    PROPERTIES_SUBTREE = """
            <interfaces>
              <interface>
                <ifDynamicInfo>
                  <ifV4State/>
                </ifDynamicInfo>
              </interface>
            </interfaces>
           """

    RESULT_TEMPALTE = """
           <read_device_vrf xmlns="http://example.com/RPC">
              <interfaces>
                 %s
              </interfaces>
           </read_device_vrf>
           """

    RESULT_VRF_ONE_TEMPALTE = """
                  <interface>
                     <name>%s</name>
                     <ifV4State>%s</ifV4State>
                  </interface>
           """

    # 设备RPC
    def ncs_rpc(self, request, context=None, template=None):

        # 获取输入
        request_xml = self.xmltodictnode(request.rpcXML.strip())

        # 获取设备ID
        device_id = request_xml.get("read_device_vrf").get("device-id")
        srte_list = request_xml.get("read_device_vrf").get("interfaces").get("interface")
        if isinstance(srte_list, dict):
            srte_list = [srte_list]

        # 根据设备ID查询设备信息
        output = self.query_device_by_device_id(device_id)

        # 查询返回值
        result = self.result_Construction(output, srte_list)
        return result


    # 构造返回体
    def result_Construction(self, output, srte_list):

        # 获取设备上接口的状态
        all_ifm_status_up = self.filter_ifm_status(output)

        # 构造返回体，返回设置的接口状态
        srte_all_status = ""
        for stre in srte_list:
            name = stre.get("name")
            status = 0
            if name in all_ifm_status_up:
                status = all_ifm_status_up[name]
            srte_all_status = srte_all_status + (self.RESULT_VRF_ONE_TEMPALTE % (name, status))
```

 版权所有 © 华为技术有限公司

```
    self.logger.info("[srte-status] srtes status is %s", srte_all_status)
    return (self.RESULT_TEMPALTE % srte_all_status).strip()


# 设置接口状态
def filter_ifm_status(self, output):
    all_ifm_status_up = {}

    # 获取接口的状态
    if output.result is not None:
        ifm_dict = self.xmltodictnode(output.result.strip())
        if ifm_dict.get("ifm") and ifm_dict.get("ifm").get("interfaces"):
            interface_dict = ifm_dict.get("ifm").get("interfaces").get("interface")
            for interface in interface_dict:
                ifm_name = interface.get("ifName")
                ifV4State = interface.get("ifDynamicInfo").get("ifV4State")
                all_ifm_status_up[ifm_name] = ifV4State
    return all_ifm_status_up


# 根据设备ID查询设备信息
def query_device_by_device_id(self, device_id):

    # 查询设备信息，并从设备中获取报文
    properties = {
        "subtree": self.PROPERTIES_SUBTREE.strip()
    }
    output = devicemgr.query_data_from_device(device_id, 'OPERATIONAL',
                                              self.INTERFACE_PATH, properties, 60000)
    return output
```

说明

如果需要查看自定义的RPC操作返回的具体结果，建议通过北向接口调用。

7.2.2.4 开发 Jinja2 模板

因为python代码中没有调用render渲染，即没有将北向报文传到jinja2模板做转换，所以就不需要开发Jinja2模板了。

7.2.3 验证业务包

7.2.3.1 验证 YANG 文件

在完成软件包的开发后，需要校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

步骤2 复制软件包“yang”目录下的YANG模型文件到“yang-offlineutil.zip”的解压路径下。

步骤3 运行如下命令验证YANG文件的正确性。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar validate console path .

当运行结果为空时，表明YANG文件格式正确。

```
C:\Users\demo>cd yang-offline-util
C:\Users\demo\yang-offline-util>Java -jar yang-offline-util.jar validate console path .
C:\Users\demo\yang-offline-util>
```

----结束

版权所有 © 华为技术有限公司

7.2.3.2 执行单元测试

步骤1 使用“yang-offline-util.zip”工具，根据YANG模型生成空的NETCONF报文。请执行如下命令。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

当运行结果为空时，表明“subtree.xml”文件成功生成。

步骤2 打开“subtree.xml”文件，并在<application>标签下各标签中填上适当的值。

```
<RPC xmlns="http://example.com/RPC">
    <instance_name>NE1</instance_name>
</RPC>
```

步骤3 双击打开test目录下的.py文件，将前面步骤修改好的<application>标签内的代码拷贝粘贴到如下<hvpnService>标签位置。示例如下。

```
import unittest
import sys
sys.path.insert(0, "../../Python")
from HVPNService.HVPNService import AocNcs_servicepoint

class Test(unittest.TestCase):
    xml = '''

    # 下面hvpnService标签的代码根据实际情况替换
    <RPC xmlns="http://example.com/RPC">
        <instance_name>NE1</instance_name>
    </RPC>
    '''


    def test_case1(self):
        result = AocNcs_servicepoint().ncs_map_test(self.xml)
        print(result)

if __name__ == "__main__":
    unittest.main()
```

步骤4 运行该测试代码，可看到生成的报文，检查输出是否正确。

回显提示“Process finished with exit code 0”，表明单元测试成功。否则请仔细检查Jinija模板文件和业务YANG模型文件中各属性是否一一对应。

----结束

7.2.4 编译业务包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件移动到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

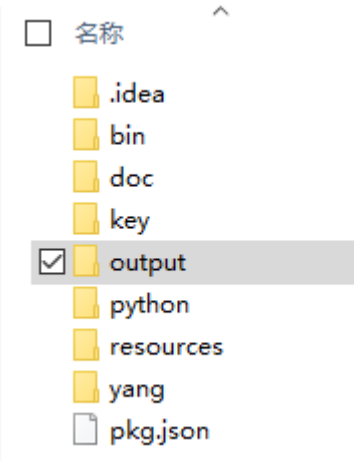
在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

 版权所有 © 华为技术有限公司

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2021-11-04 18:10:12,291 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Generate signature file success.
2021-11-04 18:10:12,542 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

7.3 开发 SSP 示例

7.3.1 开发 L3VPN SSP 示例

本示例介绍如何通过下载的默认SSP包，开发一个L3VPN SSP包，用户自定义一个L3VPN网络业务(下发到PE设备NE8000M8)，并在北向开放此网络业务。

步骤1 创建SSP模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表7-8填写软件包的相关属性值。

表 7-8 L3VPN 软件包属性

属性名称	属性值
名称	L3VPN
版本	1.0.0
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	l3vpn

版权所有 © 华为技术有限公司

属性名称	属性值
业务定义点	l3vpn
映射类型	python+template

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
    "name": "L3VPN",
    "version": "1.0.0",
    "description": "l3vpn",
    "package-type": "ssp",
    "producer": "huawei",
    "service-name":"l3vpn",
    "nce-min-versions":[
        "1.0.0"
    ],

    "hooks": [
     {
       "type": "service_settings",
       "key": "l3vpn",
       "data-with-default": "all"
     },
     {
       "type": "mapping",
       "key": "l3vpn",
       "python-class-name": "l3vpn.l3vpn.L3VPNService"
     }
    ]
}
```

步骤3 开发业务YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表7-9所示，并在YANG模型中进行定义。

表 7-9 整理的输入参数

类别	描述	参数名称	参数值
VPN业务标识	标识哪个VPN业务下发到哪台设备，提取设备名称和VPN名称。	deviceName	PE1
		vrfName	5G-RAN
VPN配置参数	VPN需要配置RD和RT值，过滤路由信息，提取RD和RT值。	rd	100:01:00
		rt	100:11:00
接口配置参数	绑定VPN的子接口需配置IPv4地址，提取子接口名称、子接口描述、子接口IP和子接口掩码。	ifName	GigabitEthernet0/5/0.1
		description	connect to pe2
		ip	10.1.2.9

 版权所有 © 华为技术有限公司

类别	描述	参数名称	参数值
		Subnet mask	255.255.255.0
BGP对等体配置参数	配置BGP对等体的IP地址，提取对等体IP。	peerAddress	10.2.2.2

最终编辑完成的L3VPN软件包YANG模型如下。

```
module l3vpn {
    //模块名称对应服务名称
    namespace "http://example.com/l3vpn";
    prefix "l3vpn";

    import huawei-ac-applications {
        prefix app;
    }
    import ietf-inet-types {
        prefix inet;
    }

    description
        "The module for L3VPN example.";

    revision 2020-11-05 {
        description "Initial revision.";
    }

    augment "/app:applications"{
        list l3vpn {
            app:application-definition "l3vpn";
            /*app:application-definition 表示定义一个系统可识别的servicepoint，
           这里的service-point必须要和pkg.json文件hooks参数中的key字段值一致。*/

            ordered-by user;
            //可选功能，当用户希望对list进行自定义排序，如安全规则list、静态路由list等场景时，需要在YANG模型中对应的list打上ordered-by user标记。

          key "deviceName vrfName";
           //定义业务标识，本例中使用设备名称和VPN名称作为组合key
           leaf deviceName {
             type string {
               length "1..512";
             }
           }

           leaf vrfName {
             description "Defines a type of service component identifier.";
             type string {
               length 1..255 {
                 description "VPN Routing/Forwarding instance name, support 1-255 characters.";
               }
             }
             default "vrf-test";
             mandatory true;
           }
          //在yang模型中定义VPN业务的RD和RT值
         leaf rd {
            description "BGP route distinguisher";
            type string;
          }

          leaf rt {
            description "Route target extended community as per RFC4360";
            type string;
```

 版权所有 © 华为技术有限公司

```
        }

        //在yang模型中定义子接口名称
        leaf ifName {
          type string {
            length "0..255";
          }
        }
        //在yang模型中定义子接口描述
        leaf description {
          type string {
            length "0..242";
          }
        }
        //在yang模型中定义子接口IP
        leaf ip {
          type inet:ipv4-address;
        }
        //在yang模型中定义子接口掩码
        leaf subnet {
          type inet:ipv4-address;
        }
        //在yang模型中定义对等体IP信息
        leaf peerAddress {
          type inet:ipv4-address;
        }
      }
    }
}
```

步骤4 开发代码

实现EasyMap功能，需要继承NcsService并重写ncs_map方法，实现业务逻辑。

L3VPN软件包代码示例如下所示。

```
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射。
from aoc.ncs.ncsservice import NcsService


class GNDRule(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入的policy_rule信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['policy_rule']

        # 将neid和policy填充到jinja模板
        res = self.render('policy_rule.j2', {
            'neid': aoccontext.neid,
            'policy': request_dic_node
        })
        self.logger.info('return is %s' % res)
        return res
```

ordered-by user场景，如果是根节点下的list，需要用户编写排序逻辑，如果非根节点下的list，系统自动处理排序逻辑，用户无需编码。代码示例如下所示。

```
# NcsService是系统提供的供SSP/GND包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射。
from aoc.ncs.ncsservice import NcsService


class AddressPool(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):

        # 将基于北向yang输入address_pool信息由xml报文转化为DictNode
        request_dic_node = request.xmldictnode['address_pool']

        # 将neid和address_pool填充到jinja模板
        res = self.render('pool_ipv4.j2', {"address_pool": request_dic_node,
```

 版权所有 © 华为技术有限公司

```
                                        'neid': aoccontext.neid})
        self.logger.info('return is %s' % res)
        return res
```

步骤5 开发Jinja2模板

把编辑的内容放入到模板文件“template>l3vpn>NE8000M8.j2”中

```
<bgp xmlns:x="urn:huawei:yang:huawei-bgp"  x:tag="no_delete">
    <global>
        <yang-enable>true</yang-enable>
    </global>
</bgp>
<network-instance xmlns="urn:huawei:yang:huawei-network-instance">
    <instances>
        <instance>
            <name>{{l3vpn.vrfName}}</name>
            {%- if l3vpn.vrfDes %}
            <description>{{vrfDes}}</description>
            {%- endif %}
            <afs xmlns="urn:huawei:yang:huawei-l3vpn">
                <af>
                    <type>ipv4-unicast</type>
                    <route-distinguisher>{{ l3vpn.rd }}</route-distinguisher>
                    <tunnel-policy>LDP</tunnel-policy>
                    <vpn-targets>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>export-extcommunity</type>
                        </vpn-target>
                        <vpn-target>
                            <value>{{ l3vpn.rt }}</value>
                            <type>import-extcommunity</type>
                        </vpn-target>
                    </vpn-targets>
                </af>
            </afs>
            <bgp xmlns="urn:huawei:yang:huawei-bgp">
                <base-process>
                    <afs>
                        <af>
                            <type>ipv4uni</type>
                            <ipv4-unicast>
                                <import-routes>
                                    <import-route>
                                        <protocol>direct</protocol>
                                        <process-id>0</process-id>
                                    </import-route>
                                    <import-route>
                                        <protocol>static</protocol>
                                        <process-id>0</process-id>
                                    </import-route>
                                </import-routes>

                            </ipv4-unicast>
                        </af>
                    </afs>
                    <peers>
                        <peer>
                            <address>{{ l3vpn.peerAddress }}</address>
                            <remote-as>100</remote-as>
                        </peer>
                    </peers>
                </base-process>
            </bgp>
        </instance>
    </instances>
</network-instance>
<ifm xmlns="urn:huawei:yang:huawei-ifm">
    <interfaces>
```

 版权所有 © 华为技术有限公司

```
      <interface>
        <name>{{ l3vpn.ifName }}</name>
        <type>GigabitEthernet</type>
        <description>{{ l3vpn.description }}</description>
        <vrf-name>{{ l3vpn.vrfName }}</vrf-name>
        <ipv4 xmlns="urn:huawei:yang:huawei-ip">
          <addresses>
            <address>
              <ip>{{ l3vpn.ip }}</ip>
              <type>main</type>
              <mask>{{ l3vpn.subnet }}</mask>
            </address>
          </addresses>
        </ipv4>
      </interface>
    </interfaces>
</ifm>
```

将业务扩展映射到不同款型的设备，编写调用对应款型设备Jinja2模板的逻辑，将编辑好的内容放入到模板文件“template>l3vpn>servicepoint.j2”中。

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
    <nes>
      <ne>
        <neid>{{l3vpn.deviceName| to_ne_id}}</neid>
        {%- if l3vpn.deviceName | is_snd('NE8000M8SPC300_SND')  %}
          {% include 'l3vpn/NE8000M8.j2' %}
        {% elif l3vpn.deviceName | is_snd('NE8000M8_SND') %}
          {% include 'l3vpn/NE8000M8.j2' %}
       {% endif %}
      </ne>
    </nes>
</inventory-cfg>
```

ordered-by user场景，用户需要在Jinja2模板中编写排序处理逻辑。

```
<l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
       <l3vpncomm>
         <l3vpnInstances>
          {% if last %}
            <l3vpnInstance xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="last">
          {% elif next_ins %}
            <l3vpnInstance xmlns:ns0="urn:ietf:params:xml:ns:yang:1"  ns0:insert="before"
ns0:key="[vrfName='{{ next_ins }}']">
          {% else %}
          <l3vpnInstance>
          {% endif %}
            <vrfName>{{l3vpn.vrfName}}</vrfName>
            <vrfDescription>{{l3vpn.vrfDescription}}</vrfDescription>
            ...
          </l3vpnInstance>
          {% endif %}
         </l3vpnInstances>
       </l3vpncomm>
</l3vpn>
```

----结束

7.3.2 开发 L2EVPN SSP 示例

本示例介绍如何通过下载的默认SSP包，开发一个L2EVPN SSP包，用户自定义一个L2EVPN网络业务(下发到PE设备NE5000E)，并在北向开放此网络业务。

步骤1 创建SSP模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

版权所有 © 华为技术有限公司

2. 并在弹出的对话框中设置软件包的相关属性。
   请根据表7-10填写软件包的相关属性值。

   表 7-10 L2EVPN 软件包属性

属性名称	属性值
名称	SSP-CUII-EVPN
版本	1.0.26
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	cuii_evpn
业务定义点	cuii_evpn
映射类型	python+template

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
  "name": "SSP-CUII-EVPN",
  "version": "1.0.26",
  "description": "evpn for cuii",
  "package-type": "ssp",
  "producer": "huawei",
  "service-name": "cuii_evpn",
  "group": "cuii",
  "nce-min-versions": [
    "1.0.0",
    "2.0.0"
  ],
  "hooks": [
    {
      "type": "mapping",
      "key": "l2vpnsvc",
      "python-class-name": "com.huawei.awang.evpn.aoc_ncs_evpn.AocNcsEvpnService"
    },
    {
      "type": "mapping",
      "key": "l2site",
      "python-class-name": "com.huawei.awang.evpn.aoc_ncs_evpn.AocNcsEvpnService"
    }
  ]
}
```

步骤3 开发业务YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表7-11所示，并在YANG模型中进行定义。

 版权所有 © 华为技术有限公司

表 7-11 整理的业务 YANG 模型定义的参数

类别	参数名称			参数值	参数含义
接口配置	pe-name			10.0.7.7	连接的PE名称
	physical-if			Eth-Trunk/0/0/11	物理接口名称
	encapsulation-type			port	封装类型（枚举型）
	vlan-ids			123	vlan
	s-vlan-ids			11	svlan
	c-vlan-ids			22	cvlan
	description			Test	接口的描述信息
EVPN配置	vpn-node			VPWS	Vpn包括evpn-node,bd-node,evpl-node
	vpn-id			110	定义一个服务器id
	vpws-vpn-node	vpws-id		110	vpws实例id
		xconnect-group		VPWS-p2p	-
		local-service-id		111	本地服务器id
		remote-service-id		112	远程服务器id
		evpn-vpn-node-grp	evpn-name	VPWS-HW-110	Evpn实例名称
			description	VPWS-HW-110_TEST	Evpn 实例描述
			route-distinguisher	100:101	路由区分符
			export-route-targets	100:102	导出RT
			import-route-targets	100:102	导入RT

最终编辑完成的L2EVPN软件包YANG模型如下。

```
module chinaunicom-l2evpn-svc-ext {
  yang-version 1;
  // 模块名称对应服务名称
  namespace "urn:ietf:params:xml:ns:yang:unicom:l2vpn:svc";
  prefix chinaunicom-l2evpn-svc-ext;
  import ietf-l2vpn-svc {
    prefix l2vpn;
```

 版权所有 © 华为技术有限公司

```
 }
 import ietf-inet-types {
   prefix inet;
 }
 import huawei-ac-applications {
   prefix app;
 }
......
// 定义与接口配置相关的输入参数
grouping requested-type-grouping {
    container requested-type-profile {
      leaf pe-name {
        type string{
          length "1..250";
        }
        description "attached PE Name";
      }
      leaf physical-if {
        type string;
        description
          "physical interface name.";
      }
      leaf encapsulation-type {
        type encapsulation-type;
        description
          "Encapsulation type. ";
      }
      leaf vlan-ids {
        type string {
            length "1..30";
        }
        description
            "Vlan, when encapsulation-type is vlan,dot1q,stacking. For example, several input: 10";
      }
      leaf s-vlan-ids {
        type string {
            length "1..30";
        }
        description
            "Svlan, when encapsulation-type is qinq. For example, input: 5";
      }
      leaf c-vlan-ids {
        type string {
            length "1..30";
        }
        description
            "Cvlan, when encapsulation-type is qinq. For example, input: 10";
      }
      leaf description {
        type string {
          length "0..242";
        }
        description
          "Interface description.";
      }

      leaf circuit-id {
        description "circuit description for PE-CE port.";
        type string {
          length "0..32";
        }
      }
    }
 }
  augment "/app:applications/l2vpn:l2vpn-svc/l2vpn:sites/l2vpn:site/l2vpn:site-network-accesses/l2vpn:site-
network-access/l2vpn:bearer/l2vpn:requested-type" {
    uses requested-type-grouping;
 }
```

 版权所有 © 华为技术有限公司

```
// 定义与EVPN业务配置相关的输入参数
 grouping evpn-vpn-node-grp {
   description
     "Vpn evpn node.";
   leaf evpn-name {
     type string {
       length "1..31";
     }
     mandatory true;
     description
       "Evpn instance name.";
   }
   leaf description {
     type string {
       length "0..63";
     }
     mandatory true;
     description
       "Evpn instance description.";
   }
   leaf route-distinguisher {
     type string {
       length "3..255";
     }
     mandatory true;
     description
       "Route distinguisher.";
   }
   leaf-list export-route-targets {
     type string {
       length "3..255";
     }
     description
       "Export route targets.";
   }
   leaf-list import-route-targets {
     type string {
       length "3..255";
     }
     description
       "Import route targets.";
   }
   leaf remote-frr {
     type enable-disable-default-type;
     default "default";
     description
       "Remote frr for vpws evpn instance.";
   }
   leaf local-remote-frr {
     type enable-disable-default-type;
     default "default";
     description
       "Local remote frr. Only support normal evpn instance and vpws evpn instance.";
   }
 }

  augment "/app:applications/l2vpn:l2vpn-svc/l2vpn:sites/l2vpn:site/l2vpn:site-network-accesses/l2vpn:site-
network-access/l2vpn:vpn-attachment/l2vpn:attachment-flavor" {
   // 定义与EVPL配置相关的输入参数
   case vpn-node {
     leaf vpn-node {
       type evpn-layer-type;
       mandatory true;
       description
         "Vpn include evpn-node,bd-node,evpl-node eg.";
     }
     leaf vpn-id {
       type string;
       mandatory true;
```

 版权所有 © 华为技术有限公司

```
      description
        "Defines a service identifier.";
    }
    container vpws-vpn-node {
      description
        "Vpn vpws node.";
      leaf vpws-id {
        type uint32 {
          range "1..16777215";
        }
        mandatory true;
        description
          "Vpws Instance Value id.";
      }
      leaf xconnect-group {
        type string;
      }
      leaf local-service-id {
        type uint32 {
          range "1..16777215";
        }
        mandatory true;
        description
          "Local service id.";
      }
      leaf remote-service-id {
        type uint32 {
          range "1..16777215";
        }
        mandatory true;
        description
          "Remote service id.";
      }
      container evpn-attachment {
        description
          "Vpn attachment evpn node.";
        uses  evpn-vpn-node-grp;
      }
    }
  }
 }
}
```

步骤4 开发代码

实现EasyMap功能，需要继承NcsService并重写ncs_map方法，实现业务逻辑。

L2EVPN软件包代码示例如下所示。

```
# 正则表达式模块re，提供match、search等方法，import导入的模块，在调用其方法时需要使用：模块名.方法
()
import re
# os模块提供了多数操作系统的功能接口函数，如getcwd、path.abspath、path.split等
import os
# logging模块提供了日志接口功能，如logger等
import logging
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
# devicemgr提供了一些查询设备信息的接口。
from aoc import NcsService, devicemgr
# UserException是系统定义的异常接口，可以把用户自定义的异常呈现到前台界面
from aoc.exception.aocexceptions import UserException
# 引入EVPN过滤器
from com.huawei.awang.evpn.filter.filter import Filter


# AocNcsEvpnService继承NcsService类并重写ncs_map方法
class AocNcsEvpnService(NcsService):
    ......
    def ncs_map(self, request, aoc_context=None, template=''):
        """
            重写ncs_map方法编译获取创建VRF所需的参数；
```

 版权所有 © 华为技术有限公司

```
        request 参数：XML包，包含设备配置数据；
        aoc_context参数：字典类型，包含设备属性信息，默认值为None；
                properties['transactionId']
                properties['deviceType']
                properties['deviceVendor']
                properties['deviceVersion']
        template参数为jinja2模板，默认值为空；
        return：返回值为基于 Jinja2 模板映射的数据包。
        """
        self.logger.info("[AocNcsEvpnService][ncs_map] request is: %s, context is: %s, "
                         "template is s%s" % (request.xml, aoc_context, template))
        # xmltodictnode()将xml类型转变为dict类型
        evpn_dict = self.xmltodictnode(request.xml)
        self.logger.info("[AocNcsEvpnService][ncs_map] evpn_dict is: %s" % evpn_dict)
        response = None

        # 将 XML 数据映射到基于site的 NE YANG（网元YANG）结构。
        if evpn_dict is None or 'site' not in evpn_dict \
                or 'site-network-accesses' not in evpn_dict['site'] \
                or 'site-network-access' not in evpn_dict['site']['site-network-accesses'] \
                or 'vpn-service' in evpn_dict:
            # 使用 Jinja2 模板将 XML 数据映射到 NE YANG 结构。
            return self.render('template-inventory-cfg.j2', {})

        # 将 XML 数据映射到基于service的 NE YANG（网元YANG） 结构。
        nes = []
        template_contexts = []
        for network_access in evpn_dict['site']['site-network-accesses']['site-network-access']:
            if 'bearer' not in network_access or 'requested-type' not in network_access['bearer'] \
                    or 'requested-type-profile' not in network_access['bearer']['requested-type'] \
                    or 'pe-name' not in network_access['bearer']['requested-type'][ \
                    'requested-type-profile']:
                continue
            # 获取输入的pe_name的值，赋值给ne_ip
            ne_ip = network_access['bearer']['requested-type']['requested-type-profile']['pe-name']
            # get_ne_id()：参数为设备ip（ne_ip），返回值为设备id
            ne_id = self.get_ne_id(ne_ip)
            if ne_id not in nes:
                nes.append(ne_id)
                template_context = {
                    'ne_id': ne_id,
                    'ne_ip': ne_ip,
                    # 参数为设备ip（ne_ip），返回值为映射模板（.j2）的路径
                    'template': self.find_ne_template(ne_ip)
                }
                template_contexts.append(template_context)
        inventory_template_content = {
            'nes': template_contexts,
            'l2evpns': evpn_dict
        }
        self.logger.info("inventory_template_content: %s" % inventory_template_content)
        # 通过jinja2模型将XML数据映射到网元结构。
        response = self.render('template-inventory-cfg.j2', inventory_template_content)
        self.logger.info("[AocNcsL3vpnService][ncs_map] response is: %s" % response)
        return response

    def find_ne_template(self, ne_ip):
        """
        通过设备信息获取相应jinja模板；
        参数为ne_ip；
        返回值为jinja模板的路径
        """
        template = None
        if str(ne_ip) == '':
            return template
        # 启动设备模板适配器.
        self.load_tpl_adapter()
        try:
            self.logger.debug("[AocNcsEvpnService][find_ne_template] query ne : %s" % ne_ip)
```

 版权所有 © 华为技术有限公司

```
        device_info_page = devicemgr.query_basic_data_from_db('', '', ne_ip)
        """
        query_basic_data_from_db()用于查询设备详细信息；
        neid参数：AOC的设备id，默认值为空；
        nename参数：设备的系统名称，默认值为空；
        nemanageip参数：设备的管理IP，此处为ne_ip，默认值为空；
        返回值为包含各项设备信息的列表页。
        """
        self.logger.debug("[AocNcsEvpnService][find_ne_template] query ne[%s] result: %s"
                          % (ne_ip, device_info_page))

        device_info = None
        for entity in device_info_page.queryDevicesPagedEntity:
            if entity.neId is not None:
                device_info = entity
                break
    except Exception as exception:
        self.logger.error("[AocNcsEvpnService][find_ne_template]"
                          " query ne error. %s" % exception)
        raise UserException("The device doesn't exist: %s" % ne_ip)

    if device_info is not None:
        for tpl in self.template_adapter.template:
            """
            基于设备的供应商、类型、版本去匹配相应的jinja模板
            """
            vendormatch = re.match(str(tpl.deviceVendor).lower(), device_info.manufacturer.lower())
            typematch = re.match(str(tpl.deviceType), device_info.hardWare)
            versionmatch = re.match(str(tpl.deviceVersion), device_info.softWare)
            if vendormatch and typematch and versionmatch:
                # 获取jinja模板的路径
                template = tpl.path
                break
    if template is None:
        self.logger.error("[AocNcsEvpnService][find_ne_template] template "
                          "is not exist : %s" % device_info)
        raise UserException("The device doesn't support: %s" % ne_ip)
    self.logger.debug("[AocNcsEvpnService][find_ne_template] template for ne[%s] is %s"
                      % (ne_ip, template))
    return template

def get_ne_id(self, ne_ip):
    """
    通过设备ip获取设备id
    ne_ip参数：设备ip
    返回值为设备id
    """
    template = None
    if str(ne_ip) == '':
        return template
    # 启动设备模板适配器.
    self.load_tpl_adapter()
    try:
        self.logger.debug("[VivoL3vpn][get_ne_id] query ne : %s" % ne_ip)
        device_info_page = devicemgr.query_basic_data_from_db('', '', ne_ip)
        self.logger.info("device_info_page:%s" % device_info_page)
        self.logger.debug("[VivoL3vpn][get_ne_id] query ne[%s] result: %s"
                          % (ne_ip, device_info_page))

        device_info = None
        for entity in device_info_page.queryDevicesPagedEntity:
            if entity.neId is not None:
                device_info = entity
                break
    except Exception as exception:
        self.logger.error("[VivoL3vpn][get_ne_id]query ne error. %s" % exception)
        raise UserException("The device doesn't exist: %s" % ne_ip)

    return device_info.neId
```

 版权所有 © 华为技术有限公司

```
def load_tpl_adapter(self):
    """
    # 启动设备模板适配器.
    """
    if self.template_adapter is not None:
        return
    try:
        # 读取resources目录下的template-adapter.xml文件，获取设备与j2文件的对应关系
        resources_path = os.path.join(self.resource_dir, 'resources')
        adapter_path = os.path.join(resources_path, 'template-adapter.xml')
        self.logger.info("[AocNcsEvpnService][load_tpl_adapter]"
                         " load device adapter file path : %s" % adapter_path)
        with open(adapter_path, "r") as adapter_fp:
            template_adapter_str = adapter_fp.read()
            # 将xml格式的模板信息转换为dictnode格式
            self.template_adapter = self.xmltodictnode(template_adapter_str).templates
            self.logger.info("[AocNcsEvpnService][load_tpl_adapter] "
                             "load device adapter success : %s" % template_adapter_str)
    except IOError as io_error:
        self.logger.error("[AocNcsEvpnService][load_tpl_adapter] "
                          "adapter file is invalid. %s" % io_error)
        raise UserException("Adapter file is invalid")
```

步骤5 开发Jinja2模板

把编辑的内容放入到模板文件“template> l2evpn.j2”中

```
<evpn xmlns="urn:huawei:yang:huawei-evpn" xmlns:ns0="urn:huawei:ac" ns0:operation="create">
  {%- set evpn_name = evpn['evpn-name'] | special_char %}
  <vpn-instance>{{ evpn_name }}</vpn-instance>
  <vpws>vpws</vpws>
  <route-distinguisher>{{ evpn['route-distinguisher'] }}</route-distinguisher>

  {%- if evpn['description'] %}
    {%- set des_without_specil_char = evpn['description'] | special_char %}
    <description>{{ des_without_specil_char }}</description>
  {%- endif %}

  {%- if evpn['export-route-targets'] %}
    {%- for rt in evpn['export-route-targets'] %}
      <vpn-target>
        <number>{{ rt }}</number>
        <vt-type>export-extcommunity</vt-type>
      </vpn-target>
    {%- endfor  %}
  {%- endif %}

  {%- if evpn['import-route-targets'] %}
    {%- for rt in evpn['import-route-targets'] %}
      <vpn-target>
        <number>{{ rt }}</number>
        <vt-type>import-extcommunity</vt-type>
      </vpn-target>
    {%- endfor  %}
  {%- endif %}
</evpn>
```

将业务扩展映射到不同款型的设备，编写调用对应款型设备Jinja2模板的逻辑，将编辑好的内容放入到模板文件“template >template-inventory-cfg.j2”中。

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
  {%- for ne in nes %}
    <ne>
      <neid>{{ ne.ne_id }}</neid>
      {%- include ne.template %}
    </ne>
  {%- endfor %}
```

 版权所有 © 华为技术有限公司

```
  </nes>
</inventory-cfg>
```

----结束

7.3.3 开发 L3EVPN SSP 示例

本示例介绍如何通过下载的默认SSP包，开发一个L3EVPN SSP包，用户自定义一个L3EVPN网络业务(下发到PE设备NE5000E)，并在北向开放此网络业务。

步骤1 创建SSP模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。
   请根据表7-12填写软件包的相关属性值。

   表 7-12 L3EVPN 软件包属性

属性名称	属性值
名称	SSP-CUII-L3VPN
版本	1.0.25
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	Cuii_l3vpn
业务定义点	Cuii_l3vpn
映射类型	python+template

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
    "name": "SSP-CUII-L3VPN",
    "version": "1.0.25",
    "description": "l3vpn for cuii",
    "package-type": "ssp",
    "producer": "huawei",
    "service-name":"cuii-l3vpn",
    "nce-min-versions":[
        "1.0.0",
        "2.0.0"
    ],
    "hooks": [
      {
        "type": "mapping",
        "key": "vpnsvc",
        "python-class-name":"com.huawei.aunicom.vrf.aoc_ncs_l3vpn.AocNcsL3vpnService"
      },
{
        "type": "mapping",
```


版权所有 © 华为技术有限公司

```
        "key": "site",
        "python-class-name":"com.huawei.aunicom.vrf.aoc_ncs_l3vpn.AocNcsL3vpnService"
      },
{
        "type": "mapping",
        "key": "initConfig",
        "python-class-name":"com.huawei.aunicom.vrf.aoc_ncs_l3vpn.AocNcsL3vpnService"
      },
    ]
}
```

步骤3 开发业务YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表7-13所示，并在YANG模型中进行定义。

表 7-13 整理的参数

类别	描述	参数名称			参数值
VRF name	创建一个VPN实例，配置名称。	vrf-prefix			BMW-HIN
VPN基础配置	配置VPN实例id等信息	source-pe			-
		destination-pe			-
		tunnel-id			Tunnel100
		forward-class			-
		vrf-id			ICBC-35
QOS设置	配置QOS相关参数	class-profile	qos-class (used by: inbound-classes and outbound-classes)	class-id	IPP5
				class-name	DCI-IPP5
				behavior-name	DCI-AF4
				rate-limit	30
				priority-level	1
				guaranteed-bw-percent	class-id为IPP4时使用
		qos-share			share-mode
		qos-share-id			shshDCIoo4-4
		modify-timestamp			-

版权所有 © 华为技术有限公司

类别	描述	参数名称				参数值
requested-type信息	配置接口、vlan相关参数	requested-type profile	requested-type choice	dot1q-case	physical-if	-
					vlan-id	-
				physical-case	physical-if	-
					physical-if	GigabitEthernet1/0/0.10012001
				qinq-case	s-vlan-id	1001
					c-vlan-id	2001
			int-description			-
			circuit-id			shshDCloo4-4_i
rd, rt信息	配置bfd、bgp、rdrt相关参数	bfd-grouping	bfd-enabled			enable
			echo-disable			-
			holdtime	profile-name		-
				fixed-value		-
				dynamic-value	interval	500
					multiplier	4
		bgp-profile	as-override			true
			soo	soo-enabled		true
				soo-value		100:10
			password	password-enabled		true
				password-value		1234abcd
			bgp-timer	keep-alive		10
				hold-time		30
			bfd (uses bfd-grouping)			-
			bgp-max-prefix	max-prefix		15000
				warning-threshold		80
				violation-cation	warning	-

版权所有 © 华为技术有限公司

类别	描述	参数名称				参数值
					discard-extra-paths	-
					restart	5
				restart-interval		-
			routing-policies	routing-policy-in		-
				routing-policy-out		-
				deny-all		-
				default-route		true
			routing-aggregations	network		10.1.1.0
				mask		255.255.255.0
				summary-only		true
				as-set		true
		constraint-opaque-value2				-
		route-ipv4-extended-community	global-administrator			-
			local-administrator			-
		extended-community	route-target-extended-community-case	global-administrator		-
				local-administrator		-
			route-target-ipv4-case (uses route-ipv4-extended-community)			-
			route-target-extended-community-case2	global-administrator		-
				local-administrator		-
		rdrt-profile	vrf-id			default: vpn-id

版权所有 © 华为技术有限公司

类别	描述	参数名称				参数值
			id			default: vrf-id
			custom	import-rt	imrt-id (uses extended-community)	-
				export-rt	exrt-id (uses extended-community)	-
				rd		-

最终编辑完成的L3EVPN软件包YANG模型如下。

1. 在YANG模型中定义VRF name

```
augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:vpn-services/l3vpn:vpn-svc" {
      leaf vrf-prefix {
         type string {
            length "1..3";
            pattern "[A-Z]*";
         }

         description "Controller use '-' connect vrf-prefix and vrf-id to generate real VRF name on the device.";
      }
   }
```

2. 在YANG模型中定义VPN基础配置

```
   /*tunnel service*/
   augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:vpn-services/l3vpn:vpn-svc/l3vpn:transport-constraints/l3vpn:unicast-transport-constraints/l3vpn:constraint" {
      leaf source-pe {
         type string;
      }
      leaf destination-pe {
         type string;
      }
      leaf tunnel-id {
        type uint16;
      }
      leaf forward-class {
        type uint8;
      }
      leaf-list vrf-id {
         type string;
         description "If vrf-id is not provided, use vpn-id as the default";
      }
   }
```

3. 在YANG模型中定义QOS配置

```
   grouping class-profile {
      list qos-class {
         key class-id;
```

 版权所有 © 华为技术有限公司

```
        leaf class-id {
            type string;
            description
             "Identification of the
             class of service.
             This identifier is internal to
             the administration.";
        }
        leaf class-name {
            type string;
            description
             "class map name";
        }
        leaf behavior-name {
            type string;
            description
             "behavior name";
        }
        leaf rate-limit {
            type uint8;
            units percent;
            description
             "To be used if class must
             be rate
             limited. Expressed as
             percentage of the svc-bw.";
        }
        leaf priority-level {
            type uint8;
            description
             "Defines the level of the
             class in
             term of priority queueing.
             The higher the level is the
             higher
             is the priority.";
        }
        leaf guaranteed-bw-percent {
            type uint8;
            units percent;
            description
             "To be used to define the
             guaranteed
             BW in percent of the svc-bw
             available at the priority-level.";
        }
        description
         "List of class of services.";
    }
  }

   augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:service/l3vpn:qos" {
      leaf qos-share {
          type boolean;
          default false;
      }
      leaf qos-share-id {
          when "../qos-share = 'true'";
          type string{
              pattern "[A-Z][A-Z0-9\-]*";
          }
          description "Controller use qos-share-id generate qos-profile id.";
      }
      leaf modify-timestamp {
          type uint64;
          description "Indicates the millisecond-level timestamp of the last modification of QoS.";
      }
  }
```


版权所有 © 华为技术有限公司

```
  augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
      case custom-unicom {
          container inbound-classes {
              uses class-profile;
          }
          container outbound-classes {
              uses class-profile;
          }
      }
  }
```

4. 在YANG模型中定义requested-type信息

```
  augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:bearer" {
      container bearer-attachment {
          leaf pe-name {
              type string;
          }
          leaf pe-mgmt-ip {
              type inet:ipv4-address;
          }
          description "attached PE";
      }
  }

  grouping requested-type-grouping {
      container requested-type-profile {
          choice requested-type-choice {
              case dot1q-case {
                  container dot1q {
                      leaf physical-if {
                          description "physical interface name.";
                          type string;
                      }
                      leaf vlan-id {
                          type uint16 {
                              range "1..4096";
                          }
                      }
                  }
              }
              case physical-case {
                  container physical {
                      leaf physical-if {
                          description "physical interface name.";
                          type string;
                      }
                  }
              }
              case qinq-case {
                  container qinq {
                      leaf physical-if {
                          description "physical interface name.";
                          type string;
                      }
                      leaf s-vlan-id {
                          type uint16 {
                              range "1..4096";
                          }
                          description "Outer vlan id, service provider vlan";
                      }
                      leaf c-vlan-id {
                          type uint16 {
                              range "1..4096";
                          }
                          description "Inner vlan id, customer vlan";
                      }
                  }
```

 版权所有 © 华为技术有限公司

```
            }
        }
        leaf int-description {
            type string {
              length "1..200";
            }
        }
        leaf circuit-id {
            description "circuit description for PE-CE port.";
            type string;
        }
    }
}
```

5. 在YANG模型中定义bfd信息

```
  augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:bearer/l3vpn:requested-type" {
    uses requested-type-grouping;
}

grouping bfd-grouping {
    leaf bfd-enabled {
        type boolean;
        description
         "BFD activation";
    }
    leaf echo-disable {
        type boolean;
            description
            "BFD echo status";
        default false;
    }
    choice holdtime {
        case profile {
            leaf profile-name {
                type string;
                description
                 "Service provider well
                 known profile.";
            }
            description
             "Service provider well
             known profile.";
        }
        case fixed {
            leaf fixed-value {
                type uint32;
                units msec;
                description
                 "Expected holdtime
                 expressed
                 in msec.";
            }
        }
        case dynamic {
            container dynamic-value {
                leaf interval {
                    type uint16;
                    units msec;
                    default 500;
                }
                leaf multiplier {
                    type uint16;
                    default 3;
                }
                description
                 "interval * multiplier is
                 timeout value.";
            }
        }
```

 版权所有 © 华为技术有限公司

```
    }
}

grouping bgp-profile {
    leaf as-override {
        type boolean;
        default false;
    }
    container soo {
        leaf soo-enabled {
            type boolean;
        }
        leaf soo-value {
            type string;
        }
    }
    container password {
        leaf password-enabled {
            type boolean;
        }
        leaf password-value {
            type string;
        }
    }
    container bgp-timer {
        leaf keep-alive {
            type uint16;
            default 60;
            units "seconds";
        }
        leaf hold-time {
            type uint16;
            default 180;
            units "seconds";
        }
    }
    container bfd {
        uses bfd-grouping;
    }
    container bgp-max-prefix {
        leaf max-prefix {
          type uint32;
          default 5000;
        }
        leaf warning-threshold{
          type uint8;
          default 75;
        }
        leaf violate-action {
          type enumeration {
            enum warning {
              description "Only give warning message when limit is exceeded";
            }
            enum discard-extra-paths {
              description "Discard extra paths when limit is exceeded";
            }
            enum restart {
              description "Restart time interval";
            }
          }
          description
           "bpg neighbour max-prefix violate action";
        }
        leaf restart-interval {
          type uint16;
          description "Time interval (min) after which peering session will be reestablished";
        }
    }
    container routing-policies {
```


版权所有 © 华为技术有限公司

```
        leaf routing-policy-in {

            type string ;
            description
             "routing policy name";
          }
        leaf routing-policy-out {

            type string ;
            description
             "routing policy name";
        }
        leaf deny-all {
          type string;
          description
              "deny-all routing policy name";
        }
        leaf default-route {
          type boolean;
          default false;
        }
      }
      container routing-aggregations {
        list routing-aggregation {
          key "network mask";
          leaf network {
            type inet:ipv4-address;
            description
             "Aggregation network.";
          }
          leaf mask {
            type uint8 {
              range "0..32";
            }
            description
             "Subnet mask expressed in bits";
          }
          // Hidden leaf populated by service.
          // Huawei templates can us this leaf directly to get mask on xxx.xxx.xxx.xxx format
          // This is hidden for all users
          leaf net-mask {
            type inet:ipv4-address;
          }
          leaf summary-only {
            type boolean;
            default true;
            description
             "";
          }
          leaf as-set {
            type boolean;
            default true;
            description
             "";
          }
        }
      }
    }
    augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:routing-protocols/l3vpn:routing-protocol/l3vpn:bgp" {
      uses bgp-profile;
    }
    augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:routing-protocols/l3vpn:routing-protocol/l3vpn:static" {
      uses bfd-grouping;
    }
```

6. 在YANG模型中定义rd, rt信息

```
    augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:vpn-services/l3vpn:vpn-svc/l3vpn:transport-
constraints/l3vpn:unicast-transport-constraints/l3vpn:constraint/l3vpn:constraint-list" {
```

 版权所有 © 华为技术有限公司

```
    leaf constraint-opaque-value2 {
        type string;
        description
         "Opaque value that can be used to
         specify constraint parameters.";
    }
}
grouping route-ipv4-extended-community {
    reference "http://tools.ietf.org/html/rfc4360";
    leaf global-administrator {
        type inet:ipv4-address;
    }
    leaf local-administrator {
        type uint16;
    }
}
grouping extended-community {
    choice extended-community {
        reference "http://tools.ietf.org/html/rfc4360#section-4";
        default route-target-extended-community-case;
        case route-target-extended-community-case {
            container route-target-extended-community {
                leaf global-administrator {
                    type short-as-number;
                }
                leaf local-administrator {
                    type uint32;
                }
            }
        }
        case route-target-ipv4-case {
            container route-target-ipv4 {
                uses route-ipv4-extended-community;
            }
        }
        case route-target-extended-community-case2 {
            container route-target-extended-community2 {
                leaf global-administrator {
                    type uint32;
                }
                leaf local-administrator {
                    type uint16;
                }
            }
        }
    }
}
/*version2.1*/
grouping rdrt-profile {
    leaf vrf-id {
        type string;
        description "If vrf-id is not provided, use vpn-id as the default";
    }
    // This leaf is never visible for any user
    // Service callback will populate it with vrf-id if it is set and vpn-id if vrf-id is not set.
    // Templates refer to this leaf instead of vrf-id.
    leaf id {
        config false;
        description "If vrf-id is set, this is the same as vrf-id. If not this is the same as vpn-id";
        type string;
    }
    choice site-role {
        case custom-case {
            container custom {
                list import-rt {
                    key imrt-id;
                    leaf imrt-id {
                      type string;
                    }
```


版权所有 © 华为技术有限公司

```
                uses extended-community;
            }
            list export-rt {
                key exrt-id;
                leaf exrt-id {
                  type string;
                }
                uses extended-community;
            }
            leaf rd {
                type route-distinguisher;
            }
        }
      }
    }
    augment "/app:applications/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/
l3vpn:site-network-access/l3vpn:vpn-attachment" {
        uses rdrt-profile;
    }
}
```

步骤4 开发代码

实现EasyMap功能，需要继承NcsService并重写ncs_map方法，实现业务逻辑。

L3EVPN软件包代码示例如下所示。

```
# 正则表达式模块re，提供match、search等方法，import导入的模块，在调用其方法时需要使用：模块名.方法
()
import re
# os模块提供了多数操作系统的功能接口函数，如getcwd、path.abspath、path.split等
import os
# logging模块提供了日志接口功能，如logger等
import logging
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
# devicemgr提供了一些查询设备信息的接口。
from aoc import NcsService, devicemgr
# UserException是系统定义的异常接口，可以把用户自定义的异常呈现到前台界面
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
# devicemgr提供了一些查询设备信息的接口。
from aoc import NcsService, devicemgr
# UserException是系统定义的异常类，可以把用户自定义的异常呈现到前台界面
from aoc.exception.aocexceptions import UserException
from com.huawei.aunicom.vrf.filter.filter import Filter
from com.huawei.aunicom.vrf.util import Util

class AocNcsL3vpnService(NcsService):
    def __init__(self, logger=None, resource_dir=''):
        """
        初始化AocNcsL3vpnService
        """
        self.template_adapter = None
        devicemgr.logger = logger
        if logger is None:
            devicemgr.logger = logging
        NcsService.__init__(self, logger, resource_dir)

    def ncs_map(self, request, aoc_context=None, template=''):
        """
        此函数通过覆盖NcsService父类中的ncs_map，来编译并获得新建一个VRF所需的参数，通过不同的模板映
射，下发不同的设备配置
        :param request: 一个XML包，包含了业务配置数据
        :param aoc_context: 一个dict，其中包括
                properties['transactionId'] 事务id
                properties['deviceType'] 设备种类
                properties['deviceVendor'] 设备供应商
                properties['deviceVersion'] 设备版本
        :param template: jinja2模板
        :return: 此函数的返回值，是一个依据jinja2模板映射而成的包
        """
```

 版权所有 © 华为技术有限公司

```
    start_time = time.time()
    self.logger.info("[AocNcsL3vpnService][ncs_map] request is: %s, context is: %s"
                     ", template is: %s" % (Util.safe_str(request.xml), aoc_context, template))
    l3vpn_dict = self.xmltodictnode(request.xml)

    response = None
    if l3vpn_dict and l3vpn_dict['site']:
        # Build an NE template based on site
        itc = self.get_site_template_content(aoc_context, l3vpn_dict)
        # Use the Jinja2 template to map XML data to the NE YANG structures.
        response = self.render('template-inventory-cfg.j2', itc)
        self.logger.info("inventory_template_content: %s" % Util.safe_str(itc))

    if l3vpn_dict and l3vpn_dict['vpn-svc']:
        # Build an NE template based on vpnsvc
        itc = self.get_vpnsvc_template_content(aoc_context, l3vpn_dict)
        # Use the Jinja2 template to map XML data to the NE YANG structures.
        response = self.render('template-inventory-cfg.j2', itc)

    self.logger.info("[AocNcsL3vpnService][ncs_map] cost time is %s seconds, response is: %s"
                     % ((time.time() - start_time), Util.safe_str(response)))
    return response

def get_vpnsvc_template_content(self, aoc_context, l3vpn_dict):
    """
    函数get_vpnsvc_template_content，用于获取vpnsvc service yang结构模板
    :param aoc_context: 一个dict，其中包括
            properties['transactionId'] 事务id
            properties['deviceType'] 设备种类
            properties['deviceVendor'] 设备供应商
            properties['deviceVersion'] 设备版本
    :param l3vpn_dict: l3vpn的dict, 内容来自request.xml
    :return: 此函数的返回值，是经过封装的仓库内模板内容，type dict
    """
    ne_ips = []
    ne_infos = []
    for vpn_svc in l3vpn_dict['vpn-svc']:
        if vpn_svc['transport-constraints'] and \
                vpn_svc['transport-constraints']['unicast-transport-constraints'] and \
                vpn_svc['transport-constraints'] \
                        ['unicast-transport-constraints']['constraint']:
            for constraint in vpn_svc['transport-constraints'] \
                    ['unicast-transport-constraints']['constraint']:
                if constraint['source-pe']['#text'] not in ne_ips:
                    device_ip = constraint['source-pe']['#text']
                    ne_ips.append(device_ip)
                    self.logger.info("svc device_ip: %s" % device_ip)
                    template = self.find_ne_template(device_ip, "vpnsvc")
                    self.logger.info("svc temple: %s" % template)
                    template_context = {
                        'ne_id': self.query_ne_id_by_ne_ip(device_ip),
                        'ne_ip': device_ip,
                        'template': template
                    }
                    ne_infos.append(template_context)
    inventory_template_content = {
        'nes': ne_infos,
        'vpn_svcs': l3vpn_dict,
        'aoc_context': aoc_context
    }
    return inventory_template_content

def get_site_template_content(self, aoc_context, l3vpn_dict):
    """
    函数get_site_template_content用于获取site yang结构模板
    :param aoc_context: 一个dict，其中包括
            properties['transactionId'] 事务id
            properties['deviceType'] 设备种类
            properties['deviceVendor'] 设备供应商
```

 版权所有 © 华为技术有限公司

```
            properties['deviceVersion'] 设备版本
        :param l3vpn_dict: l3vpn的dict, 内容来自request.xml
        :return: 此函数的返回值，是经过封装的仓库内模板内容，type dict
        """
        nes = []
        l3vpns = []
        # 根据业务YANG模型从pe-name字段中获取设备ip 并使用设备ip获取对应模板
        if l3vpn_dict['site']['site-network-accesses'] and \
                l3vpn_dict['site']['site-network-accesses']['site-network-access']:
            for site_network in \
                    l3vpn_dict['site']['site-network-accesses']['site-network-access']:
                if site_network['bearer'] and \
                        site_network['bearer']['bearer-attachment'] and \
                        site_network['bearer']['bearer-attachment']['pe-name'] and \
                        site_network['bearer']['bearer-attachment']['pe-name'] not in nes:
                    device_ip = site_network['bearer']['bearer-attachment']['pe-name']
                    nes.append(device_ip)
                    # 通过设备ip地址，获取所需的模型
                    template = self.find_ne_template(device_ip, "site")
                    self.logger.info("temple: %s" % template)
                    template_context = {
                        # 通过网元ip，获取网元id编码
                        'ne_id': self.query_ne_id_by_ne_ip(device_ip),
                        'ne_name': device_ip,
                        'template': template
                    }
                    l3vpns.append(template_context)
        inventory_template_content = {
            'nes': l3vpns,
            'l3vpns': l3vpn_dict,
            'aoc_context': aoc_context
        }
        return inventory_template_content

    def query_ne_id_by_ne_ip(self, ne_ip):
        """
        函数query_ne_id_by_ne_ip，是通过网元ip地址，从数据库中查询获取网元id编码
        :param ne_ip: 网元设备的ip地址
        :return: 此函数的返回值，是网元设备的id编码
        """
        ne_id = None
        """
        函数query_basic_data_from_db，是用于从数据库中获取设备的具体信息
        :param neid: AOC中设备的ip，是一个string格式的字符串，默认值为空
        :param nename: 设备的系统名，是一个string格式的字符串，默认值为空
        :param nemanageip:str, manage ip of device, default value is ''
        :return response_body: 此函数的返回值，是一个设备信息包，包内的queryDevicePagedEntity是一个list格式的数据，里面包含string格式的设备属性:  neId, name, hardWare, softWare, manageIp, type, manufacturer, status, manageMac, deviceModel, esn, location, description, layer, get_channel_index, edit_channel_index (网元id编码，名称，硬件，软件，管理ip，种类，生产商，状态，管理MAC，设备模型，esn (equipment serial number)设备序列号，位置，描述，分层，获取途径索引，编辑途径索引)
        """
        result = devicemgr.query_basic_data_from_db('', '', ne_ip)
        if result is not None:
            for entity in result.queryDevicesPagedEntity:
                ne_id = entity.neId
        self.logger.info("query_ne_id_by_ne_ip ne_ip: %s, ne_id: %s" % (ne_ip, ne_id))
        return ne_id

    def find_ne_template(self, ne_ip, service):
        """
        函数find_ne_template，通过网元ip地址，获取所需的模板路径
        :param ne_ip: 网元ip地址
        :param service: site或vpnsvc
        :return: 此函数的返回值，是模板路径，type string，可在resources文件夹中的template-adapter.xml里查看。如果service=site，查看<path>部分，如果service=svc，查看<svcpath>
        """
        template = None
        if str(ne_ip) == '':
```

 版权所有 © 华为技术有限公司

```
        self.logger.info("find_ne_template ne_ip is empty!")
        return template

    # 加载设备模板
    self.load_tpl_adapter()
    try:
        self.logger.debug("[AocNcsL3vpnService][find_ne_template] query ne : %s" % ne_ip)
        # 在数据库中查询设备的详细信息
        device_info_page = devicemgr.query_basic_data_from_db('', '', ne_ip)
        self.logger.debug(
            "[AocNcsL3vpnService][find_ne_template] query ne[%s] result: %s"
            % (ne_ip, device_info_page))
        device_info = None
        for entity in device_info_page.queryDevicesPagedEntity:
            if entity.neId is not None:
                device_info = entity
                break
    except Exception as exception:
        self.logger.error("[AocNcsL3vpnService][find_ne_template] query "
                          "ne error. %s" % exception)
        raise UserException("The device doesn't exist: %s" % ne_ip)

    if device_info is not None:
        for tpl in self.template_adapter.template:
        """
            根据供应商，设备类型，和设备版本，来筛选出匹配此设备的模板
            """
            vendor_match = re.match(str(tpl.deviceVendor), device_info.manufacturer, re.IGNORECASE)
            type_match = re.match(str(tpl.deviceType), device_info.hardWare)
            version_match = re.match(str(tpl.deviceVersion), device_info.softWare)
            if vendor_match and type_match and version_match:
                self.logger.debug("[AocNcsL3vpnService][find_ne_template]match tpl : %s" % tpl)
                # 根据单设备或多设备业务来选择不同的j2模板
                if service == "site":
                    template = tpl.path
                elif service == "vpnsvc":
                    template = tpl.svcpath
                break
    if template is None:
        self.logger.error("[AocNcsL3vpnService][find_ne_template] "
                          "template is not exist : %s" % device_info)
    self.logger.debug("[AocNcsL3vpnService][find_ne_template] template for ne[%s] "
                      "is %s" % (ne_ip, template))
    return template

def load_tpl_adapter(self):
    """
    加载设备模型
    """
    if self.template_adapter is not None:
        self.logger.debug("load_tpl_adapter template_adapter is not None: %s" %
                          self.template_adapter)
        return
    try:
        # 读取resources目录下的template-adapter.xml文件获取设备与j2文件的对应关系
        resources_path = os.path.join(self.resource_dir, 'resources')
        adapter_path = os.path.join(resources_path, 'template-adapter.xml')
        self.logger.info(
            "[AocNcsL3vpnService][load_tpl_adapter] load device adapter file "
            "path : %s" % adapter_path)
        adapter_fp = open(adapter_path, "r")
        template_adapter_str = adapter_fp.read()
        # 将xml格式的模板信息转换为dictnode格式
        self.template_adapter = self.xmltodictnode(template_adapter_str).templates
        self.logger.info(
            "[AocNcsL3vpnService][load_tpl_adapter] load device adapter "
            "success : %s" % template_adapter_str)
    except IOError as io_error:
        self.logger.error("[AocNcsL3vpnService][load_tpl_adapter] adapter file "
```

 版权所有 © 华为技术有限公司

```
                        "is invalid : %s" % io_error)
            raise UserException("Adapter file is invalid")
        finally:
            adapter_fp.close()

    def add_filters(self):
        """
        用户定义的滤镜，来覆盖add_filters
        :return: 目标数据
        """
        self.logger.info("[AocNcsL3vpnService] add filters")
        return Filter(self.logger).all_filter()

    def setup(self):
        """
        Setup.
        """
        self.logger.info("set up")

    def teardown(self):
        """
        Teardown.
        """
        self.logger.info("teardown")
```

步骤5 开发Jinja2模板

把编辑的内容放入到模板文件“template>l3vpn>NE8000M8.j2”中。

```
{# static routes #}
{# CLI Example #}
{# <HUAWEI> system-view #}
{# [~HUAWEI] ip route-static vpn-instance BMW-HIN 10.10.10.10 255.255.255.252 GigabitEthernet0/0/0
10.2.2.2 bfd enable #}

{# Get vrf list by ne name #}
{%- if l3vpns and l3vpns | vrfs_by_ne_name(ne['ne_name']) %}
    {%- set vrfs = l3vpns | vrfs_by_ne_name(ne['ne_name']) %}
    {%- for vrf in vrfs %}
        {# Get static routes by ne name #}
        {%- set static_routes_access = l3vpns | static_routes_by_routing_protocol(ne['ne_name']) %}
        {# Get ipv4 static routes of static_routes #}
        {%- set has_ipv4_static_access = static_routes_access | has_ipv4_static_in_static_routes %}
        {%- if has_ipv4_static_access %}
            <ip xmlns="urn:huawei:yang:huawei-static-route">
                <route-static>
                    {%- set vrf_prefix = aoc_context | query_vrf_prefix(vrf['vpn_id']) %}
                    {%- for static_route_access in static_routes_access %}
                        {%- if static_route_access['vrf_id'] == vrf['vrf_id'] and static_route_access['ipv4_statics'] %}
                            {%- for ipv4_static_route in static_route_access['ipv4_statics'] %}
                                <vpn-instance xmlns:ns0="urn:huawei:ac" ns0:operation="create">
                                    {# Replace special characters in static_route #}
                                    {%- set vrf_name_without_specil_char = static_route_access['vrf_id'] |
special_char_in_vrfname %}
                                    {% if vrf_prefix %}
                                        <vrfName>{{ vrf_prefix}}-{{ vrf_name_without_specil_char }}</vrfName>
                                    {%- else %}
                                        <vrfName>{{ vrf_name_without_specil_char }}</vrfName>
                                    {%- endif %}
                                    {# Subnet mask length to address #}
                                    {%- set sub_network = ipv4_static_route['lan'].split("/")[0] |
convert_sub_network(ipv4_static_route['lan'].split("/")[1]) %}
                                    <destination>{{ sub_network }}</destination>
                                    <mask>{{ ipv4_static_route['lan'].split("/")[1] | convert_ipv4_mask() }}</mask>
                                    <ifName>{{ ipv4_static_route['interface_name'] }}</ifName>
                                    <nexthop>{{ ipv4_static_route['next-hop'] }}</nexthop>
                                    {%- if ipv4_static_route['bfd-enabled'] and ipv4_static_route['bfd-enabled'] ==
'true' %}
                                        <bfd>enable</bfd>
                                    {%- endif %}
```

 版权所有 © 华为技术有限公司

```
                    </vpn-instance>
                {%- endfor %}

                {%- for bfd_info in static_route_access['bfd_config_for_huawei'] %}
                    {%- if bfd_info['bfd-enabled'] == 'true'  %}
                    <bfd>
                        <ifName>{{ bfd_info['interface_name'] }}</ifName>
                        {%- if bfd_info['customer-address'] %}
                            <nexthop>{{ bfd_info['customer-address'] }}</nexthop>
                        {%- endif %}
                        <local-address>{{ bfd_info['provider-address'] }}</local-address>
                        {%- if bfd_info['interval'] %}
                            <min-rx-interval>{{ bfd_info['interval'] }}</min-rx-interval>
                            <min-tx-interval>{{ bfd_info['interval'] }}</min-tx-interval>
                        {%- endif %}
                        {%- if bfd_info['multiplier'] %}
                            <detect-multiplier>{{ bfd_info['multiplier'] }}</detect-multiplier>
                        {%- endif %}
                    </bfd>
                    {%- endif %}
                {%- endfor %}
            {%- endif %}
        {%- endfor %}
    </route-static>
</ip>
{%- endif %}
{%- endfor %}
{%- endif %}
```

----结束

7.3.4 开发业务还原 SSP 示例

本示例介绍如何通过下载的默认SSP包，开发一个业务还原SSP包，定义网元层业务还原到网络层业务逻辑，实现网元层和网络层业务匹配。网元上已经使能SRTE功能，并下发了隧道相关配置，通过业务还原将网元上SRTE业务配置还原到开放可编程系统数据库中。

步骤1 创建SSP模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表7-14填写软件包的相关属性值。

表 7-14 业务还原软件包属性

属性名称	属性值
名称	SSP-SRTE
版本	4.0.30
提供商	huawei
包类型	Specific Service plugin
服务名称	cmnetSrte
映射类型	python+template

版权所有 © 华为技术有限公司

属性名称	属性值
业务定义点	neTunnels

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
  "name": "SSP-SRTE",
  "version": "4.0.30",
  "description": "srte for cmnet",
  "package-type": "ssp",
  "producer": "huawei",
  "service-name": "cmnetSrte",
  "group": "cmnet",
  "nce-min-versions": [
    "1.0.0",
    "2.0.0"
  ],
  "hooks": [
    {
      "type": "mapping",
      "key": "neTunnels",
      "python-class-name": "com.huawei.cmnet.srte.aoc_ncs_srte.AocNcsSrteService"
    },
    {
      "type": "discover",
      "key": "neTunnels",
      "python-class-name": "com.huawei.cmnet.srte.srte_restore.SrteRestoreService"
    }
  ]
}
```

步骤3 开发业务YANG模型

开发YANG模型时，分析梳理出配置下发涉及的参数，如表7-15所示，并在YANG模型中进行定义。

表 7-15 整理的输入参数

类别	描述	参数名称	参数值
SRTE隧道配置	创建隧道接口，配置隧道参数与协议。 • deviceName标识SRTE业务下发到哪台设备。 • tunnelName与tunnel-id为SRTE隧道标识。description是该隧道的描述信息。 • bandwidth标识隧道带宽信息。 • setup-priority标识隧道建立优先级。	deviceName	PE1
		signal-protocol	segment-routing
		tunnel-id	4000
		destIp	10.10.110.114
		tunnelName	Tunnel4000
		description	SRTETunnel
		statistic	true
		reserved-for-binding	true

 版权所有 © 华为技术有限公司

类别	描述	参数名称	参数值
		bandwidth	ct0 1
		setup-priority	0
		pce-delegate	active
		hotstandby	hotstandby
隧道接口配置	• mtu标识该隧道接口的网络分片配置。 • ip address标识该接口的ip信息，地址的输入形式分为两种，一种使用实际IP地址加子网信息进行配置；另一种使用unnumbered 的形式进行配置，直接将隧道与接口绑定完成配置；本例中使用unnumbered 的方法，直接将本隧道与interface Tunnel2009 绑定。	mtu	46
		ip-address	unnumbered interface Tunnel2009
Mpls bfd配置	• ability标识是否开启bfd功能。 • mode标识bfd的模式。 • min-tx-interval标识最小发送间隔。 • min-rx-interval标识最小接受间隔。	ability	enable
		mode	seamless
		min-tx-interval	3
		min-rx-interval	3
lgp配置	• shortcut-type标识设置OSPF在进行增强的SPF计算时，使用本隧道。 • igpMetricValue标识在IGP Shortcut的SPF计算中，本隧道的度量值。	shortcut-type	3
		igpMetricValue	3
隧道路径配置	• hop-limit标识该隧道路径跳转次数的限制。 • explicit-path-name标识该隧道路径的名称。 • path-type标识该隧道路径的主备模式。	hop-limit	20
		explicit-path-name	expath_0110/ expath_0111
		path-type	expath_0111 secondary

最终编辑完成的业务还原软件包YANG模型如下。

```
module cmnetSrte {
    ......
   grouping ne-tuls {
      list neTunnel {
      app:application-definition "neTunnels";
```

 版权所有 © 华为技术有限公司

```
            key "deviceId tunnelName";
            uses srte-tunnel;
        }
    }

    augment "/app:applications" {
        uses ne-tuls;
}

     // 设置SRTE隧道配置
     grouping srte-tunnel {
        leaf deviceId {
            type string {
                length "1..max";
            }
            description "device id";
        }
        leaf tunnelName {
            type string {
                length "1..255";
            }
            mandatory true;
        }
        leaf tunnelId {
            type uint32 {
                range "1..65535";
            }
        }
        leaf destIp {
            type string;
            mandatory true;
        }
        leaf ipAddressType {
            type ip-types;
        }
        leaf description {
            type string {
                length "0..1024";
            }
        }
        leaf controlMode {
            type control-modes;
            mandatory true;
        }
...
```

步骤4 开发代码

实现Discover功能，需要继承NcsService并重写Discover方法，实现业务的逻辑。

业务还原软件包代码示例如下所示。

```
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
# devicemgr提供了一些查询设备信息的接口。
from aoc import NcsService, devicemgr
# IPv4Network提供了IP地址是否公网、转换二进制等接口。
from ipaddress import IPv4Network
from aoc.ncs.filter import ipaddr


class SrteRestoreService (NcsService):
    def discover(self, discover_input, aoc_context):
        """
        设备发现函数
        :param discover_input: 设备发现入参
        :param aoc_context: 上下文
        :return:转换后的xml报文
        """
        time_begin = time.time()
        self.logger.info("[SrTeRestoreService][discover] discoverInput is: %s,"
                         " aoc_context is: %s" % (discover_input, aoc_context))
```

 版权所有 © 华为技术有限公司

```
    # 从discover_input中获取设备信息
    device_info_list = []
    if discover_input.deviceInfo is not None:
        device_info_list = discover_input.deviceInfo
    self.logger.info("[SrTeRestoreService][discover] device_info_list is: %s"
                     % device_info_list)

    # 读取上下文信息 接着上一轮处理的进度继续处理
    discovery_id = discover_input.discoveryId
    self.cache_context = discover_input.cacheContext
    self.logger.info("start real discovery: %s---%s" % (discovery_id, time.time()))

    # 介绍一下DiscoverOutput
    result = DiscoverOutput()
    # 遍历输入信息列表中的设备信息
    for device_info in device_info_list:
        device_id = device_info.deviceId

        self.logger.info("[SrTeRestoreService][discover] device_info.attributes is: %s" %
device_info.attributes)
        # 判断设备中需要处理的业务个数，如果大于0小于等于最大值，则将储存的最大值刷新
        if device_info.attributes and self.handler_max_size >= int(device_info.attributes):
            self.handler_max_size = int(device_info.attributes)

        # 读取设备RDB,查询对应设备ifm的总数
        self.read_ifm_count_from_rdb(aoc_context, device_id)
        # get the template by device info
        template = self.find_ne_template(device_id)

        if template is None:
            raise ValueError("The device doesn't support [deviceId: %s]" % device_id)

        # 读取设备RDB,查询对应设备ifm，每次查询100条
        ifm_dicts = self.read_ifm_from_rdb(aoc_context, device_id)
        # build ne template
        if ifm_dicts and ifm_dicts['interface']:
            # 处理SrTe隧道模板内容，并返回模板内容集合
            ifm_list = self.handler_template_content(ifm_dicts, aoc_context, device_id)
            self.logger.info("[SrTeRestoreService][discover] ifm_list is %s" % ifm_list)

            # 组建需要还原的数据
            sr_tes_restore_template_content = {
                "ifm_list": ifm_list,
                "ne_id": device_id
            }

            # 根据对应网元的模板对数据进行反向映射完成业务还原，template为业务还原的jinja
            tunnels = DictNode(xmltodict.parse)
                self.render(template, sr_tes_restore_template_content),
                encoding="utf-8", strip_whitespace=True))
            self.logger.info("[SrTeRestoreService][discover] tunnels is %s" % tunnels)
            if tunnels['items']:
                # 处理返回给AOC的响应response
                self.handler_response(tunnels, device_id, result)

        # 更新上下文信息，为下一轮业务还原做标记。若记录个数超过上限或数据处理完则结束整个流程。
        if int(self.cache_context["offset"]) < int(self.cache_context["ifm_count"]):
            result.goOn = True
        for key in self.cache_context:
            self.logger.debug("interface %s" % self.cache_context[key])
            result.cacheContext[key] = self.cache_context[key]

    self.logger.info("[SrTeRestoreService][discover] result %s" % result)
    time_end = time.time()
    self.logger.info("[SrTeRestoreService][discover] cost time is %s seconds" % (time_end - time_begin))
    return result
```

 版权所有 © 华为技术有限公司

```
    # 代码已经自适应全量业务还原与增量业务还原，无需其他编码
    def handler_template_content(self, ifm_dicts, aoc_context, device_id):
    """
    处理SrTe隧道模板内容，并返回模板内容集合
    :param ifm_dicts: ifm_dicts.
    :param cache_context: discover上下文.
    :param aoc_context: 上下文.
    :param device_id: 设备id.
    :return: ifm_list
    """
    ifm_list = []
    # 查询network-instance
    ipv4s_and_ipv6s = self.read_ipv4_and_ipv6_from_data_store(device_id, aoc_context)
    ipv4s = ipv4s_and_ipv6s['ipv4s']
    ipv6s = ipv4s_and_ipv6s['ipv6s']
    # 通过device_id查询AOC页面中已经存在的隧道
    ifm_name_dict = self.valid_ifm_exist(aoc_context, device_id)
    for ifm in ifm_dicts['interface']:
        self.logger.info("[SrTeRestoreService][handler_template_content]:1")
        # signal-protocol为segment-routing时是SR-TE隧道
        if self.is_segment_routing(ifm):
            self.logger.info("[SrTeRestoreService][handler_template_content]:is_segment_routing")
            # 如果在AOC页面中已经存在则不需要再进行还原
            if ifm_name_dict and ifm['name'] in ifm_name_dict:
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:in ifm_name_dict")
            # 根据cacheContext判断接口是否已经还原
            if self.cache_context is not None and self.cache_context[ifm['name']] is not None and \
                    self.cache_context[ifm.name] == 'exist':
                self.cache_context[ifm['name']] = 'exist'
                continue
            self.logger.info("[SrTeRestoreService][handler_template_content]:not exist")
            # 解析RDB结果
            if ifm['name'] in ipv4s:
                ifm['ipv4'] = ipv4s[ifm['name']]
            if ifm['name'] in ipv6s:
                ifm['ipv6'] = ipv6s[ifm['name']]

            # 更新上下文中ifm信息 记录其已经被还原
            self.cache_context[ifm['name']] = 'exist'
            self.logger.info("[SrTeRestoreService][handler_template_content] ifm:%s" % ifm['name'])
            sr_te_restore_template_content = {
                "interface": ifm
            }
            self.init_size = self.init_size + 1
            # 查看是否超出单次处理的最大次数
            if self.init_size > self.handler_max_size:
                break
            ifm_list.append(sr_te_restore_template_content)
    return ifm_list
```

步骤5 开发Jinja2模板

把编辑的内容放入到模板文件“template>cmnetSrte.j2”中。

```
{# 嵌套items标签，方便多个业务配置同时渲染 #}
<items>
{%- for node in  ifm_list %}
<item>
{# 标签的数据结构请保持与导出的业务配置模板一致 #}
<neTunnel xmlns="urn:icmcc:params:xml:ns:yang:cmcc-ip-srte">
    <deviceId>{{ ne_id }}</deviceId>
    {%- set interface = node.interface %}
{# 增加判断语句，对数据进行逻辑处理 #}
{# 变量来自于网元配置数据，所以可参考配置下发Jinja模板或设备配置界面导出模板中的数据结构设置 #}
{%- if interface['name'] %}
    <tunnelName>{{ interface['name']}}</tunnelName>
    {%- if interface['tunnel-protocol'] %}
    {%- if interface['tunnel-protocol']['te-tunnel'] %}
    {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr'] %}
```

 版权所有 © 华为技术有限公司

```
    <igpAttr>
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['hold-time'] %}
      <advertiseHoldTime>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr'][ 'hold-time'] }}</
advertiseHoldTime>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['relative'] %}
      <igpMetricType>relative</igpMetricType>
      <igpMetricValue>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['relative'] }}</igpMetricValue>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['absolute'] %}
      <igpMetricType>absolute</igpMetricType>
      <igpMetricValue>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['absolute'] }}</igpMetricValue>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['shortcut-type'] %}
      <shortcutType>{{ interface['tunnel-protocol']['te-tunnel']['igp-attr']['shortcut-type'] }}</shortcutType>
      {%- endif %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['adv-enable'] %}
      {%- if interface['tunnel-protocol']['te-tunnel']['igp-attr']['adv-enable'] == 'false' %}
      <enableAdvertise>disable</enableAdvertise>
      {%- else %}
      <enableAdvertise>enable</enableAdvertise>
      {%- endif %}
      {%- endif %}
    </igpAttr>
    {%- endif %}
    {%- endif %}
    {%- endif %}
...
```

----结束

7.3.5 开发查询设备信息 RPC SSP 示例

本示例介绍如何通过下载的默认SSP包，开发一个RPC SSP包，用户自定义查询设备信息，获取设备报文。

步骤1 创建SSP模板

1. 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。
2. 并在弹出的对话框中设置软件包的相关属性。

   请根据表7-16填写软件包的相关属性值。

表 7-16 查询设备信息 RPC 软件包属性

属性名称	属性值
名称	RPC
版本	1.0.0
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	RPC
业务定义点	RPC
映射类型	python+template

版权所有 © 华为技术有限公司

3. 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。
4. 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

步骤2 编辑软件包配置文件

```
{
   "name": "RPC",
   "version": "1.0.15",
   "description": "This is a RPC service",
   "package-type": "ssp",
   "service-name":"RPC",
   "group": "xxxx",
   "producer": "Huawei",
   "nce-min-versions":[

 ],

   "hooks": [
    {
      "type": "service-rpc",
      "key": "(http://example.com/RPC?revision=2018-12-09)read_device_vrf",
      "python-class-name": "RPCSSP.RpcService.AocNcsAaaService"
    }
   ]
}
```

步骤3 开发业务YANG模型

这里定义设备接口状态查询功能，定义输入device-id、interface ，定义输出interface、ifV4State。

最终编辑完成的YANG模型如下。

```
module RpcService {
   namespace "http://example.com/RPC";
   prefix "RPC";

   import huawei-ac-applications {
      prefix app;
   }
   description
      "The module for RpcService example.";

   revision 2018-12-09 {
      description "Initial revision.";
   }

   augment "/app:applications"{
      list RPC {
         app:application-definition "aaa";
         key "instance_name";
         leaf instance_name {
            type string;
         }
      }
   }
   //定义输入：device-id、interface
   rpc read_device_vrf {
   input {
         leaf device-id {
            mandatory true;
            type string;
         }
         container interfaces {
            list interface {
               key name;
               leaf name {
                  type string {
```

 版权所有 © 华为技术有限公司

```
                    length "1..max";
                  }
                }
              }
            }
          }
          //定义输出：interface、ifV4State
          output {
            container interfaces {
              list interface {
                key name;
                leaf name {
                  type string {
                    length "1..max";
                  }
                }
                leaf ifV4State {
                  type string;
                }
              }
            }
          }
        }
      }
```

步骤4 开发代码

当用户自定义RPC功能时，需要继承NcsService并重写ncs_rpc方法，按照业务需求实现自定义功能的逻辑，最终返回处理结果。

RPC软件包代码示例如下所示。

```
# 必选，导入NcsService类，NcsService是aoc_api提供的供SSP包继承的父类。devicemgr可用于从设备上实时查询数据。
from aoc import NcsService, devicemgr


class AocNcsAaaService(NcsService):
    # 查询srte状态
    INTERFACE_PATH = "/huawei-ifm:ifm"

    # 设置设备查询路径
    PROPERTIES_SUBTREE = """
            <interfaces>
              <interface>
                <ifDynamicInfo>
                  <ifV4State/>
                </ifDynamicInfo>
              </interface>
            </interfaces>
           """

    RESULT_TEMPALTE = """
           <read_device_vrf xmlns="http://example.com/RPC">
              <interfaces>
                 %s
              </interfaces>
           </read_device_vrf>
           """

    RESULT_VRF_ONE_TEMPALTE = """
                  <interface>
                     <name>%s</name>
                     <ifV4State>%s</ifV4State>
                  </interface>
           """

    # 设备RPC
    def ncs_rpc(self, request, context=None, template=None):
```

 版权所有 © 华为技术有限公司

```
    # 获取输入
    request_xml = self.xmltodictnode(request.rpcXML.strip())

    # 获取设备ID
    device_id = request_xml.get("read_device_vrf").get("device-id")
    srte_list = request_xml.get("read_device_vrf").get("interfaces").get("interface")
    if isinstance(srte_list, dict):
        srte_list = [srte_list]

    # 根据设备ID查询设备信息
    output = self.query_device_by_device_id(device_id)

    # 查询返回值
    result = self.result_Construction(output, srte_list)
    return result


# 构造返回体
def result_Construction(self, output, srte_list):

    # 获取设备上接口的状态
    all_ifm_status_up = self.filter_ifm_status(output)

    # 构造返回体，返回设置的接口状态
    srte_all_status = ""
    for stre in srte_list:
        name = stre.get("name")
        status = 0
        if name in all_ifm_status_up:
            status = all_ifm_status_up[name]
        srte_all_status = srte_all_status + (self.RESULT_VRF_ONE_TEMPALTE % (name, status))
    self.logger.info("[srte-status] srtes status is %s", srte_all_status)
    return (self.RESULT_TEMPALTE % srte_all_status).strip()


# 设置接口状态
def filter_ifm_status(self, output):
    all_ifm_status_up = {}

    # 获取接口的状态
    if output.result is not None:
        ifm_dict = self.xmltodictnode(output.result.strip())
        if ifm_dict.get("ifm") and ifm_dict.get("ifm").get("interfaces"):
            interface_dict = ifm_dict.get("ifm").get("interfaces").get("interface")
            for interface in interface_dict:
                ifm_name = interface.get("ifName")
                ifV4State = interface.get("ifDynamicInfo").get("ifV4State")
                all_ifm_status_up[ifm_name] = ifV4State
    return all_ifm_status_up


# 根据设备ID查询设备信息
def query_device_by_device_id(self, device_id):

    # 查询设备信息，并从设备中获取报文
    properties = {
        "subtree": self.PROPERTIES_SUBTREE.strip()
    }
    output = devicemgr.query_data_from_device(device_id, 'OPERATIONAL',
                                              self.INTERFACE_PATH, properties, 60000)
    return output
```

步骤5 开发Jinja2模板

代码中没有调用render渲染，即没有将北向报文传到jinja2模板做转换，所以就不需要开发Jinja2模板了。

----结束

 版权所有 © 华为技术有限公司

8 开发告警插件包

开放可编程提供告警处理能力，用户可以自定义告警插件包或在对应软件包中编写告警处理逻辑。

目前主要用于的场景：

- 设备告警信息自定义处理：用户获取设备告警，进行自定义处理，然后发送到其他系统，一般通过GND包处理。
- 自定义AOC告警并上报：用户将AOC中的异常捕获并支持上报自定义告警到OSS系统，一般通过SSP包处理。

8.1 创建业务包模板

8.2 开发业务包

8.3 编译业务包

8.4 验证告警插件包

8.1 创建业务包模板

步骤1 在主菜单中选择“包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的对话框中设置软件包的相关属性。

请根据**表8-1**或**表8-2**填写软件包的相关属性值。

表 8-1 GND 告警插件包属性

属性名称	属性值
名称	alarm
版本	1.0.0
提供商	HUAWEI
包类型	Generic NE Driver
服务名称	alarm

 版权所有 © 华为技术有限公司

属性名称	属性值
业务定义点	alarm
映射类型	python

表 8-2 SSP 告警插件包属性

属性名称	属性值
名称	alarm
版本	1.0.0
提供商	HUAWEI
包类型	Specific Service plugin
服务名称	alarm
业务定义点	alarm
映射类型	python

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

----结束

8.2 开发业务包

8.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

版权所有 © 华为技术有限公司

表 8-3 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：Specific Service plugin、Generic NE driver、Specific NE driver、SND Migrate plugin。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。	如果配置，必须包含： 三方包name（必选）：依赖的软件包名，长度不能超过64个字符。 version（必选）字段：版本号三段式规则。	可选
service-name	ssp包特有值。服务名称。	自定义，不能超过 64个字符。 ssp包时，必须设置，不同名的包不能使用相同的service-name。 其他类型软件包时，不需要设置。	可选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。EasyMap，type设置为mapping。RPC，type设置为service-rpc。Discover，type设置为discover。告警包功能设置为service-alarm。如需要支持填充在YANG模型中定义参数默认值，type设置为service_settings。data-with-default可以设置为all或explicit，all表示填充定义的参数默认值，explicit表示默认不填充定义的参数默认值。 ● key（必选），回调key值，不能超过128个字符。一般情况下，YANG模型中的“app:application-definition”的值对应key值，即业务定义点。type设置为service-rpc时，key值规则为“（namespace:revision)localname”，localname指YANG模型rpc节点名称，例如"(http://example.com/RPC?revision=2018-12-09)read_device_vrf"。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

检查文件内容，修改属性设置，一般只需修改“type”、"key"、“python-class-name”，参数含义和修改方法请参见上述表格。最终编辑文件示例如下：

```
{
   "name": "alarm",
   "version": "1.0.0",
   "description": "This is a alarm service",
   "package-type": "ssp",
   "service-name":"alarm",
   "group": "xxxx",
   "producer": "Huawei",
   "nce-min-versions":[
```

 版权所有 © 华为技术有限公司

```
 ],

  "hooks": [
    {
      "type": "service-alarm",
      "key": "(http://example.com/alarm?revision=2018-12-09)read_device_vrf",
      "python-class-name": "alarm.com.huawei.alarmservice.AlarmTest"
    }
  ]
}
```

----结束

8.2.2 开发业务包代码

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码，实现告警功能逻辑。

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

设备告警信息自定义处理

用户可以在GND包代码中定义，获取设备告警，告警信息处理，告警通知南北向 ，以及接收告警事件处理。示例如下：

```
# 导入AlarmReceiver，用来重写trap_handle方法
from aoc.gnd.alarm_receiver import AlarmReceiver
# 导入AlarmMgr，调用write_alarm方法
from aoc.sys.alarm_mgr import AlarmMgr
# 导入Snmp，调用get接口
from aoc.sys.snmp import Snmp
# 导入EventMgr，调用send_aoc_event接口
from aoc.sys.event_mgr import EventMgr
# 导入AocEventEntity，AocAlarmEntity，调用alarmEntity接口
from aoc.sys.sys_model_pb2.AocEventHandler_pb2 import AocEventEntity
from aoc.gnd.gnd_model_pb2.AocAlarm_pb2 import AocAlarmEntity


class AocAlarmReciver(AlarmReceiver):
    # snmp trap的定制处理，告警获取，告警信息处理和发送，事件信息发送
    def trap_handle(cls,aoccontex, trap_entity_list):
        cls.logger.info("trapHandle start !!! traps = [{}]".format(trap_entity_list))
        AocAlarmReciver.test_sync_snmp(cls)
        if isinstance(trap_entity_list, list):
            for snmp_tramp in trap_entity_list:
                if (snmp_tramp.trapOidEntity.oid == '1.3.6.1.4.1.2011.6.122.20.2.2.15'):
                    AocAlarmReciver.test_write_alarm(cls)
        AocAlarmReciver.test_add_event(cls)

    # 获取snmp告警
    @staticmethod
    def test_sync_snmp(cls):
        cls.logger.info("syncSnmpAlarm start !!!")
        try:
            # 获取告警信息，返回告警信息：参考aoc.sys.AocSyncSnmp.AocSyncSnmpOutput和
proto.aoc.gnd.AocSnmpTrap.AocSnmpTrapsEntity
            result = Snmp.get('eda6cc3d-2a6a-3b08-8bcc-575646f5fc14', ['1.3.6.1.2.1.31.1.1.1.1'])
            cls.logger.info("syncSnmpAlarm result =[{}]".format(result))
        except BaseException:
            cls.logger.exception('syncSnmpAlarm, Exception occur. %s' % e)
        finally:
            cls.logger.info("test_sync_snmp end !!!!")
```

 版权所有 © 华为技术有限公司

```
    # snmp告警处理
    @staticmethod
    def test_write_alarm(cls):
        try:
            cls.logger.info("test_write_alarm start !!!!")
            # 告警信息数据结构：aoc.sys.AocAlarm.AocAlarmEntity
            alarmEntity = AocAlarmEntity()
            alarmEntity.category = 1
            alarmEntity.clearTime = 1574939834460
            alarmEntity.occurUtc = 1574939834460
            alarmEntity.mergeKey="oRrHyWLBOgXWS+r/0btj0A"
            alarmEntity.nativeMeDn ="OS=1"
            alarmEntity.meDn = "10"
            alarmEntity.nativeMoDn="ID=68d02593-9dee-351b-9261-87729c295e5c"
            alarmEntity.moc="DEVICE"
            alarmEntity.alarmGroupId= "100000"
            alarmEntity.originSystemId="1"
            alarmEntity.originSystem = "OSS"
            alarmEntity.originSystemName="OSS"
            alarmEntity.tenant="NAAS"
            alarmEntity.tenantId="00000000-0000-0000-0000-000000000000"
            alarmEntity.moi="Name=DTC"
            alarmEntity.eventType = 2
            alarmEntity.alarmId="15794439"
            alarmEntity.alarmName="\350\256\276\345\244\207\347\212\266\346\200\201Down"
            alarmEntity.probableCause="1.
\346\216\247\345\210\266\345\231\250\345\215\227\345\220\221\344\270\232" \
                                      "\345\212\241\347\275\221\345\215" \
                                      "\241\351\200\232\344\277\241\345\274\202\345" \
                                      "\270\270\343\200\202\n2. \350\256\276\345\244" \
                                      "\207\346\225\205\351\232\234\343\200\202\n3. " \
                                      "\350\256\276\345\244\207netconf\351\200\232\351" \
                                      "\201\223\347\251\272\351\227\262" \
                                      "\346\243\200\346\265\213\346\234\252\345\205\263\351" \
                                      "\227\255\343\200\202"
            alarmEntity.resPoolId="0"

            # 告警发送接口
            AlarmMgr.write_alarm(alarmEntity)
        except BaseException:
            cls.logger.exception('test_write_alarm, Exception occur. %s' % e)
        finally:
            cls.logger.info("test_write_alarm end !!!!")


    # 事件信息发送网元层
    @staticmethod
    def test_add_event(cls):
        cls.logger.info("test_add_event start !!!!")
        try:
            cls.logger.info("test_add_event start !!!!")
            # 事件信息接口，结构定义在aoc.sys.AocEventHandler.AocEventEntity
            aocEvent = AocEventEntity()
            aocEvent.eventType = "test_python_type"
            aocEvent.eventTopic = "test_python_topic"
            aocEvent.message = "test_python_message"
            EventMgr.send_aoc_event(aocEvent)
        except BaseException:
            cls.logger.exception('test_add_event, Exception occur. %s' % e)
        finally:
            cls.logger.info("test_add_event end !!!!")
```

自定义 AOC 告警并上报

用户可以在SSP包代码中定义，包激活时告警注册和告警多语言信息注册，包卸载时告警去注册和告警多语言信息去注册，AOC异常捕获并自定义告警。

```
# NcsService是系统提供的供SSP包继承的父类，需要重写以实现业务YANG数据到Jinja2模板的映射；
from aoc.ncs.ncsservice import NcsService
```

 版权所有 © 华为技术有限公司

```
# 导入AlarmMgr，调用write_alarm方法
from aoc.sys.alarm_mgr import AlarmMgr
# 导入AocAlarmEntity，调用alarmEntity接口
from aoc.gnd.gnd_model_pb2.AocAlarm_pb2 import AocAlarmEntity
# 导入alarm_entity，调用alarm相关接口
from aoc.sys.alarm_entity import AocQueryAlarmDefinitionInput
from aoc.sys.alarm_entity import AocAlarmDefinitionEntity
from aoc.sys.alarm_entity import I18nResourceEntity
from aoc.sys.alarm_entity import I18nResourcePakageEntity
from aoc.sys.alarm_entity import I18nResourceDataEntity
from aoc.sys.alarm_entity import DeleteI18nResourceEntity


class AlarmTest(NcsService):
    def setup(self):
        """
        软件包激活初始化接口，可以调用上报告警定义信息接口和新增多语种告警资料信息
        :return:
        """
        self.logger.info("alarm test 1.0.2 setup")

        # 告警定义注册和告警多语言信息注册
       self.test_add_alarm_definitions()
        self.test_add_i18n_resources()

    def teardown(self):
        """
        软件包卸载销毁接口，可以调用删除告警定义信息接口和删除多语种告警资料信息
        :return:
        """
        self.logger.info("alarm test 1.0.2 teardown")

        # 告警定义去注册和告警多语言信息去注册
       self.test_delete_alarm_definitions()
        self.test_delete_i18n_resources()

    def test_add_alarm_definitions(self):
        """
        上报告警定义信息接口,用户自定义的告警信息
        :return:
        """
        self.logger.info("test_add_alarm_definitions start")
        alarm_definition_list = list()
        alarm_definition_data = AocAlarmDefinitionEntity()
        alarm_definition_data.alarmId = '42'
        alarm_definition_data.originSystemType = 'alarm'
        alarm_definition_data.alarmGroupId = 'u2000'
        alarm_definition_data.category = 1
        alarm_definition_data.eventType = 1
        alarm_definition_data.alarmName = 'alarmTest'
        alarm_definition_data.alarmGroupName = 'alarmGroupTest'
        alarm_definition_data.description = 'add alarm definitions test'
        alarm_definition_data.severity = 1
        alarm_definition_data.reasonType = 1
        alarm_definition_data.relativeAlmId = 1
        alarm_definition_data.clearStrategy = 1
        alarm_definition_list.append(alarm_definition_data)
        alarm_mgr = AlarmMgr(self.logger)
        alarm_mgr.add_alarm_definitions(alarm_definition_list)
        self.logger.info("test_add_alarm_definitions end")

    def test_delete_alarm_definitions(self):
        """
        删除告警定义信息
        :return:
        """
        self.logger.info("test_delete_alarm_definitions start")
        alarm_group_id = 'u2000'
        alarm_id = '42'
```

 版权所有 © 华为技术有限公司

```
    alarm_mgr = AlarmMgr(self.logger)
    alarm_mgr.delete_alarm_definitions(alarm_group_id, alarm_id)
    self.logger.info("test_delete_alarm_definitions end")

def test_add_i18n_resources(self):
    """
    新增多语种告警资料信息
    field：
    alarmName：告警名称
    probableCause：可能原因
    productName：告警源类型
    causes：产生原因
    repairAction：修复建议
    :return:
    """
    self.logger.info("test_add_i18n_resources start")
    i18n_resources_list = list()
    i18n_resources_data = I18nResourceDataEntity()
    i18n_resources_data.field = 'productName'
    # i18nResourcePakage: list of I18nResourcePakageEntity
    i18n_resource_pakage_list = []
    i18n_resource_pakage = I18nResourcePakageEntity()
    i18n_resource_pakage.lang = 'ru_RU'
    # i18nResource: list of I18nResourceEntity
    i18n_resource_list = []
    i18n_resource = I18nResourceEntity()
    i18n_resource.key = 'cloudsop.fm.alarmrecord.productname.100000'
    i18n_resource.value = 'abc'
    i18n_resource_list.append(i18n_resource)
    i18n_resource_pakage.i18nResource = i18n_resource_list
    i18n_resource_pakage_list.append(i18n_resource_pakage)
    i18n_resources_data.i18nResourcePakage = i18n_resource_pakage_list
    i18n_resources_list.append(i18n_resources_data)
    alarm_mgr = AlarmMgr(self.logger)
    alarm_definitions = alarm_mgr.add_i18n_resources(i18n_resources_list)
    self.logger.info("alarm_definitions = %s" % alarm_definitions)
    self.logger.info("test_add_i18n_resources end")

def test_delete_i18n_resources(self):
    """
    删除多语种告警资料信息
    :return:
    """
    self.logger.info("test_delete_i18n_resources start")
    delete_i18n_resources_list = list()
    delete_i18n_resources_data = DeleteI18nResourceEntity()
    delete_i18n_resources_data.lang = 'ru_RU'
    delete_i18n_resources_data.field = 'productName'
    delete_i18n_resources_data.key = 'cloudsop.fm.alarmrecord.productname.100000'
    delete_i18n_resources_list.append(delete_i18n_resources_data)
    alarm_mgr = AlarmMgr(self.logger)
    alarm_mgr.delete_i18n_resources(delete_i18n_resources_list)
    self.logger.info("test_delete_i18n_resources end")

def test_write_alarm(self):
    """
    告警上报，告警信息用户自己决定
    :return:
    """
    try:
        self.logger.info("test_write_alarm start !!!!")
        alarmEntity = AocAlarmEntity()
        alarmEntity.category = 1
        alarmEntity.clearTime = 1574939834460
        alarmEntity.occurUtc = 1574939834460
        alarmEntity.mergeKey="oRrHyWLBOgXWS+r/0btj0A"
        alarmEntity.nativeMeDn ="OS=1"
        alarmEntity.meDn = "10"
        alarmEntity.nativeMoDn="ID=fe8f28a3-30ee-3f83-a484-1bcb2b46d13d"
```

 版权所有 © 华为技术有限公司

```
            alarmEntity.moc="DEVICE"
            alarmEntity.alarmGroupId= "u2000"
            alarmEntity.originSystemId="1"
            alarmEntity.originSystem = "OSS"
            alarmEntity.originSystemName="OSS"
            alarmEntity.tenant="NAAS"
            alarmEntity.tenantId="00000000-0000-0000-0000-000000000000"
            alarmEntity.moi="Name=DTC"
            alarmEntity.eventType = 1
            alarmEntity.alarmId="42"
            alarmEntity.alarmName="alarmTest"
            alarmEntity.probableCause="1.
\346\216\247\345\210\266\345\231\250\345\215\227\345\220\221\344\270\232"
\"\345\212\241\347\275\221\345\215" \"\241\351\200\232\344\277\241\345\274\202\345"
\"\270\270\343\200\202\n2. \350\256\276\345\244" \"\207\346\225\205\351\232\234\343\200\202\n3. "
\"\350\256\276\345\244\207netconf\351\200\232\351" \"\201\223\347\251\272\351\227\262"
\"\346\243\200\346\265\213\346\234\252\345\205\263\351" \"\227\255\343\200\202"
            alarmEntity.resPoolId="0"
            alarm_mgr = AlarmMgr(self.logger)
            alarm_mgr .write_alarm(alarmEntity)
        except BaseException:
            self.logger.exception('test_write_alarm, Exception occur.')
        finally:
            self.logger.info("test_write_alarm end !!!!")


    def ncs_map(self, request, aoccontext=None, template=None):
        self.logger.info('%s' % request)
        try:
            ncs_context = dict(request.context)
            self.logger.info('%s' % ncs_context)
            if not aoccontext or not aoccontext.neid
                raise Exception("neid is None!!!")
        except Exception as e:
            self.logger.info("%s" % traceback.format_exc())
            # 捕获异常，写入异常告警
            self.test_write_alarm()

        return "<inventory-cfg xmlns=\"urn:huawei:yang:huawei-ac-nes\"><nes></nes></inventory-cfg>",
ncs_context
```

8.3 编译业务包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

步骤2 在命令行中运行如下命令，将导出的私钥文件移动到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

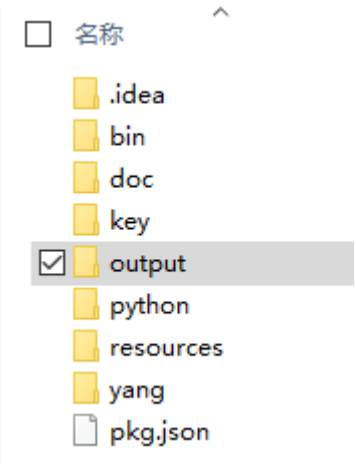
在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。

 版权所有 © 华为技术有限公司




```
2021-11-04 18:10:12,291 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Generate signature file success.
2021-11-04 18:10:12,542 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

8.4 验证告警插件包

步骤1 在环境上导入并安装软件包。

步骤2 通过触发设备告警或AOC异常，可以在查看告警界面看到上报的告警，说明测试成功。

----结束

 版权所有 © 华为技术有限公司

9 开发迁移插件包

SND不兼容升级/降级时，迁移前后软件包中的YANG模型发生的变更存在不兼容的项，需要开发SND迁移插件包对其进行处理。

9.1 创建迁移包模板

9.2 开发业务包

9.3 编译业务包

9.1 创建迁移包模板

步骤1 在主菜单中选择“包仓库”，从左侧导航栏选择进入“包管理”界面，单击“创建模板”。

步骤2 并在弹出的“增加”对话框中设置软件包的相关属性。

请根据**表9-1**填写软件包的相关属性值。

表 9-1 SMP 属性

属性名称	属性值
名称	Migrate_python
版本	1.0.12
提供商	HUAWEI
包类型	Snd Migrate Plugin
映射类型	python
设备类型	NE40E-X3A(V8)

步骤3 单击“确定”后，在弹出的提示添加成功的对话框中再次单击“确定”，即可在软件包列表中看到新添加的软件包。

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。

 版权所有 © 华为技术有限公司

----结束

9.2 开发业务包

9.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。如果已创建工程，则直接打开PyCharm工程即可。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

表 9-2 pkg 文件参数表

参数名称	含义	取值	必选/可选
name	软件包名称。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选
version	软件包版本号。	仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	软件包描述信息。	自定义，长度不能超过128个字符。	可选
package-type	包类型。	枚举值：Specific Service plugin、Generic NE driver、Specific NE driver、SND Migrate plugin。	必选
group	包分组信息，主要用作沙箱进程隔离。	自定义，长度不能超过128个字符。不填写则取默认值global。	可选
producer	软件包的提供商。	自定义，只能包含，数字、大小写字母，下划线(“_”)，连字符(“-”)。1到 64个字符。	必选

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
nce-min-versions	兼容的开放可编程系统最小版本信息。	版本号三段式规则。高位版本号发生变更，代表发生不兼容变更。支持最小版本匹配，例如1.0.0 表示兼容[1.0.0,2.0.0)左闭右开区间的所有版本。兼容多个版本的区间段，可写多行。不填表示兼容所有版本。	可选
from-devices	迁移前SND包的devices信息。	如果配置，则必须包含： 驱动设备厂商vendor（必选），不能超过 128个字符。 具体设备类型device-type（必选），不能超过 128个字符。 具体设备的版本device-version（必选），不能超过 128个字符。	可选
to-devices	迁移后SND包的devices信息。	如果配置，则必须包含： 驱动设备厂商vendor（必选），不能超过 128个字符。 具体设备类型device-type（必选），不能超过 128个字符。 具体设备的版本device-version（必选），不能超过 128个字符。	可选
from-snd-id	迁移前SND包的snd-id值。	自定义，不能超过 128个字符。 需要填写迁移前SND包的snd-id值。 如果填写from-snd-id，则以from-snd-id为准，否则再识别from-devices信息。迁移时必须要填写迁移前后SND包的标识。	可选
to-snd-id	迁移后SND包的snd-id值。	自定义，不能超过 128个字符。 需要填写迁移后SND包的snd-id值。	可选

 版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
hooks	回调映射信息，用于将用户自定义代码注入到系统中。一个包里hook唯一，但在不同包里hook可以重复。多项功能组合时可以设置多个hooks。	如果配置，则需包含： ● type（必选），回调对应类型。SMP包设置为“migrate”。 ● key（必选），回调key值，不能超过128个字符。取值“change-value”，CHANGE_VALUE规则自动调用自定义接口。取值"migrate-custom"，COMPLEX规则自动调用自定义接口。 ● python-class-name（可选），Python开发语言特有值，回调映射类。不能超过 256个字符。取值为.py文件中实现类的相对路径（相对于python目录），py文件名称不需要带后缀，路径层级之间用“.”连接。如python\temp\test.py定义实现类Test，则值为“temp.test.Test”。 ● template（可选），模板名。不能超过 256个字符。	可选

SMP迁移包示例如下，用户一般需要准确填写"type"、"key"、“python-class-name”、“from-devices”、"to-devices"等参数值。

```
{
  "name": "Migrate_python",
  "version": "1.0.12",
  "description": "Migrate pkg",
  "package-type": "mp",
  "producer": "HUAWEI",
  "nce-min-versions": [
    "2.0.0"
  ],
  "hooks": [
    {
      "type": "migrate",
      "key": "change-value",
      "python-class-name": "com.huawei.custommigrate.CustomMigrate"
    },
    {
      "type": "migrate",
      "key": "migrate-custom",
      "python-class-name": "com.huawei.custommigrate.CustomMigrate"
    }
  ],
  "from-devices":[{
    "vendor":"huawei",
    "device-type":"CX600-X2-M9",
    "device-version":"V800R021C10"
  },{
    "vendor":"HUAWEI",
    "device-type":"NE40E-X3A(V8)",
    "device-version":"V800R021C00"
  }],
```

 版权所有 © 华为技术有限公司

```
 "to-devices":[{
   "vendor":"huawei",
   "device-type":"CX600-X2-M9",
   "device-version":"V800R022C00"
 },{
   "vendor":"HUAWEI",
   "device-type":"NE40E-X3A(V8)",
   "device-version":"V800R022C00"
 }]
}
```

----结束

9.2.2 开发业务包代码

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

9.2.2.1 判断不兼容类型和处理规则

步骤1 使用工具比较迁移前后SND包中的YANG模型，识别是否存在不兼容点。

1. 使用不兼容工具incompatible-nodes-check识别不兼容点。在CMD窗口中执行如下命令。

   incompatible-nodes-check-*.*.*-jar-with-dependencies.jar 【旧yang目录】【新yang目录】【结果输出目录】

2. 当新的YANG模型节点名变更，或层级变化时，不兼容工具会停在此处。用户可修改新YANG模型，如将节点名改为一致，如在新YANG模型中添加缺少的container节点层级（注意其他节点augment的path需要同时修改）。
3. 继续执行不兼容工具命令，直到识别出所有的不兼容点。

步骤2 查看工具比较结果，对不兼容点进行初步识别，确认是否为真实不兼容点。

工具识别的不兼容点，业务场景可能是兼容的，比如int32 ->int8, 工具会识别不兼容，但如果业务数据原先就符合int8，直接迁移也没有问题。

步骤3 继续识别所有不兼容点，根据下表判断不兼容点属于哪种不兼容类型，对应哪种不兼容处理规则。

 版权所有 © 华为技术有限公司

表 9-3 规则表

规则	不兼容项目	举例	处理说明

KEY_CHANGE	key索引增加	旧YANG
		list {
		key “key1 key2”
		leaf key1
		leaf key2
		leaf key3
		leaf key4
		}
		新YANG
		list {
		key “key1 key2 key3”
		leaf key1
		leaf key2
		leaf key3
		}
			能根据原先key 值唯一找到迁移后的key 值就按简单不兼容场景处理，否则就按复杂不兼容场景处理

	key索引删除	旧YANG同上
		新YANG
		list {
		key “key1”
		leaf key1
		leaf key2
		}
			能根据原先key 值唯一找到迁移后的key 值就按简单不兼容场景处理，否则就按复杂不兼容场景处理

	key索引修改-修改为之前存在的节点	旧YANG同上
		新YANG
		list {
		key “key4 key2 key3”
		leaf key1
		leaf key2
		…
		}
			按简单不兼容场景处理，或按复杂不兼容场景处理

NODE_MAP	leaf节点改名	-	按简单不兼容场景处理
NODE_MAP	增加container路径	-	按简单不兼容场景处理

 版权所有 © 华为技术有限公司

规则	不兼容项目	举例	处理说明
CHANGE_VALUE	leaf节点值修改场景	-	按简单不兼容场景处理
SECURITY	leaf节点加密场景	-	按简单不兼容场景处理
RETAIN_VALUE	值不变场景（单位、取值范围等可能变化，但值不变）	-	按简单不兼容场景处理
REMOVE_LEVEL	删除container路径	-	按简单不兼容场景处理
ORDERED_BY_CHANGE	list节点ordered-by由user切system	-	按简单不兼容场景处理

COMPLEX	key索引修改-修改为之前不存在的节点	旧YANG
		list {
		key “key1 key2”
		leaf key1
		leaf key2
		leaf key3
		leaf key4
		}
		新YANG
		list {
		key “key1’ key2 key3”
		leaf key1’
		leaf key2
		}
			按复杂不兼容场景处理

	删除节点	-	按复杂不兼容场景处理
	module名称变化	-	按复杂不兼容场景处理
	augment去除	-	按复杂不兼容场景处理

	删除choice	choice {
		list
		}
		->
		list
			如果删除choice，导致表结构变化，按复杂不兼容场景处理。

 版权所有 © 华为技术有限公司

规则	不兼容项目	举例	处理说明

	删除case	choice {
		case {
		leaf1
		leaf2
		}}
		->
		choice {
		leaf1
		leaf2
		}
			如果原先一个case中多个节点被拆成多个case，则数据需要保证新数据中只能有一个case节点有值，按复杂不兼容场景处理。

	增加choice	list
		leaf
		->
		choice {
		list
		}
		leaf
			如果choice下节点涉及新增表，按复杂不兼容场景处理。

免配置规则	choice改名	旧choice aaa –>新choice bbb	按简单不兼容场景免配置规则处理
	case改名	旧case bbb –>新case ddd	按简单不兼容场景免配置规则处理

	删除choice	不涉及表结构变化的例子：
		choice {
		leaf
		}
		-> leaf
			如果choice只有leaf节点、container不涉及建表节点，删除choice则是兼容，按简单不兼容场景免配置规则处理

	增加choice	leaf
		->
		choice {
		leaf
		}
			如果choice下节点不涉及新增表，按简单不兼容场景免配置规则处理

 版权所有 © 华为技术有限公司

规则	不兼容项目	举例	处理说明

	增加case	choice {
		leafa
		leafb
		}
		->
		choice {
		case {
		leafa
		leafb
		}
		}
			按简单不兼容场景免配置规则处理

	namespace	-	按简单不兼容场景免配置规则处理
	presence属性新增、删除	-	按简单不兼容场景免配置规则处理

----结束

9.2.2.2 编写不兼容配置规则和代码

根据不兼容类型，进行对应处理。

- 如果是简单不兼容场景，可参见表9-4在迁移规则文件migration_rules.xml中配置对应的规则，或者属于免规则配置，不需要在migration_rules.xml中进行任何配置。migration_rules.xml从迁移包中resources目录下获取，修改完成后请放回到迁移包中。CHANGE_VALUE较特殊，虽然属于简单不兼容场景，但需要在业务回调接口中编写代码对节点进行处理。

表 9-4 迁移规则文件配置示例

规则	迁移规则文件配置示例 **说明：rule为规则名称，source-path为旧YANG节点路径，des-path为新YANG节点路径。**
KEY_CHANGE	<migration-rule> <rule>KEY_CHANGE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/huawei-if-ip:ipv4-ifs/ipv4</source-path> </migration-item> </migration-rule>

 版权所有 © 华为技术有限公司

规则	迁移规则文件配置示例 说明：rule为规则名称，source-path为旧YANG节点路径，des-path为新YANG节点路径。
NODE_MAP	<migration-rule> <rule>NODE_MAP</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/instanceChildlist</source-path> <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/instanceChildList</des-path> </migration-item> </migration-rule>
CHANGE_VALUE	<migration-rule> <rule>CHANGE_VALUE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/qosDomAppNode/dscp-values</source-path> </migration-item> </migration-rule>
SECURITY	<migration-rule> <rule>SECURITY</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNode</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNodeList</source-path> </migration-item> </migration-rule>
RETAIN_VALUE	<migration-rule> <rule>RETAIN_VALUE</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/l3vpnName</source-path> </migration-item> </migration-rule>
REMOVE_LEVEL	<migration-rule> <rule>REMOVE_LEVEL</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances</source-path> </migration-item> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/instanceChildContainer2</source-path> </migration-item> </migration-rule>

迁移规则配置示例如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<migration-rules>
  <migration-rule>
    <rule>RETAIN_VALUE</rule>
    <migration-item>
      <source-path>/huawei-l3vpn:l3vpn/l3vpnName</source-path>
    </migration-item>
  </migration-rule>
  <migration-rule>
    <rule>CHANGE_VALUE</rule>
    <migration-item>
      <source-path>/huawei-l3vpn:l3vpn/instances/instance/qosDomAppNode/dscp-values</source-
```

 版权所有 © 华为技术有限公司

```
path>
        </migration-item>
    </migration-rule>
    <migration-rule>
        <rule>REMOVE_LEVEL</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances</source-path>
        </migration-item>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
instanceChildContainer2</source-path>
        </migration-item>
    </migration-rule>
    <migration-rule>
        <rule>NODE_MAP</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
instanceChildlist</source-path>
            <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/instanceChildList</des-path>
        </migration-item>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNode</source-path>
            <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/securityNodeContainer/
securityNode</des-path>
        </migration-item>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNodeList</source-path>
            <des-path>/huawei-l3vpn:l3vpn/instance/instanceChildContainer/securityNodeList</des-path>
        </migration-item>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/choiceList/choiceListContainerleaf</
source-path>
            <des-path>/huawei-l3vpn:l3vpn/instance/choiceList/choiceListContainer/
choiceListContainerLeaf</des-path>
        </migration-item>
    </migration-rule>
    <migration-rule>
        <rule>SECURITY</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNode</source-path>
        </migration-item>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNodeList</source-path>
        </migration-item>
    </migration-rule>
</migration-rules>
```

- 如果是复杂不兼容场景，在迁移规则文件migration_rules.xml中配置COMPLEX规则，对应节点及子节点数据不往子表中迁移，同时业务回调接口中需要对相关复杂不兼容节点进行处理。

  COMPLEX规则配置示例如下所示。

```
<migration-rule>
    <rule>COMPLEX</rule>
    <migration-item>
        <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/
unSecurityNodeList1</source-path>
    </migration-item>
</migration-rule>
```

  复杂不兼容场景，业务需要额外编写回调脚本，当前支持Python、java接口。

  下面以Python接口进行举例，打开迁移包python目录下的.py文件，编写回调代码。

 版权所有 © 华为技术有限公司

```
from aoc.snd.snd_model_pb2.migrateInfo_pb2 import CustomMigrateOutput
from aoc.snd.snd_model_pb2.migrateInfo_pb2 import ValueChangeOutput
from aoc.sys.datastore_migrate import *

class CustomMigrate():
    """
    template
    """


    def __init__(self, logger):
        self.logger = logger


    def custom_migrate(self, aoccontext, request=None):
        """
        template
        """
        self.logger.info('custom_migrate start.')
        node_path = '/huawei-l3vpn:l3vpn'
        query_source_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50'}

        # 读取对应的数据源
        sources = read_datastore_sources(aoccontext, node_path, query_source_param)
        query_old_data_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50', 'old_domain_id':
'migrate_old'}
        write_new_data_param = {'ne_id': '8d394835-cb84-38f3-a4d5-36a7f2074b50', 'new_domain_id':
'migrate_new'}

        # 对数据源进行逐条处理
        for source in sources:

            # 读取旧SND数据源下的旧数据
            nodeDataWithSourceResult = read_migrate_node_data(aoccontext, path,
query_old_data_param, source)
            """
            modify old data: nodeDataWithSourceResult，写旧数据改造为新数据逻辑
            """
            self.logger.info('read_migrate_node_data, nodeDataWithSourceResult type = %s' %
type(nodeDataWithSourceResult))

            # 在新SND数据源写入新数据
            result = write_migrate_node_data(aoccontext, write_new_data_param, source,
nodeDataWithSourceResult)
            self.logger.info('result. %s' % result)
        return bytearray(str(result), encoding='utf8')


    # CHANGE_VALUE规则时，自动调用接口修改新值
    def value_change(self, aoccontext, request=None):
        """
        template
        """
        self.logger.info("value_change start.%s, %s" % (type(aoccontext), type(request)))
        out = ValueChangeOutput()
        if request.path == '/l3vpn:l3vpn/interface/ip/binding/vpn-instance':
            out.newData = '1234'
        else:
            out.newData = request.oldData
        self.logger.info("value_change end.")
        return out
```

9.3 编译业务包

按照如下步骤编译软件包。

步骤1 在PyCharm窗口中，单击窗口下方的“Terminal”展开命令行窗格。

 版权所有 © 华为技术有限公司

步骤2 在命令行中运行如下命令，将导出的私钥文件移动到软件包路径下“key”目录中。

(dem) C:\Users\demo\PycharmProjects\dem>copy path\to\privkey.asc .\key\privkey.asc

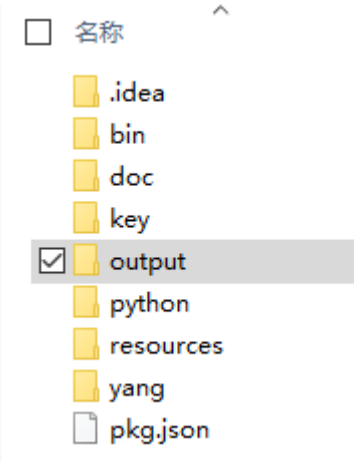
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

(dem) C:\Users\demo\PycharmProjects\dem>cd bin

(dem) C:\Users\demo\PycharmProjects\dem\bin>makeFile.bat

步骤4 运行完成后，如果命令回显出现下面的内容表示编译成功。此时，可前往软件包路径下的“output”目录中获取软件包及其签名文件。




```
2021-11-04 18:10:12,291 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.common.FileUtil] - [Sign]
Generate signature file success.
2021-11-04 18:10:12,542 INFO
[com.huawei.cloudsop.extended.pkg.mgr.tools.tool.Main] - [ZipAndSign]
Sign: Execute success
```

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

 版权所有 © 华为技术有限公司

10 导入并安装软件包

开发完成软件包后，将软件包导入开放可编程环境并安装好，方便进行在线调测或使用。

10.1 登录系统

10.2 导入并激活软件包

10.1 登录系统

开放可编程提供了两种部署方式：一种是集成到NCE-IP，作为其中一个App（Agile Open Container）发布，随NCE-IP统一部署；一种是独立软件包发布，开放可编程Mini软件包单独安装。本节描述了如何通过浏览器登录系统。目前支持的浏览器列表如下：

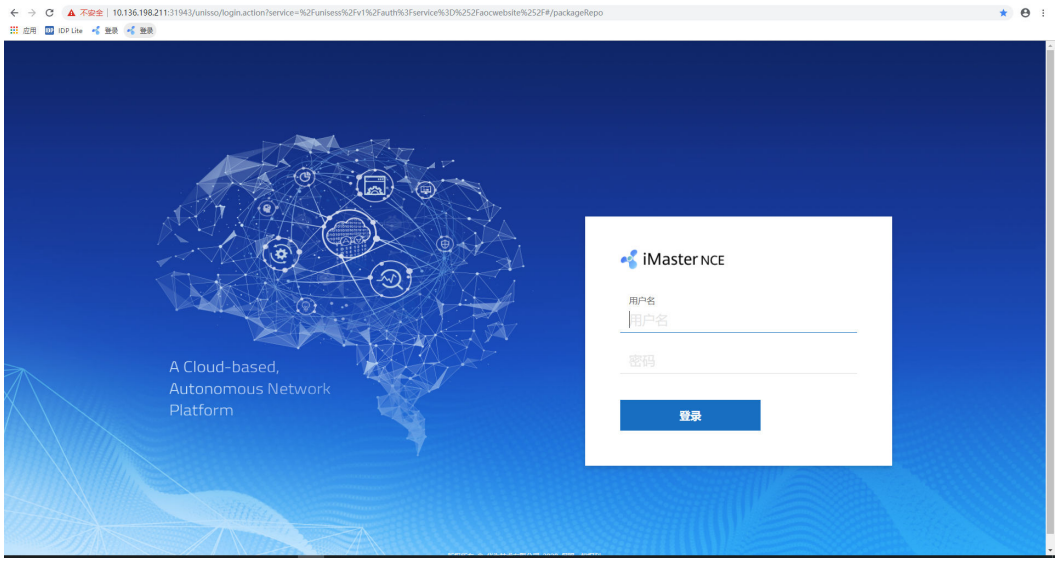
浏览器	浏览器版本
Microsoft Edge （Chromium Inside）	MicrosoftEdge_89.0.774.27_Beta
Chrome Stable Channel	Chrome 73.0.3683 开始
Firefox	• Firefox 65 开始 • Firefox 68 ESR 开始

登录 Agile Open Container App

步骤1 登录NCE运维面。在浏览器地址栏中输入“https://运维面IP地址:31943”，按Enter进入登录界面。输入“用户名”和“密码”，单击“登录”。

版权所有 © 华为技术有限公司

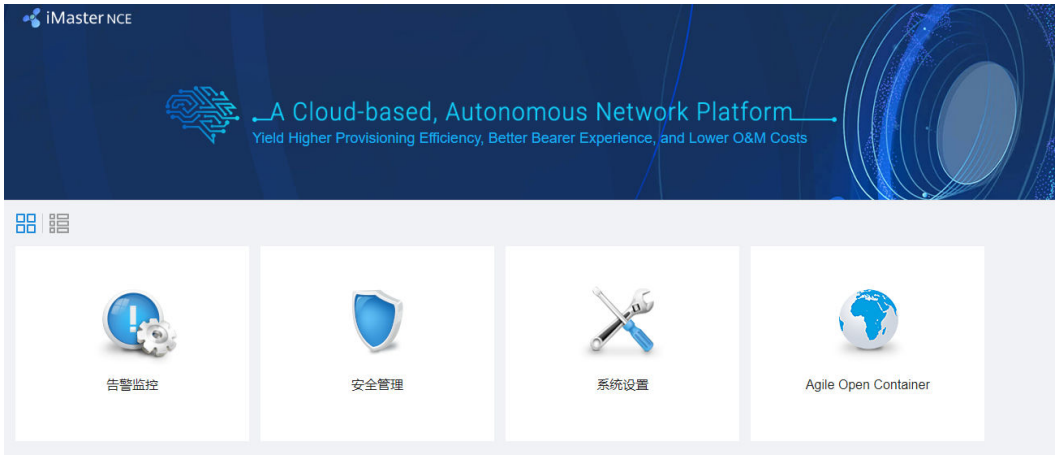
图 10-1 登录 NCE 界面



说明

- 首次登录系统需要修改密码，请妥善保存修改后的密码。为了提高系统安全性，建议用户定时修改密码，以防密码被暴力破解等安全风险。
- 运维面IP地址是指Common_Service节点配置的客户端登录IP地址。当Common_Service节点为集群部署时，此处的IP为Common_Service界点集群的浮动地址。当Common_Service节点为单实例时，此处的IP为Common_Service节点的客户端登录IP地址。

步骤2 登录系统后，选择“Agile Open Container”App，进入“Agile Open Container”首页。



 版权所有 © 华为技术有限公司

步骤3 在首页根据实际使用场景单击对应的快捷入口或者单击任意快捷入口进入系统主菜单。

----结束

登录开放可编程 Mini 系统

步骤1 登录开发者社区的“资源下载”页签，下载开放可编程Mini软件包。

步骤2 解压开放可编程Mini软件包，双击start.bat启动服务，弹出窗口。

```
[AOCmini软件包下载路径]\AOCmini\envs\Product-AOCService\controller>..\..\..\rtsp\tomcat\bin\catalina.bat start
Using CATALINA_BASE:   "[AOCmini软件包下载路径]\AOCmini/envs/Product-AOCService/"
Using CATALINA_HOME:   "[AOCmini软件包下载路径]\AOCmini/rtsp/tomcat"
Using CATALINA_TMPDIR: "[AOCmini软件包下载路径]\AOCmini/envs/Product-AOCService/\temp"
Using JRE_HOME:        "[AOCmini软件包下载路径]\AOCmini/rtsp/jdk/"
Using CLASSPATH:       "[AOCmini软件包下载路径]\AOCmini/rtsp/tomcat\bin\bootstrap.jar;[AOCmini软件包下载路径]\AOCmini/rtsp/tomcat\bin\tomcat-juli.jar"

=========================================
AOCmini is starting, please wait a moment.
```

步骤3 等待系统启动完成。等待3分钟后，系统启动完成。

```
2020-09-10 16:10:28 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:33 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:39 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:43 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:48 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:53 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:58 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:03 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:08 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:13 Console message: AOCmini started successfully, please visit https://127.0.0.1:32018/aocwebsite/ in browser.
```

步骤4 登录界面。在浏览器地址栏中输入“https://127.0.0.1:32018/aocwebsite”，按Enter进入开放可编程首页。

 版权所有 © 华为技术有限公司

图 10-2 开放可编程首页

步骤5 在首页根据实际使用场景单击对应的快捷入口或者单击任意快捷入口进入系统主菜单。

----结束

10.2 导入并激活软件包

步骤1 在主菜单中选择“包管理 > 包仓库”，从左侧导航栏选择进入“包管理”界面，单击“导入”。

图 10-3 软件包导入

步骤2 在弹出的界面中，选择需要导入的软件包和签名文件。

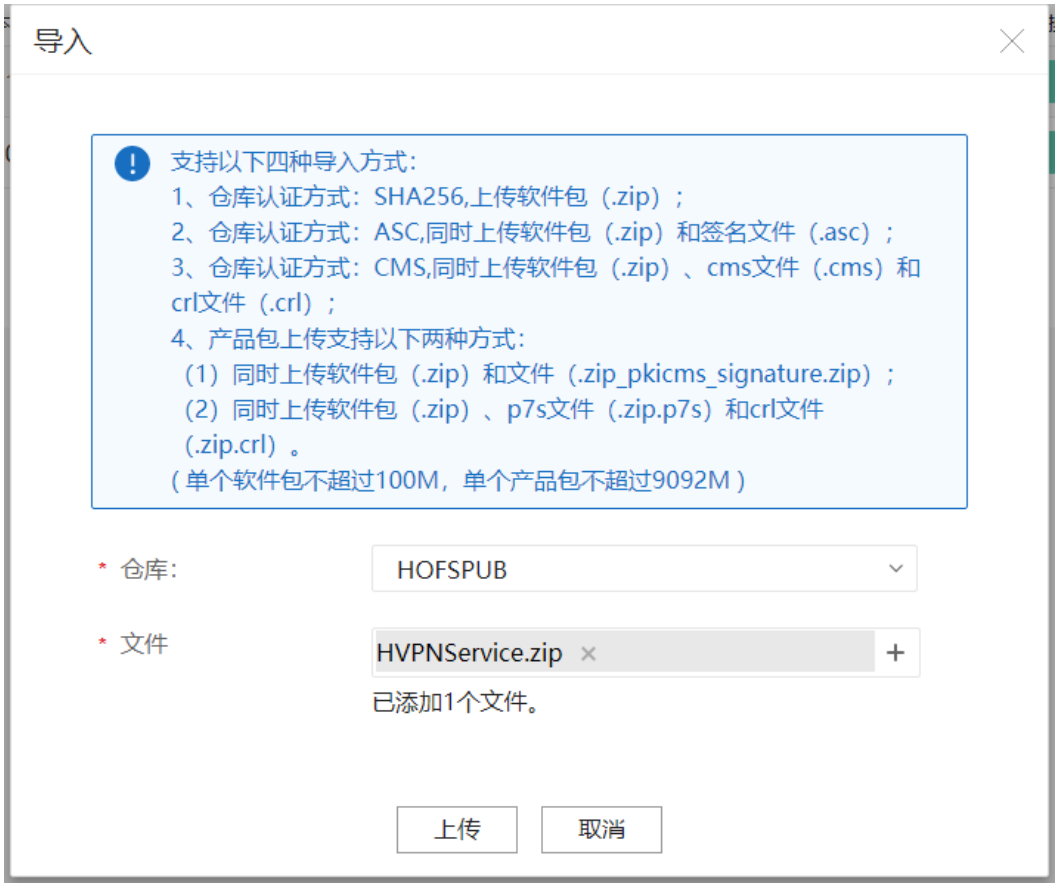
版权所有 © 华为技术有限公司

图 10-4 软件包导入选择对话框

步骤3 选择完成，单击“上传”。

 版权所有 © 华为技术有限公司

图 10-5 单击“上传”



导入完成后，在“包管理”界面可以看到导入的软件包。

图 10-6 软件包导入成功

步骤4 在主菜单中选择“包仓库”，从左侧导航栏选择进入“包管理”界面，单击 。

图 10-7 软件包安装

步骤5 软件包安装需要一定时间，请耐心等待。如果软件包状态变为激活，则说明安装成功。如果为其他信息，则根据详情列失败信息排除故障后重新安装。

 版权所有 © 华为技术有限公司

图 10-8 软件包安装成功

----结束

 版权所有 © 华为技术有限公司

11 软件包升级说明

兼容升级指升级前后软件包中的YANG模型发生的变更都是兼容的。不兼容升级指升级前后软件包中的YANG模型发生的变更存在不兼容的项，在升级时需要进行处理。

兼容/不兼容升级规格

当前支持的兼容/不兼容升级规格如下表所示。

表 11-1 兼容/不兼容升级规格

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

schema node（container, list, leaf, leaf-list, anyxml, anydata, choice, case, rpc, action, input, output, notification）	新增可选schema node	新YANG接口中出现了原Yang接口中没有的可选schema node	兼容	修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { type string; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	新增必选schema node	新YANG接口中出现了原Yang接口中没有的必选schema node（如key、mandatory）	不兼容	修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { mandatory true; type string; } }

	删除schema node	原YANG接口中的schema node，在新Yang接口中无法找到	不兼容	修改前 container topology-config { leaf id { type string; } leaf name { type string; } } 修改后 container topology-config { leaf id { type string; } }

typedef,feature	新增	新增了一个typedef/feature定义	兼容	修改前：无
				修改后：
				typedef my-type {
				status deprecated;
				type int32;
				}

	删除	删除了一个typedef/feature定义	不兼容	修改前： typedef my-type { status deprecated; type int32; } 修改后：无

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

数据类型(type)	整形类型(int8、int16、int32、int64、uint8、uint16、uint32和uint64)之间的变更，且变更后的范围扩大	比如，int8变更为int32，变更后范围扩大，属于兼容变更。	兼容	修改前：
				leaf id {
				type uint32;
				}
				修改后：
				leaf id {
				type uint64; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	数据类型变更（除上述场景）或类型不变范围缩小	1. 整形类型之间的变更，但变更后范围缩小（变更后范围不能完全包含变更前的范围，都属于范围缩小，如int8变更为uint16）。
		2. 非整形类型之间的变更及整形非整形之间的变更。（比如，string变更为leafref，uint变更为string）
		3. 数据类型不变，但范围缩小（变更后范围不能完全包含变更前的范围，都属于范围缩小，比如，string类型，变更前范围是0..20, 变更后范围是1..30）
			不兼容	举例1：
				修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type uint32; } }
				举例2：
				修改前
				typedef interface-ref { type unit8; }
				修改后
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

数量范围	数量范围扩大	同一element的数量范围发生了兼容变化：min-elements变小或max-elements变大	兼容	修改前 leaf-list id { type string; min-elements 2 max-elements 5 } 修改后 leaf-list id { type string; min-elements 1 max-elements 6 }}
	数量范围缩小	同一element的数量范围发生了不兼容变化：min-elements变大或max-elements变小	不兼容	修改前 leaf-list id { type string; min-elements 1 max-elements 6 } 修改后 leaf-list id { type string; min-elements 2 max-elements 5 }}

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
取值范围	取值范围扩大	同一属性的取值范围发生了兼容变化：range 变大	兼容	修改前 type uint8 { range "0..60"; } 修改后 type uint8 { range "0..63"; }
	取值范围缩小	同一属性的取值范围发生了不兼容变化：range 变小	不兼容	修改前 type uint8 { range "0..64"; } 修改后 type uint8 { range "0..63"; }
长度范围	长度范围扩大	同一属性的长度范围发生了兼容变化：length 变大	兼容	修改前 type string { length "0..64"; } 修改后 type string { range "0..70"; }
	长度范围缩小	同一属性的长度范围发生了不兼容变化：length 变小	不兼容	修改前 type string { length "0..64"; } 修改后 type string { range "0..50"; }
	长度范围标签从有到无	同一属性的长度范围标签删除	兼容	修改前 type string { length "0..64"; } 修改后 type string { }

	长度范围标签从无到有	同一属性新增长度范围标签	不兼容	修改前
				type string {
				}
				修改后
				type string {
				range "0..50"; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	固定长度变化	属性长度采用固定长度（length）约束时，当固定长度发生变化时，无论变大还是变小，都属于不兼容变化	不兼容	固定长度当作上下限一样的范围长度。 固定长度变为可变长度，可能是业务设计方案发生了变化，所以定义为不兼容变更。
	固定长度标签从有到无	同一属性的固定长度标签删除	不兼容	
	固定长度标签从无到有	同一属性新增固定长度标签	不兼容	

枚举	枚举项新增	同一属性的枚举值中，出现了原来不存在的枚举项	兼容	修改前
				type enumeration {
				enum aaa;
				}
				修改后
				type enumeration {
				enum aaa;
				enum bbb; }

	枚举项删除	同一属性的枚举值中，原枚举项在新的YANG接口中无法找到	不兼容	修改前 type enumeration { enum aaa; enum bbb; } 修改后 type enumeration { enum aaa; }

	枚举项修改	同一枚举的枚举项发生了变化	不兼容	修改前
				type enumeration {
				enum aaa;
				}
				修改后
				type enumeration {
				enum bbb; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	枚举值新增	同一属性的枚举值中，出现了原来不存在的枚举值	不兼容	兼容变更：
				修改前
				type enumeration {
				enum one;
				}
				修改后
				type enumeration {
				enum one {
				value 0; } }
				不兼容变更：
				修改前 type enumeration { enum one; } 修改后 type enumeration { enum one { value 1; } }

	枚举值删除	同一属性的枚举值中，原枚举值在新的YANG接口中无法找到	不兼容	兼容变更
				修改前
				type enumeration {
				enum aaa {
				value 0;
				}
				enum bbb { value 1; } }
				修改后 type enumeration { enum aaa { value 0; } enum bbb; }
				不兼容变更
				修改前 type enumeration { enum aaa { value 0; } enum bbb { value 3; } }
				修改后 type enumeration { enum aaa { value 0; } enum bbb; }

	枚举值变化	同一枚举的值发生了变化	不兼容	修改前
				type enumeration {
				enum aaa {
				value 0; } }
				修改后
				type enumeration { enum aaa { value 1; } }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
默认值	默认值变更	同一属性的默认值发生了变化	不兼容	修改前 leaf spfInitInterval { type "uint32"; default 50; } 修改后 leaf spfInitInterval { type "uint32"; default 40; }
	默认值标签从有到无	同一属性的默认值标签删除	不兼容	修改前 leaf spfInitInterval { type "uint32"; default 50; } 修改后 leaf spfInitInterval { type "uint32"; }

	默认值标签从无到有	同一属性新增默认值标签	不兼容	修改前
				leaf spfInitInterval {
				type "uint32";
				}
				修改后
				leaf spfInitInterval {
				type "uint32";
				default 40; }

正则表达式	正则表达式变化	同一属性的正则表达式发生了变化	不兼容	修改前 type string { pattern "[0-9\.]*"; } 修改后 type string { pattern "[0-9]*"; }
	正则表达式标签从有到无或删除一个或多个正则表达式	同一属性的正则表达式标签删除，表示约束范围放大	兼容	修改前 type string { pattern "[0-9\.]*"; } 修改后 type string { }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	正则表达式标签从无到有或增加一个或多个正则表达式	同一属性新增正则表达式标签，表示约束范围缩小	不兼容	修改前
				type string {
				}
				修改后
				type string {
				pattern "[0-9]*"; }

	正则表达式pattern的子语句modifier从有到无或从无到有	同一个正则表达式pattern，其子语句modifier从有到无或从无到有	不兼容	修改前：
				type string {
				length "1..max";
				pattern ’[a-zA-Z_][a-zA-Z0-9\-_.]*’;
				pattern ’[xX][mM][lL].*’ {
				modifier invert-match;
				}
				}
				修改后：
				type string {
				length "1..max";
				pattern ’[a-zA-Z_][a-zA-Z0-9\-_.]*’;
				pattern ’[xX][mM][lL].*’;
				}

key约束	key约束删除	同一list的key约束删除	不兼容	修改前 list link { key "id name"; uses link-config; } 修改后 list link { key "id"; uses link-config; }
	key约束新增	同一list的key约束新增	不兼容	修改前 list link { key "id"; uses link-config; } 修改后 list link { key "id name"; uses link-config; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	key约束修改	同一list的key约束修改	不兼容	修改前 list link { key "id"; uses link-config; } 修改后 list link { key "name"; uses link-config; }
	key顺序变化	同一list的key顺序变化	不兼容	修改前 list link { key "protocol ip"; leaf protocol { type string; } leaf ip { type string; } } 修改后 list link { key "ip protocol"; leaf ip { type string; } leaf protocol { type string; } }
mandatory约束	mandatory约束删除	同一属性的mandatory约束由true变为false	兼容	修改前 leaf name { mandatory true; type string; } 修改后 leaf name { mandatory false; type string; }
	mandatory约束新增	同一属性的mandatory约束由false变为true	不兼容	修改前 leaf name { mandatory false; type string; } 修改后 leaf name { mandatory true; type string; }

		同一对象的新增属性的mandatory约束为true	不兼容	修改前
				leaf name {
				type string;
				}
				修改后
				leaf name {
				mandatory true; type string; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

status约束	status约束使用	同一属性的status约束由obsolete变为current或者deprecated;
		同一属性的status约束由deprecated变为current;
			兼容	修改前:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status obsolete;
				description "Matching mode of nodes.";
				}
				修改后:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}

	status约束废弃	同一属性的status约束由current变为deprecated或者obsolete;
		同一属性的status约束由deprecated变成obsolete;
			不兼容	修改前:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}
				修改后:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status deprecated;
				description "Matching mode of nodes.";
				}

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
unique约束	unique约束删除	同一属性的约束删除	兼容	修改前 list server { … unique "ip port"; … } 修改后 list server { … … }

	unique约束新增	同一属性的约束新增	不兼容	修改前
				list server {
				…
				…
				}
				修改后
				list server {
				…
				unique "ip port"; … }

	unique约束修改	同一对象的unique属性个数增加	兼容	修改前
				list server {
				…
				unique "ip";
				…
				}
				修改后
				list server {
				…
				unique "ip port"; … }

		同一对象的unique属性个数减少	不兼容	修改前
				list server {
				…
				unique "ip port";
				…
				}
				修改后
				list server {
				…
				unique "ip"; … }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
		同一对象的unique属性变更	不兼容	修改前 list server { ... unique "ip port"; ... } 修改后 list server { ... unique "ip port-b"; ... }
config约束	config约束新增	同一属性的config约束（包括继承得到的场景）由true变为false	不兼容	修改前 container network-topology-oper { config true; ... } 修改后 container network-topology-oper { config false; ... }
	config约束删除	同一属性的config约束（包括继承得到的场景）由false变为true	兼容	修改前 container network-topology-oper { config false; ... } 修改后 container network-topology-oper { config true; ... }

bits约束	bits的position值增加	同一位域属性的bits的position增加	不兼容	不兼容变更：
				修改前
				type bits {
				bit disable-nagle;
				}
				修改后
				type bits {
				bit disable-nagle {
				position 1; } }
				兼容变更：
				修改前
				type bits { bit disable-nagle; }
				修改后
				type bits { bit disable-nagle { position 0; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	bits的position值删除	同一位域属性的bits的position删除	不兼容	不兼容变更：
				修改前
				type bits {
				bit disable-nagle {
				position 1; } }
				修改后 type bits { bit disable-nagle; }
				兼容变更：
				修改前
				type bits { bit disable-nagle { position 0; } }
				修改后
				type bits { bit disable-nagle; }

	bits的position值修改	同一位域属性的bits的position修改	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0; } }
				修改后
				type bits { bit disable-nagle{ position 1; } }

	新增bit项	同一位域属性的bits的新增bit项	兼容	修改前
				type bits {
				bit disable-nagle {
				position 0; } }
				修改后
				type bits { bit disable-nagle { position 0; }; bit auto-sense-speed { position 1; } }

	bit项删除	同一属性的bit项删除	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed { position 1; } } 修改后
				type bits { bit disable-nagle { position 0; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	bit项修改	同一属性新增bit项修改	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed {
				position 1;
				}
				}
				修改后
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed-xxx
				{ position 1; } }

私有扩展语法support-filter	support-filter从true改为false	某个非key leaf节点的support-filter属性从true改为false（不定义默认也为false）	不兼容	修改前： leaf flag { type string; ext:support-filter true; } 修改后： leaf flag { type string; }
	support-filter从false改为true	某个非key leaf节点的support-filter属性从false（不定义默认也为false）改为true	兼容	修改前： leaf flag { type string; } 修改后： leaf flag { type string; ext:support-filter true; }
私有扩展语法value-meaning，item，meaning	value-meaning中的item和meaning对应关系变化	某个leaf节点的类型定义中，value-meaning属性中的item和meaning对应关系变化	不兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning IGMP; } } } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	value-meaning从无到有或某条item和meaning从无到有	某个leaf节点的类型定义中，value-meaning属性整体增加，或者增加某条item和meaning	兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } }
	value-meaning从有到无或某条item和meaning从有到无	某个leaf节点的类型定义中，value-meaning属性整体删除，或者删除某条item和meaning	不兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } }
私有扩展语法case-sensitivity	case-sensitivity从有到无	某个leaf节点的case-sensitivity属性取值删除，或从非lower-and-upper改为lower-and-upper（不定义默认也为lower-and-upper）	不兼容	修改前： leaf def { type string; ext:case-sensitivity upper2lower; } 修改后： leaf def { type string; }
	case-sensitivity从无到有	某个leaf节点的case-sensitivity属性取值新增，或从lower-and-upper（不定义默认也为lower-and-upper）改为非lower-and-upper	不兼容	修改前： leaf def { type string; } 修改后： leaf def { type string; ext:case-sensitivity upper2lower; }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	case-sensitivity属性取值变更	某个leaf节点的case-sensitivity属性取值变更	不兼容	修改前： leaf def { type string; ext:case-sensitivity lower2upper; } 修改后： leaf def { type string; ext:case-sensitivity upper2lower; }

私有扩展语法value-range	value-range属性范围扩大	可输入的数值范围value-range扩大	兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "0..128"; } }

	value-range属性范围缩小	可输入的数值范围value-range缩小	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "10..63"; } }

	value-range属性从无到有	增加可输入的数值范围value-range属性	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				}
				}
				修改后：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	value-range属性从有到无	删除可输入的数值范围value-range属性	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; } }

私有扩展语法masklen，bit，position	masklen属性范围扩大	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中masklen范围扩大	兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

	masklen属性范围缩小	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中masklen范围缩小	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	原有的bit与position对应关系变化	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中原有的bit与position对应关系变化	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Fri { position 1; } ext:bit Tue { position 2; } ... } }

	某条bit和position从无到有	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中新增某条bit和position的对应关系。	兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ... } }
				修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	某条bit和position从有到无	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中删除某条bit和position的对应关系。	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ... } }

私有扩展语法task-name	task-name从无到有	同一模块的task-name标签从无到有	兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } }
	task-name从有到无	同一模块的task-name标签从有到无	不兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … container aaa { … } }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	task-name变更	同一模块的task-name属性值变更	不兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaaom"; container aaa { … } }
私有扩展语法node-ref	增加node-ref	同一个rpc增加node-ref	兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }
	删除node-ref	同一个rpc删除node-ref。一个rpc可能有多个node-ref，包括删除其中一个或所有node-ref	不兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } }
	node-ref变更	同一个node-ref属性的值发生变更。	不兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
私有扩展语法operation-exclude	operation-exclude新增	某一个节点新增operation-exclude属性，或operation-exclude属性值增加	不兼容	修改前： container aaa { leaf bbb { type uint32; } } 修改后： container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	operation-exclude删除	某一个节点删除operation-exclude属性，或operation-exclude属性值删除	兼容	修改前： container aaa { leaf bbb { ext:operation-exclude update|create; type uint32; } } 修改后： container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	operation-exclude变更	包括：子语句when条件变更；子语句filter过滤条件变更。	不兼容	修改前： list aaa { ext:operation-exclude create|delete { ext:filter "name = '_public_'"; } ... } 修改后： list aaa { ext:operation-exclude create|delete { ext:filter "name = '_default_'"; } ... }
私有扩展语法dynamic-default	无默认值变更为dynamic-default	某一个叶子节点从无默认值变更成dynamic-default	不兼容	修改前： container aaa { leaf bbb { type uint32; } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
	dynamic-default变更为无默认值	某一个叶子节点从dynamic-default变更为无默认值	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; } }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	静态默认值变更为dynamic-default	某一个叶子节点从静态默认值（default属性）变更为dynamic-default	不兼容	修改前： container aaa { leaf bbb { type uint32; default 12; } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
	dynamic-default变更为静态默认值	某一个叶子节点从dynamic-default变更为静态默认值（default属性）	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; default 12; } }
	dynamic-default变更	dynamic-default属性的default-value取值变更或default-value的子语句when条件发生变更	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
私有扩展语法generated-by	generated-by新增、修改、删除	-	不兼容	-

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
私有扩展语法 refine-ext	refine-ext中新增扩展 operation-exclude	某一个节点的refine-ext中新增扩展 operation-exclude	不兼容	-
	refine-ext中删除扩展 operation-exclude	某一个节点的refine-ext中删除扩展 operation-exclude	兼容	-
	refine-ext中 operation-exclude发生变更	某一个节点的refine-ext中的 operation-exclude发生变更	不兼容	-
	refine-ext中 generated-by新增、删除、变更	某一个节点的refine-ext中的 generated-by发生变更	不兼容	-
description标签	description 标签变更	同一属性description标签变更（包括私有语法中的description标签变更）	兼容	修改前 description "xxx"; 修改后 description "xxx xxx";
leafref约束	新增leafref	新增了leafref关系，进行了关联限制约束	不兼容	修改前 typedef interface-ref { type unit8; } 修改后 typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	删除leafref	删除了leafref关系，放开了关联限制约束。	兼容	修改前 typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } } 修改后 typedef interface-ref { type string; }

	同一字段的path变化	leafref中同一对象关联的path变化，对关联限制约束进行了修改	不兼容	修改前
				typedef interface-ref {
				type leafref {
				path "/if:interfaces/if:interface/if:ip"; } } 修改后
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

	require-instance从true改为false	leafref的子语句require-instance默认为true，由true改为false	兼容	修改前：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}
				修改后：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	require-instance从false改为true	leafref的子语句require-instance默认为true，由false改为true	不兼容	修改前：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}
				修改后：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}

decimal64	fraction-digits变更	decimal64类型的子语句fraction-digits变更，比如从2变更为3	不兼容	修改前：
				typedef my-decimal {
				type decimal64 {
				fraction-digits 2;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}
				修改后：
				typedef my-decimal {
				type decimal64 {
				fraction-digits 3;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

units标签	units标签值变更	同一属性的units标签值发生了变化	不兼容	修改前
				leaf interval {
				type uint16;
				default 30;
				units minutes;
				}
				修改后
				leaf interval {
				type uint16;
				default 30;
				units seconds; }

	units标签从有到无	同一属性的units标签删除	兼容	修改前 leaf interval { type uint16; default 30; units minutes; } 修改后 leaf interval { type uint16; default 30; }

	units标签从无到有	同一属性新增units标签	不兼容	修改前
				leaf interval {
				type uint16;
				default 30;
				}
				修改后
				leaf interval {
				type uint16;
				default 30;
				units minutes; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
reference标签	reference变更	同一属性的reference发生了变化	兼容	修改前 typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... } 修改后 typedef uri { type string; reference "RFC 5040: Uniform Resource Identifier (URI): Generic Syntax"; ... }

	reference标签删除	同一属性的reference标签删除，表示约束范围放大	兼容	修改前
				typedef uri {
				type string;
				reference
				"RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
				...
				}
				修改后
				typedef uri {
				type string; }

	reference标签新增	同一属性新增reference标签，表示约束范围缩小	兼容	修改前 typedef uri { type string; } 修改后 typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... }
when/must	when/must变更	同一属性的when/must发生了变化	不兼容	修改前 leaf isFlowKey { when "../../setId = 2"; } 修改后 leaf isFlowKey { when "setName = 3"; }
	when/must标签从有到无	同一属性的when/must标签删除，表示约束范围放大	兼容	修改前 leaf isFlowKey { when "../../setId = 2"; } 修改后 leaf isFlowKey { ⋯... }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	when/must标签从无到有	同一属性新增when/must标签，表示约束范围缩小	不兼容	修改前
				leaf isFlowKey {
				…...
				}
				修改后
				leaf isFlowKey {
				when "setName = 3"; }

	must条件的error-message变更，从有到无，从无到有	must条件的子语句error-message变更，从有到无，从无到有	兼容	修改前：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-message "An ethernet MTU must be 1500";
				}
				修改后：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

	must条件的error-app-tag变更，从有到无，从无到有	must条件的子语句error-app-tag变更，从有到无，从无到有	兼容	修改前：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-app-tag "must-violation";
				}
				修改后：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

if-feature标签	if-feature标签值变更	同一属性的if-feature标签值发生了变化	不兼容	修改前 container location { if-feature has-gps; leaf longitude { mandatory true; ... } } 修改后 container location { if-feature has-gps-xxx; leaf longitude { mandatory true; ... } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	if-feature标签从有到无	同一属性的if-feature标签删除	兼容	修改前 container location { if-feature has-gps; leaf longitude { mandatory true; ... } } 修改后 container location { leaf longitude { mandatory true; ... } }

	if-feature标签从无到有	同一属性新增if-feature标签	不兼容	修改前
				container location {
				leaf longitude {
				mandatory true;
				...
				}
				}
				修改后
				container location {
				if-feature has-gps; leaf longitude { mandatory true; ... } }

submodule标签	submodule标签值变更	同一模块的submodule标签值发生了变化	不兼容	修改前： submodule huawei-ac-ne-staticrt-staticrtbase 修改后： submodule huawei-ac-ne-staticrt-staticrtbase-xxx
	submodule标签新增	同一模块新增submodule标签	兼容	-
	submodule标签删除	同一模块删除submodule标签	不兼容	-

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

belongs-to	belongs-to变更、新增、删除	同一个子模块的belongs-to属性变更	不兼容	修改前：
				submodule huawei-qos-cbqos {
				belongs-to huawei-qos {
				prefix qos;
				}
				修改后：
				submodule huawei-qos-cbqos {
				belongs-to huawei-acl {
				prefix qos;
				}

module	module标签值变更	同一模块的module标签值发生了变化	不兼容	修改前 module huawei-ac-ne-staticrt 修改后 module huawei-ac-ne-staticrt-xxx
prefix	prefix声明变更	同一模块的prefix声明发生了变化	不兼容	修改前 prefix "ac-ne-ifm"; 修改后 prefix "ac-ne-ifm-xxx";
yang-version	yang-version变更	当前yang-version有两个取值，如下： yang-version 1; yang-version 1.1; 当不显式指定时，yang-version的默认值为1. 当yang-version从1变更为1.1或者从1.1变更为1，都属于不兼容变更。	不兼容	修改前： yang-version 1.1; 修改后： 没有显式指定yang-version

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

organization/ contact/ revision	organization/ contact/ revision的变更，从有到无，从无到有	organization/ contact/ revision的变更，从有到无，从无到有	兼容	示例：
				organization
				"Huawei Technologies Co., Ltd.";
				contact
				"Huawei Industrial Base
				Bantian, Longgang
				Shenzhen 518129
				People's Republic of China
				Website: http://www.huawei.com
				Email: support@huawei.com";
				revision 2018-07-02 {
				description "
				Initial version.";
				}

case	Case新增	同一节点中出现了原来不存在的Case	兼容	修改前 choice interface-type { case a { leaf ethernet { ... } } } 修改后 choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } }
	Case删除	同一节点中删除一个Case	不兼容	修改前 choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } } 修改后 choice interface-type { case a { leaf ethernet { ... } } }
namespace	namespace变更	namespace标签值变更	不兼容	修改前 namespace "urn:huawei:yang:huawei-ac-ne-ifm"; 修改后 namespace "urn:huawei:yang:huawei-ac-ne-ifm-xxx";

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

notification携带的参数	元素新增	同一个notification中新增参数，只能从后面追加，不能从中间插入新参数	兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				}
				修改后
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status {
				type types:admin-status;
				}
				}

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	元素删除	同一个notification的参数删除	不兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				修改后
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	元素变更	同一个notification的同一参数变更	不兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				修改后
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } leaf admin-status-xxx { type types:admin-status; } }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

identity	identity新增	新增identity定义	兼容	修改前
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				修改后
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	identity删除	删除identity定义	不兼容	修改前
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; }
				修改后
				identity authentication-method { description "Base identity for user authentication methods."; }
				identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	identity变更	identity定义变更	不兼容	修改前
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }
				修改后
				identity authentication-method { description "Base identity for user authentication methods."; }
				identity radius-xxx { base authentication-method; description "Indicates user authentication using RADIUS."; }

base	base新增	同一个identity下，新增一个base语句。	兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	base删除	同一个identity下，删除一个base语句。	不兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}

	base变更	同一个identity下，base发生变更。	不兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

extension	extension新增	新增extension定义	兼容	修改前
				extension item {
				argument value;
				}
				修改后
				extension item {
				argument value;
				}
				extension index { argument value; }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	extension删除	删除extension定义	不兼容	修改前
				extension item {
				argument value;
				}
				extension index { argument value; }
				修改后
				extension item { argument value; }

	extension变更	extension定义变更	不兼容	修改前
				extension item {
				argument value;
				}
				修改后
				extension item-xxx {
				argument value;
				}

argument	argument变更	extension的子语句argument变更，argument后指定的是参数名称	不兼容	修改前：
				extension c-define {
				argument index;
				}
				修改后：
				extension c-define {
				argument value;
				}

	argument从有到无，从无到有	extension的子语句argument从有到无，说明该extension原来有参数改为无参数
		extension的子语句argument从无到有，说明该extension原来无参数改为有参数
			不兼容	修改前：
				extension c-define {
				argument index;
				}
				修改后：
				extension c-define;

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

yin-element	yin-element变更	argument的子语句yin-element有两个取值，true和false，默认值为false，当yin-element由true变false或由false变true	不兼容	修改前：
				extension c-define {
				argument index {
				yin-element true;
				}
				}
				修改后：
				extension c-define {
				argument index;
				}

revision-date	新增	在import和include中新增revision-date子语句	不兼容	include huawei-ac-ne-evnbgp-type {
				revision-date 2016-01-05;
				}

	删除	在import和include中删除revision-date子语句	不兼容	
	变更	在import和include中修改revision-date子语句	不兼容	

augment	新增	新增augment定义	兼容	新增如下定义：
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	删除、变更	删除或者变更augment定义	不兼容	删除或者修改如下定义：
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

refine	新增、删除、变更	新增、删除、变更都算不兼容	不兼容	container connection {
				container source {
				uses target {
				refine "address" {
				description "Source IP address";
				}
				refine "port" {
				description "Source port number";
				}
				}
				}

presence	变更	presence描述变更	兼容	修改前：
				container "ssh" { presence "Enables SSH"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }
				修改后：
				container "ssh" { presence "Enable SSH configuration"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }

 版权所有 © 华为技术有限公司

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	新增、删除	presence属性新增、删除	不兼容	container "ssh" {
				presence "Enables SSH";
				description "SSH service specific configuration";
				// more leafs, containers and stuff here...
				}

ordered-by	新增、删除、变更	新增、删除、变更都算不兼容	不兼容	leaf-list cipher {
				type string;
				ordered-by user;
				description "A list of ciphers";
				}

版权所有 © 华为技术有限公司

12 开放可编程开发规范

12.1 YANG开发规范
12.2 Python开发规范
12.3 其他注意事项

12.1 YANG 开发规范

12.1.1 YANG 文件的基本内容和布局

YANG文件通常遵循如下文件布局格式：

```
module <module-name> {
   // header information
   <YANG-version statement>
   <namespace statement>
   <prefix statement>

   <import statements>
   <include statements>

   <organization statement>
   <contact statement>
   <description statement>
   <reference statement>

   // revision history
   <revision statements>

   // module definitions
   <other statements>
}
```

文件前头是模块或者是子模块的关键字（module/submodule），关键字后是对应的模块、子模块的名字，然后通过{}包含整个模型文件的语句段。

说明：

文件内容可以划分如下部分：

 版权所有 © 华为技术有限公司

1. 文件头信息（header information）：文件头信息域中包含文件的YANG版本信息，现在有的版本是1.0和1.1， YANG文件不明确指明版本时，默认为1.0版本，在YANG1.0时（对应标准RFC6020）YANG文件的版本信息可选，而在YANG1.1时（对应标准为RFC7950）必须明确定义版本为1.1。
2. 引用文件的声明：本文件定义的模块/子模块对其他的模块、子模块可以进行引用声明，通过import引用其他的模块，通过include来引用子模块。
3. 文件的其它元数据信息：定义组织信息（organization）、文件联系人（contact）、文件描述（description）、文件引用说明（reference）。
4. 文件的修订历史：通过revision的说明，说明当前文件的版本和历史的版本信息。
5. 文件内容：在文件内容域中可以包含类型定义、复用语句段（grouping）、特性开关说明（feature）、标识符（indentify）定义、文件信息节点，顶层的节点建议是container、rpc、notification、augment的组合。

12.1.2 YANG 模型文件命名/模块命名统一规定

12.1.2.1 规则：YANG 模型文件名称的合法字集为：小写字母（a-z）、数字（0-9）和连字符（-），且最大长度不能超过 63 个字符（不含版本号和.yang）

说明：

1. 设备北向接口以“huawei-”开头。
   a. 私有YANG文件命名：

      huawei-<featurename>.yang，其中<featurename>为特性名。当该YANG为主模块时，模块名（即huawei-<featurename>）长度不超过31字符。
   b. 私有YANG的deviation文件命名：

      huawei-<featurename>-deviations-[<productname>].yang, 其中<featurename>为特性名，<productname>为产品名，即私有YANG的deviation YANG文件名为私有YANG文件名后面加“-deviations-[<productname>]”。要求huawei-<featurename>长度不超过31字符，<productname>长度不超过20字符（以路由器举例，路由器的<productname>在极端情况下又可以细化格式为<xxxx>-[-ptn][-vs]，要保证<xxxx>-[-ptn][-vs]整体不超过）。
   c. 标准YANG的deviation文件命名：

      huawei-<standardname>-deviations-[<productname>].yang，其中<standardname>为标准YANG名称，<productname>为产品名。
   d. Schema mount下的deviation文件命名：

      huawei-<mount point名>-<featurename | standardname>-deviations-[<productname>].yang。
   e. 诊断特性的YANG模型文件命名：

      以“huawei-diagnose-”开头。
2. AC控制器北向接口以“huawei-ac-”开头。
   a. 私有YANG文件命名：

      huawei-ac-<featurename>.yang，其中<featurename>为特性名。
   b. 私有YANG的deviation文件命名：

      huawei-ac-<featurename>-deviations-[<productname>].yang， 其中<featurename>为特性名，<productname>为产品名。<productname>可选。

 版权所有 © 华为技术有限公司

c. 标准YANG的deviation文件命名：

huawei-ac-<standardname>-deviations-[<productname>].yang，其中<standardname>为标准YANG名称，<productname>为产品名。<productname>可选。

3. NCE北向接口以“huawei-nce-”开头。

12.1.2.2 规则：YANG 模块名必须与 YANG 文件名称一致（不含版本号和.yang），YANG 模块名要求全系统唯一，最大长度不能超过 63 个字符

说明：

1. 比如YANG文件名称为huawei-aaa.yang，则其模块名（ module name ）应该为huawei-aaa。
2. 子模块名（ sub module name ）也一样，与子模块YANG文件名称一致。比如子模块的YANG文件名称为huawei-aaa-lam.yang，则其子模块名应该为huawei-aaa-lam。

12.1.3 YANG 模型文件 namespace 统一规定

12.1.3.1 规则：YANG 模型文件的 namespace 规则为 urn:XXX:yang:<module-name>

说明：

YANG模型文件namespace要遵循相同的规则，格式为urn:XXX:yang:<module-name>。其中<module-name>为模块名。

示例：

假如模块名为huawei-abc，其namespace为：urn:huawei:yang:huawei-abc。

12.1.4 YANG 模型文件的元素名称定义统一规定

12.1.4.1 规则：YANG 模型文件的元素命名合法字符集：小写字母（a-z）、数字（0-9）和连字符（-），且最大长度不能超过 63 个字符。其中首字符必须以小写字母开头

说明：

这里的元素名包含除枚举项（非枚举类型名称）和identity标识符之外的所有标识符，枚举类型名称也要遵守该规则。

元素的命名要尽量使用定义清晰的名词，或者名称的组合，组合间必须使用“-”连接；此外可以使用形容词+名词的方式来定义名称的组合，特定场景下可以使用动宾组合。

元素名称不建议以hw, huawei开头。

元素名称中不建议存在类似config, configs, cfg, cfgs, configuration, configurations, infomation, infomations, information, informations, table, tables, new, news, cli, mib, display等冗余后缀。

元素名称最大长度不能超过63个字符，建议不要超过31个字符。

版权所有 © 华为技术有限公司

示例：

user, board, create-time, ipv4-address 为符合规范命名。

反例：

123, -abc, if_name, abc@1, Admin, testName, createtime, interface-cfg, arp-table, huawei-info为不符合规范命名。

12.1.4.2 规则：YANG 模型文件中的枚举项 enum/identity 标识符合法字符集：字母（优先使用小写字母 a-z，如有专有术语、缩略语等不适合用小写字母的可使用大写字母 A-Z）、数字（0-9）、小数点（.）和连字符（-），且最大长度不能超过 63 个字符。其中 enum 的首字符必须以字母或数字开头，identity 的首字符必须以字母开头

说明：

对于字母，优先使用小写，只有不合适使用小写的场景（如专有术语、缩略语）才允许使用大写。一个枚举项或identity或者全大写或者全小写，不能大小写混用。

一个枚举项不能是纯数字。

按照RFC6020的要求，enum后是一个字符串，identity后是一个标识符，RFC6020中对标识符的命名，要求首字符不能以数字开头，所以两者在首字符的要求上有差异。

示例：

```
identity chassis-type {
  description
  "Base identity for chassis type.
   vendors can define physical type of the chassis";
}
identity MA5800-X2 {
  base dev-com:chassis-type;
  description
  "Backplane of the MA5801.";
}
```

错误示例：

```
leaf speed {
  type enumeration {
    enum 1Gbps{
      description "1G speed mode.";
    }
    enum 2.5Gbps{
      description "2.5G speed mode.";
    }
    enum 10Gbps{
      description "10G speed mode.";
    }
  }
  description "The speed of interface.";
}
```


版权所有 © 华为技术有限公司

12.1.4.3 规则：Deviations YANG 文件中如果需要使用 typedef 对数据模型中的类型做重定义，名称必须为原类型名称加”-deviations”后缀

说明：

YANG deviations标准语法不能对type类型下的属性做deviate说明（比如不能直接对某个枚举项做裁剪），只能对整个type类型做replace，所以可能在deviations YANG文件中重定义某个数据模型中定义的类型，做某些修改，再对type类型做replace，达到裁剪定制的目的。

示例1：

数据模型中定义的类型及节点引用：

```
typedef protocol {
  type enumeration {
    enum ISIS {
      description "ISIS protocol.";
    }
    enum OSPF {
      description "OSPF protocol.";
    }
}
leaf import-protocol {
type protocol;
description “import protocol”;
}
```

某个产品形态上只支持部分枚举项，deviations YANG定义如下：

```
typedef protocol-deviations {
  type enumeration {
    enum ISIS {
      description "ISIS protocol.";
    }
}
deviation /abc:import-protocol {    //假设定义数据模型文件的prefix为abc
  deviate replace {
    type protocol-deviations;
  }
}
```

12.1.4.4 规则：各元素命名风格要统一，同一系统中的相同概念的命名保持相同的名称

示例：

如“bd”与“bridge-domain”在使用中应该保持统一，不应该使用不同的命名。

12.1.5 YANG 文件排版和注释的统一规定

12.1.5.1 规则： YANG 文件排版和注释要风格统一，保证良好的视觉分级效果

说明：

YANG文件排版和注释要风格统一，保证良好的视觉分级效果。具体如下：

1. 使用缩进保持良好的视觉分级效果，建议缩进采用2个空格，禁止直接使用TAB来缩进，避免不同的编辑器设置导致的可能出现的对齐的差异。

版权所有 © 华为技术有限公司

2. 相对独立的功能块之间、标识符说明之后必须加空行。
3. 所有定义的标识符必须提供描述（description）且不能为空，同时不要将标识符直接复制过来作为描述。描述的内容要能够对该元素的定义原因、含义及相关的配置约束做明确说明，方便他人理解和使用。
4. 建议针对引用的相关标准的说明，可以使用reference来描述。
5. 建议注释符（包括“/* */”、“//”）与注释内容之间要用一个空格进行分隔。
6. 建议模型中关系较为紧密的定义应尽可能相邻。譬如优先把必配的参数根据逻辑关系放在前面，而选配的参数可以放在后面。

12.1.6 模型文件划分原则

12.1.6.1 原则：模型的划分根据系统功能特性为基础，根据实际业务逻辑必需的数据来定义模型的要素

说明：

1. 模型的划分单元是特性，特性之间分别单独定义YANG文件，例如：topology，inventory都属于独立的特性。
2. 如果一个特性含有多个子特性，可以为每个子特性再单独定义YANG文件，但不做强制要求。可以根据子特性配置管理功能的独立性，配置管理对象规模大小，做适当的调整。

12.1.6.2 原则：在内置类型，IETF 公共类型无法满足要求的前提下，可以在全系统定义一个公共的数据类型文件，其它 YANG 文件按需引用

说明：

引用公共类型的选择顺序： 优先使用内置类型，其次IETF的公共类型，最后自定义的公共类型。当内置类型或IETF公共类型满足要求时，禁止再自定义其它公共类型。

1. 内置类型包括：int8, int16, int32, int64, uint8, uint16, uint32, uint64, string, boolean, enumeration, bits, binary, leafref, identityref, empty, union, instance-identifier, decimal64。具体含义及规格见RFC6020。
2. IETF公共类型：当前IETF定义YANG公共类型的文件有两个，ietf-inet-types.yang、ietf-yang-types.yang，具体类型见YANG文件。其它：接口的公共类型定义文件为iana-if-type.yang。
3. 自定义公共类型：自定义公共类型可以分层定义，网络层面统一定义一个公共类型文件huawei-pub-type.yang，各产品各领域可以在此基础上按需单独定义该产品该领域的公共类型YANG文件。

12.1.6.3 原则：按照业界 YANG 模型的定义习惯，类型与操作可以与数据模型定义到同一个 YANG 文件中

12.1.6.4 原则：YANG 模块划分要高内聚、低耦合，符合业务聚合特征，且必须符合架构分层依赖原则。不能出现循环依赖的场景

12.1.7 模型文件内部各部分内容的规定及设计规则

版权所有 © 华为技术有限公司

12.1.7.1 规则：list 的记录对象，必须按照 YANG 语法来定义 key

说明：

key字段要定义在非key字段的前面，有多个key时，定义顺序要和list的key属性中的顺序一致。

示例：

```
container interfaces {
list interface {
      key “name index”; //存在组合key时，中间用空格隔开, key属性name在index的前面，下面定义节点时先定义name，再定义index。
      leaf name {
         type string ;
      }
      leaf index {
         type uint32 ;
      }
}
}
```

12.1.7.2 规则：list 对象外要单独定义一层 container，container 的名字为 list 的名字后加 s 或 es

说明：

增加该规则的原因，是方便针对list的多个实例做批量操作，比如要删除list的所有实例，直接删除container即可。或者要对list的所有实例做整体的replace替换，在container层做replace就会很方便。RFC8407中对此也做了说明，如下：

The NETCONF and RESTCONF protocols do not currently support the ability to delete all list (or leaf-list) entries at once. This deficiency is sometimes avoided by use of a parent container (i.e., deleting the container also removes all child entries).

list对象外层单独定义的父container名字，为list的名字后直接加s或es，这里不考虑语法。比如，list名字为’topology’，则其父container名字建议为’topologys’.再比如，list名字为’address’，则其父container名字建议为’addresses’.

示例：

```
container interfaces {
list interface {
      key “name index”;
      leaf name {
         type string {
            length “1..31”;
}
      }
      leaf index {
         type uint32 ;
      }
}
}
```

 版权所有 © 华为技术有限公司

12.1.7.3 规则：除 key 值节点以外，在创建父节点时必须携带的字段，要通过 mandatory 来指定

说明：

mandatory的取值为true和false，默认值是false。

示例：

```
leaf ipmask {
    type uint32;
    mandatory true;
}
```

说明：

在YANG中，创建父节点时（或rpc的input/output参数下发/上送时，notification的参数上送时）必须携带的字段，有且仅有如下几种情况：

1. key字段（仅list才可以并且必须定义key）
2. mandatory true的字段（如果该节点或其父节点有when条件，当when条件满足时，该节点有意义，mandatory true才生效，否则当when条件不满足时，该节点无意义，mandatory true也不生效）
3. 子节点为list/leaf-list，而且子节点的min-elements大于0（假设为m），说明在父节点存在时，子节点最少要有m条实例。

12.1.7.4 规则：在所有类型的 YANG 文件中，必须用 config 为只读状态的 element 定义访问权限

说明：

1. config声明将字符串true或false作为参数。如果config为true，该element表示配置。配置的数据节点将会是<get-config>请求的回复的一部分，将会在<copy-config>或<edit-config>请求中发送。
2. 如果config为false，该element表示状态数据。状态数据的节点将会是<get>的回复的一部分，但不能是<get-config>请求的回复的一部分，也不能在<copy-config>或<edit-config>请求中发送。
3. 如果config未指定，默认值与父schema节点的config值相同。如果父节点是一个case节点，值与case节点的父choice节点相同。
4. 如果顶级节点不指定一个config声明，默认为true。
5. 对于只读状态的元素，应该指定config为false。
6. 如果父节点或祖先节点已经指定config为false，为了整洁不冗余，子孙节点不需要再显式指定config为false。

示例：

```
leaf loper {
type uint16;
config false;
}
```

12.1.7.5 规则：如果一个或者多个 element 的组合在所有记录中取值是唯一的，则在 YANG 文件中，必须用 unique 定义其唯一性属性

说明：

 版权所有 © 华为技术有限公司

unique声明，如果一个表下有多个字段分别都是唯一的，用多个unique分别来说明，如下：

unique "aaa";

unique "bbb";

如果是多个字段组合后唯一，在一个unique下说明，几个字段之间用空格隔开，如下：

unique "ip port";

示例：

有如下列表：

```
list server {
key "name";
unique "ip port";
leaf name {
type string;
}
leaf ip {
type inet:ip-address;
}
leaf port {
type inet:port-number;
}
}
```

如下配置无效：

```
<server>
<name>smtp</name>
<ip>192.168.2.1</ip>
<port>25</port>
</server>

<server>
<name>http</name>
<ip>192.168.2.1</ip>
<port>25</port>
</server>
```

而如下配置有效，因为”http”和”ftp”两个实例没有对unique约束条件中的两个叶子节点都赋值，因此在unique约束条件被执行时不考虑。

```
<server>
<name>smtp</name>
<ip>192.168.2.1</ip>
<port>25</port>
</server>

<server>
<name>http</name>
<ip>192.168.2.1</ip>
</server>

<server>
<name>ftp</name>
<ip>192.168.2.1</ip>
</server>
```

 版权所有 © 华为技术有限公司

12.1.7.6 规则：一个对象属性 element 的取值，如果有明确的度量单位，则要使用 units 指定单位信息

示例：

```
leaf interval {
type uint16;
units min;
    default 30;
}
```

units语法后面允许两种格式，一种是单位，另一种是数值+空格+单位，数值只能是整数，小数（浮点数），分数。单位必须从第9章中选取。

比如，units “100 ms”；

units “0.01 ms”；

units “1/256 °C”；

12.1.7.7 规则：对于有明确默认值的配置对象属性，在 YANG 文件中必须要使用 default 属性为对应的 element 指定默认值

说明：

1. 是否有明确默认值的几种场景说明：
   a. 不同产品形态默认值不一样，属于有明确默认值的场景，在YANG文件中选择大多数产品的默认值作为其默认值，其它产品使用deviation语法对默认值做偏差声明。

      示例：

      YANG文件中的定义：

      ```
      leaf port-tx-en-delay {
      type uint32 {
      range "0..86400000";
      }
      default "0";
      }
      ```

      某个产品上该节点的默认值是10，则使用deviation语法做偏差声明，如下：

      假设引用port-tx-en-delay节点所在YANG文件时，声明其prefix为port，port-tx-en-delay为顶层节点。

      示例：

      ```
      deviation /port: port-tx-en-delay {
         deviate replace {
            default 10;
         }
      }
      ```

   b. 某个节点的默认值是由PAF或硬件芯片等决定的，属于没有明确默认值的场景，无法定义默认值。
   c. 某个节点的默认值是由其它一个或多个节点动态计算出来的，属于没有明确默认值的场景，无法定义默认值。
2. 只支持leaf/leaf-list element指定默认值，不支持为非leaf/leaf-list element指定默认值。

 版权所有 © 华为技术有限公司

12.1.7.8 规则：对于字符串类型，必须对长度做限制，可以使用 length 显式指定长度范围，也可以使用 pattern 隐式约束长度；有格式或字符输入限制时，必须使用 pattern 约束

示例1：对格式或字符输入有限制，且pattern可以隐式约束长度，只定义了pattern，不用再定义length。

```
typedef ipv4-address {
   type string {
      pattern
         '(([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])\.){3}'
       +  '([0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])'
       + '(%[\p{N}\p{L}]+)?';
   }
   description
      "The ipv4-address type represents an IPv4 address in
       dotted-quad notation.  The IPv4 address may include a zone
       index, separated by a % sign.

       The zone index is used to disambiguate identical address
       values.  For link-local addresses, the zone index will
       typically be the interface index number or the name of an
       interface.  If the zone index is not present, the default
       zone of the device will be used.

       The canonical format for the zone index is the numerical
       format";
 }
```

示例2：对格式或字符输入有限制，且pattern无法隐式约束长度，length和pattern都定义。

```
leaf vendor-name {
      type string {
           length "1..20";
pattern ‘[a-zA-Z0-9_]*’;
      }
}
```

示例3：对格式或字符输入没有限制（即可输入任意可见字符），只定义length，未定义pattern。

```
leaf vendor-name {
      type string {
           length "1..20";
      }
}
```

12.1.7.9 规则：对于有明确取值范围的数值类型 element，在 YANG 文件中必须使用 range 属性指定对应的 element 的取值范围，且最大最小值要简洁，可以使用 min 或 max 代替该数值类型的最小或最大值

说明：

如果节点的取值范围与其数值类型本身的范围完全一样，允许不定义range。比如，节点的类型为uint8，uint8的范围是0-255的整数，如果该节点的取值范围就是0-255的整数，则允许不定义range，如果该节点的取值范围是1-255的整数，则必须定义range。

 版权所有 © 华为技术有限公司

示例：

```
leaf ent-index {
      type uint32 {
range "1..max"; //仅对最小值有约束，最大值使用uint32类型的最大值
      }
}
```

12.1.7.10 规则：grouping 中的节点引用 grouping 外的节点要避免使用相对路径

说明：

1. 如grouping中的节点使用了leafref，如果leafref是通过相对路径引用本grouping内节点的，那属于正常使用。但是如果是通过相对路径引用其他grouping内节点的，基本上需要uses的地方一定同时uses两个grouping才不会报错，这影响了grouping的可重用性。
2. 即使leafref使用绝对路径也要小心一些，因为这意味着所有uses该grouping的地方，对该节点的解释都是一致的，即都要ref到一个固定的节点，这在一定程度上也影响了grouping的可重用性。
3. 比如，grouping中的节点使用了when条件，如果when条件中使用了相对路径，就要求所有uses该grouping的地方针对该相对路径都是合法的才行。

12.1.7.11 规则：定义 rpc 时，不必在 output 中定义返回错误码和错误提示

说明：

很多rpc定义了output是为了返回错误码和错误提示。这是完全没有必要的，因为通信协议本身会返回错误码和错误提示的，不需要rpc自己定义，按照标准的协议处理机制，在出错时直接返回错误。

12.1.7.12 规则：对于有确定最大/最小条目数限制的 list 或 leaf-list，在 YANG 文件中必须要使用 max-elements/min-elements 方式指定条目数量规格

示例：

```
leaf-list number{
type string {
length "1..20";
      }
max-elements 5;
}
```

12.1.7.13 规则：废弃节点必须用 status 属性明确标识为 obsolete，不建议使用的节点必须用 status 属性明确标识为 deprecated，废弃节点或不建议使用的节点如果有替代节点，必须在描述中说明替代节点是什么

说明：

status属性用来标识该节点的状态，取值为current,deprecated或obsolete，含义如下：

1. **current**: 意味着这个定义是当前的、有效的。
2. **deprecated**: 表明了一个过时的定义，该节点仍然必须支持，用于兼容原有接口，但已经有新的替代节点的，建议采用新的替代节点。

版权所有 © 华为技术有限公司

3. **obsolete**: 意味着已经废弃，必须不再支持。

示例：

```
leaf my-leaf {
type unit8;
status obsolete;
description “xxx”; //如果有替代节点，要在描述中说明替代节点是什么
}
```

12.1.7.14 规则：YANG 文件中 import 和 include 语句中，禁止包含 revision-date 语句

说明：

如果某个软件版本支持同一个YANG文件的不同版本号，则在import和include时，需要使用revision-date语句来区分引用的是哪个版本号的YANG文件。但我们当前的软件版本同一个YANG文件只支持一个版本号，如果带上revision-date，当被引用的文件更新后，引用该文件的地方都要发生变更，非常不方便，所以要求import和include语句中，禁止包含revision-date语句。

示例：

```
import huawei-extension {
  prefix ext;
}
```

错误的例子：

```
import huawei-extension {
  prefix ext;
  revision-date 2019-03-11;
}
```

12.1.7.15 规则：文件中全部内容（包括标点符号）必须使用英文，不允许使用中文或其他语言。

说明：

无。

12.1.7.16 规则：版权信息必须按照统一要求定义。

说明：

版权信息Copyright在YANG文件头注释的地方定义，格式统一规定如下：

Copyright (C) XXXX-XXXX Huawei Technologies Co., Ltd. All rights reserved.

其中XXXX-XXXX为年份，要根据实际情况来定义：

（1）如果该YANG文件从2016年开始启动设计开发，到2017年最新一次正式发布，则Copyright (C)后面的年份为2016-2017。

（2）如果该YANG文件启动设计开发和最新一次正式发布的年份是都在2017年，则Copyright (C)后面的年份为2017，不能写为2017-2017。

示例：

 版权所有 © 华为技术有限公司

```
/*************************************************
 * Copyright (C) 2015-2017 Huawei Technologies Co., Ltd. All rights reserved.
 * Module description & tree structure: ·····
 *************************************************/
```

12.1.7.17 规则：organization、contact、revision 必须按照统一要求定义。

说明：

organization使用公司的标准英文名称，对应为”Huawei Technologies Co., Ltd.”

contact使用公司统一的地址，如下：

```
contact
"Huawei Industrial Base
 Bantian, Longgang
 Shenzhen 518129
 People's Republic of China
 Website: http://www.huawei.com
 Email: support@huawei.com
 "
```

revision部分是记录版本的修订历史时间，在各不同版本的说明中要增加变化的说明（如修改状态、增加限制、修改限制定义），同时必须遵守如下要求：

（1）YANG接口基线文件变更时，除第（2）点描述的特殊情况外，均需要新增revision说明，不允许基于最新的revision进行更新。同时，新增的revision说明必须放在所有revision的最前面（由新到旧排列），新增的revision日期必须晚于修改前的最新revision日期。

（2）特殊情况下，如果修改前的版本为内部版本不是正式发布（这种情况下，该版本基线文件不存在也不允许任何产品版本引用），则不需要新增revision说明，而是基于最新的revision进行更新，revision修订日期可以刷新或保持不变。例如：

2017年08月20日某开发人员修改接口文件并新增“revision 2017-08-20 {…}”说明，几天后08月24日发现接口存在问题并进行了再次修改，此时需要将“revision 2017-08-20 {…}”改为“revision 2017-08-24 {…}”，不需要新增revision说明，同时revision 2017-08-20历史版本的接口基线文件，不允许被正式发布。

（3）作为内部版本不会被正式发布的原因包括但不限于：存在语法错误、测试出存在严重问题、开发人员中间修改等。如果内部版本被误作为正式版本发布后，需要及时进行修改并进行升级或打补丁，修改时需要按照第（1）点要求，新增revision说明。

示例：

```
module huawei-sysname-subsysname {
// omit other part
revision 2014-10-26 {
    description
      "Add ** production adaption:
Deprecated ** item
Obsolete ** item
Add restriction on item **";
}

revision 2014-09-26 {
    description
      "Initial revision.";
}
}
```


版权所有 © 华为技术有限公司

12.1.7.18 规则：YANG 文件及其交互信息中禁止出现公司、部门、项目的标识及缩写，禁止出现芯片名称、型号。（文件名/模块名出现 huawei 字样除外）

说明：

随着公司的发展，会有其他公司OEM华为的产品，如果YANG文件或交互信息中使用了公司名称或者缩写，就需要修改，造成不必要的工作量，而部门、项目标识以及芯片的名称、型号容易造成泄密，也不能在YANG及其交互信息中出现。

12.1.7.19 规则：YANG 文件要通过 pyang 工具的规范检查校验。

说明：

YANG文件要通过pyang工具的规范检查校验，检验命令为：pyang xxx.yang --lint.

pyang工具的详细使用说明如下：

- pyang安装与使用指导：
  http://rnd-isourceb.huawei.com/netconf/xsd2YANG/wikis/pyang%E7%9A%84%E4%B8%8B%E8%BD%BD%E5%AE%89%E8%A3%85%E4%B8%8E%E4%BD%BF%E7%94%A8?suffix=md

12.1.7.20 规则：YANG 文件中 pattern 必须采用 XSD 类型正则表达式，隐式包含“以^开头和以$结尾”

说明：

RFC6020和RFC7950中要求YANG文件中使用XSD正则规范。说明如下：

The "pattern" statement, which is an optional substatement to the "type" statement, takes as an argument a regular expression string, as defined in [XSD-TYPES]. It is used to restrict the built-in type "string", or types derived from "string", to values that match the pattern.

XSD正则规范认为，“pattern”的正则表达式最常用的用法是匹配整个文本。XSD类型的正则表达式，隐式包含了“以^开头和以$结尾”，不需要显式定义。

YANG文件的“pattern”的正则表达式不允许使用"^" 表示以XXX开头，"$"表示以XXX结尾。如果“pattern”的正则表达式不需要约束以XXX开头或以XXX结尾，则在正则表达式的开头或结尾使用“.*”。

示例1：字符串类型叶子节点，用正则表达式使所有值必须以字符A (#x41) 开始，并以字符Z (#x5a) 结尾。

```
leaf name {
    type string {
        pattern "A.*Z";  //不允许写为“^A.*Z$”
    }
}
```

示例2：字符串类型的属性，值中某个位置必须包含至少3个连续A (#x41) 字符。

```
leaf name {
    type string {
        pattern ".*AAA.*";
    }
}
```


版权所有 © 华为技术有限公司

12.1.7.21 规则：如下节点不能作为 list 的 key：描述信息，密码，大小写不敏感的节点，数据可能拆分合并的复杂类型节点等

说明：

1. 描述信息是可修改的，不适合作为list的key。
2. 密码配置明文会回显密文，且密码可能发生修改，不适合作为list的key。
3. 大小写不敏感的节点，即使用扩展语法case-sensitivity取值为lower2upper或upper2lower的节点，因为存在大小写的转换，作为key可能造成第三方工具对象匹配失败。
4. 数据可能拆分合并的复杂类型节点也不适合作为list的key，比如类型为huawei-pub-type.yang中的id-range、vlan-range等类型的节点。

示例：

huawei-pub-type.yang中的vlan-range类型，基础类型是string，输入格式有两种：

1. 第一种，全量覆盖式，如'1,3,5-7'。
2. 第二种，增量修改式，如'+1,-3,+(20-30)'，可输入的数字范围为1-4094，假设原来配置了'1,3'，再增量配置'+2'，则最后的结果为'1-3'，对于此类数据会拆分合并的类型节点，不适合作为list的key。

12.1.7.22 规则：YANG 文件中节点的 description 要清晰、准确、完整、规范的表述该节点的具体含义

说明：

YANG文件中节点的description分为两部分。

1. 第一部分为第一句话，第一句话必须且仅描述该节点的具体含义，含义表述要清晰准确完整，且长度不超过255个字符。第一句话以英文句号“.”结尾，结尾后可直接换行或用空格分割下一句话。（MD-CLI会提取该第一句话作为CLI中节点的帮助信息）
2. 第二部分为除第一句话外的其它内容。描述该节点无法用语法表达的特殊约束，使用场景，注意事项等信息。第二部分可以没有。

对description内容的要求：

1. 不能出现命令行、MIB相关的专有术语，比如view，undo。第一部分的description不能出现YANG相关的专有术语，比如，yang，schema node等。
2. 对于有特殊格式要求的参数，要有输入样例。
3. 要从业务的角度进行描述，尽量不要直接写xpath路径，更不能直接写xpath表达式，比如count(../xxx)，可以改为英文语法：the count of ...。

示例：YANG关键字为container，在list上层带s的container场景下，xxx来自于list节点的描述。

```
container discriminator-maps {
   description "List of match relation between SBFD discriminator and IP";
   list discriminator-map {
      description  "Match relation between SBFD discriminator and IP";
   }
```


版权所有 © 华为技术有限公司

12.1.7.23 规则：对于密码类型的处理原则

说明：

考虑到安全要求，不能支持密码明文展示。因此，针对密码类型字段有以下的处理：

1. 对于新开发的YANG模型，不再支持密码类型字段，只有密码字段，密码必须加密保存，用户get/get-config时统一返回加密后的密文。
2. 对于存量YANG模型，在密码类型字段上标注为deprecated(待废弃)，为了兼容性考虑，支持密码类型为simple方式的配置下发，但是统一加密保存，查询时返回密文，密码类型为cipher。

12.1.7.24 规则：presence container 的使用原则

说明：

Presence container是有实际配置含义的实体，这里的有实际配置含义不仅仅代表该container本身有意义，更多地是指该container及其下面的子树作为一个整体有实际配置含义，从这个意义上来说，presence container类似单实例的list。

符合以下情况之一需要使用presence container来表达：

1. container本身有配置含义，如使能/去使能某个功能。
2. container下的子节点有必选节点，但是对于container的最近的有意义的祖先节点（list/presence container）来说却不是必选的，此时该container应为presence container。
3. container下的子节点有默认值，但是对于container的最近的有意义的祖先节点（list/presence container）来说却不是默认值（属于可有可无的），此时该container应为presence container。

示例1：presence container代表使能/去使能

```
container bras-diameter {
  description
    "Diameter.";
  container diameter {
    presence "Enables the Diameter protocol.";
    description
      "Configure the Diameter protocol.";
  }
}
```

示例2：container下存在mandatory节点的情况

```
    list static-instance {
      key "name";
      description
        "Configure static instance of IFIT.";
      leaf name {
        type string {
          length "1..31";
        }
      }
      container flow {
        presence "Enable flow";
        description
          "Configure IFIT flow.";
        leaf flow-type {
          ext:operation-exclude update;
```


版权所有 © 华为技术有限公司

```
            type ifit-common:flow-type;
            mandatory true;
            description
              "Flow type of an IFIT flow.";
          }
        }
      }
```

示例3：container下有默认值的情况

```
container global {
    container reconfigure {
        presence "Create DHCPv6 reconfigure policy.";
        description
          "Enable/disable DHCPv6 reconfigure policy.";
        leaf retransmit-times {
            type uint8 {
                range "1..8";
          }
            default "8";
            description
                "Set the number of retransmission times of DHCPv6 reconfigure packets sent to
users.";
        }
    }
}
```

12.2 Python 开发规范

12.2.1 排版

12.2.1.1 规则：程序块采用 4 个空格缩进风格编写

说明：程序块采用缩进风格编写，缩进的空格数为4个，是业界通用的标准。

12.2.1.2 规则：Python 文件必须使用 UTF-8 编码

说明：Python文件必须使用UTF-8编码，文件头可以添加编码声明 # coding: utf-8

12.2.1.3 规则：一行只写一条语句

说明：不允许把多个短语句写在一行中，即一行只写一条语句。多条语句写在一行，这样做一个很明显得缺点就是在调试的时候无法单步执行。

12.2.1.4 规则：相对独立的程序块之间、变量说明之后必须加空行

说明：相对独立的程序块之间、变量说明之后加上空行，代码可理解性会增强很多。

正确示例：

```
if len(deviceName) < _MAX_NAME_LEN:
...

writer = LogWriter()
```


版权所有 © 华为技术有限公司

12.2.1.5 规则：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符前后要加空格

说明：采用这种松散方式编写代码的目的是使代码更加清晰。在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。给操作符留空格时不要连续留一个以上空格。

1. 逗号、分号（假如用到的话）只在后面加空格。
   ```
   print(a, b, c)
   ```
2. 比较操作符">"、">="、"<"、"<="、"==”，赋值操作符"="、"+="，算术操作符"+"、"-"、"%"，逻辑 操作符"and"、"or"等双目操作符的前后加空格。
   ```
   a = b + c
   a += 2
   if current_time >= MAX_TIME_VALUE:
   ```
3. "*"、"**"等作为操作符时，前后可以加空格，但若和更低优先级的操作符同时使用并且不涉及括 号，则建议前后不加空格。
   ```
   a = b * c
   a = c ** b
   x = x*2 - 1
   ```

12.2.1.6 规则：加载模块必须分开每个模块占一行

说明：单独使用一行来加载模块，让程序依赖变得更清晰。

```
import sys
import os
```

注意虽然一行只能加载一个模块，但同一个模块内的多个符号可以在同一行加载。

```
from sys import stdin, stdout
```

12.2.1.7 规则：导入部分(imports)置于模块注释和文档字符串 之后，模块全局变量和常量声明之前

说明：导入模块的部分应置于模块注释和文档字符串之后，模块全局变量和常量声明之前，使程序读起来更有条理，更清晰。

示例：

```
"""
功能：XXX类，该类主要涉及XXX功能
版权信息：华为技术有限公司，版本所有(C) 2010-2019
"""

import sys
import os

showList = []
```

12.2.2 注释

 版权所有 © 华为技术有限公司

12.2.2.1 规则：类和接口的文档字符串写在类声明(class ClassName:)所在行的下一行，并向后缩进 4 个空格

说明：类和接口的文档字符串的内容可选择包括（但不限于）功能描述，接口清单等。功能描述除了描述类或接口功能外，还要写明与其他类或接口之间的关系；接口清单列出该类或接口的接口方法的描述。

```
class TreeError(libxmlError):
    """
    功能描述:
    接口:
    """
```

12.2.2.2 规则：公共函数的文档字符串写在函数声明(def FunctionName(self):)所在行的下一行，并向后缩进 4 个空格

说明：公共函数文档字符串的内容可选择包括（但不限于）功能描述、输入参数、输出参数、返回值、调用关系（函数、表）、异常描述等。异常描述除描述函数内部抛出的异常外，还必须说明异常的含义及什么条件下抛出该异常。

```
def load_batch(fpath):
    """
    功能描述:
    参数:
    返回值:
    异常描述:
    """
```

12.2.2.3 规则：公共属性的注释写在属性声明的上方，与声明保 持同样的缩进。行内注释应以#和一个空格作为开始，与后面的文字注释以一个空格隔开

说明：行内注释的形式是在语句的上一行中加注释。行内注释要少用。它们应以#和一个空格作为开始。

```
# Compensate for border
x = x + 1
```

12.2.2.4 规则：模块文档字符串写在文件的顶部，导入(import) 部分之前的位置，不需要缩进

说明：模块文档字符串应当包含功能描述和版权声明。 正确示例：

```
"""
功 能：XXX类，该类主要涉及XXX功能
版权信息:
"""
```

12.2.2.5 规则：文档字符串多于一行时，末尾的"""要自成一行

说明：对于只有一行的文档字符串，把"""放到同一行也没问题。

```
"""Return a foobang
Optional plotz says to frobnicate the bizbaz first.
"""

"""API for interacting with the volume manager."""
```

 版权所有 © 华为技术有限公司

12.2.2.6 规则：注释必须与其描述的代码保持同样的缩进，并放在其上方相邻位置

示例：

```
if image_service is not None:
    # Deletes the image if it is in queued or saving state
self._delete_image(context, image_meta['id'], image_service)
```

12.2.3 命名

12.2.3.1 规则：包（Package）、模块（Module）名使用意义完 整的英文描述，采用小写加下划线 （lower_with_under）的风格命名

说明：模块应该用小写加下划线的方式（如lower_with_under.py）命名。尽管已经有很多现存的模块使用类似于CapWords.py这样的命名,但现在已经不鼓励这样做, 因为如果模块名碰巧和类名一致, 这会让人困扰。

```
from sample_package import sample_module
from sample_module import SampleClass
```

12.2.3.2 规则：类（Class）名使用意义完整的英文描述，采用大 写字母开头的单词（CapWords）风格命名

说明：类沿用面向对象语言最常用的CapWords风格命名。

```
class SampleClass(object):
pass
```

12.2.3.3 规则：函数（Function）、方法（Method）、函数参数（Function Parameters）名使用意义完整的英文描述，采用小写加下划线（lower_with_under）的风格命名

说明: 函数、方法采用小写加下划线的风格命名，与类名做区分。函数参数采用小写加下划线的风格命名，与一般变量的命名风格保持一致。模块内部使用的函数用单下划线(_)开头，表示函数是protected 的(使用from module1 import *时不会包含)。

```
def sample_public_function(sample_parameter):
pass

def sample_internal_function(sample_parameter):
pass

class SampleClass(object):
    def sample_member_method(self, sample_parameter):
        pass
```

12.2.3.4 规则：变量（variable）采用小写加下划线 （lower_with_under）的风格命名。常量（constant） 采用大写加下划线（CAPS_WITH_UNDER）的风格命名

说明: 常量使用大写加下划线的风格命名，与变量做区分。

```
sample_global_variable = 0
M_SAMPLE_GLOBAL_CONSTANT = 0

class SampleClass(object):
    SAMPLE_CLASS_CONSTANT = 0
```


版权所有 © 华为技术有限公司

```
    def sample_member_method(self, sample_parameter):
        pass

def sample_function():
    sample_function_variable = 0
    sample_instant_variable = SampleClass()
```

12.2.3.5 规则： 类或对象的私有成员一般用单下划线_开头；对于 需要被继承的基类成员，如果想要防止与派生类成员重名，可用双下划线__开头

说明： Python没有严格的私有权限控制，业界约定俗成的用单下划线“_”开头来暗示此成员仅供内部使用。双下划线“__”开头的成员会被解释器自动改名，加上类名作为前缀，其作用是防止在类继承场景中出现名字冲突，并不具有权限控制的作用，外部仍然可以访问。双下划线开头的成员应当只在需要避免名字冲突的场景中使用（比如设计为被继承的工具基类）。

```
class MyClass:
    def my_func(self):
        self._member = 1 # 单下划线开头，暗示此成员仅供类的内部操作使用，外部不应该访问。

    def _my_private_func(self): # 单下划线开头，暗示此方法仅供类的内部操作使用，外部不应该访问。
        pass

class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable) # 双下划线开头，会被解释器改名为_Mapping__update。外部如果使用修改后的名字仍可访问

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)

    __update = update # 作为update方法的私有复制成员，不会跟派生类成员重名

class MappingSubclass(Mapping):
    # 和基类同名方法，修改了参数个数，但是不会影响基类__init__
    def update(self, keys, values):
        for item in zip(keys, values):
            self.items_list.append(item)

__update = update # 被解释器改名为_MappingSubclass__update，不会跟基类成员重名
```

12.2.4 编码

12.2.4.1 规则：避免在无关的变量或无关的概念之间重用名字， 避免因重名而导致的意外赋值和错误引用

说明： Python的函数/类定义和C语言不同，函数/类定义语句实际上是给一个名字赋值。因此重复定义一个函数/类的名字不会导致错误，后定义的会覆盖前面的。但是重复定义很容易掩盖编码问题，让同一个名字的函数/类在不同的执行阶段具有不同的含义，不利于可读性，应予以禁止。

Python在解析一个被引用的名字时遵循LEGB顺序（Local - Enclosed - Global - Builtin），从内层一直查找到外层。内层定义的变量会覆盖外层的同名变量。在代码修改时，同名的变量容易导致错误的引用，也不利于代码可读性，应当尽量避免。

 版权所有 © 华为技术有限公司

12.2.4.2 规则：类的方法不需访问实例时，根据具体场景选择使 用@staticmethod 或者@classmethod 进行装饰

说明： 一般的类方法要接收一个self参数表示此类的实例，但有些方法不需要访问实例，不需要访问任何成员，或者只需要显式访问这个类自己的成员。这样的方法不需要额外参数，应当用@staticmethod装饰。 在Python 3.X版本中，允许直接定义不含self参数的方法，并且允 许不通过实例调用。但是一旦通过实例调用这个方法，就会因为参数不匹配而出错。 加上 @staticmethod进行修饰，可以让Python解释器明确此方法不需要self参数，提前拦截问题，可读性也 更好。

12.2.5 异常处理

12.2.5.1 规则：使用 try…except…结构对代码作保护时，需要在异常后使用 finally…结构保证操作对象的释放

说明： 使用try…except…结构对代码作保护时，如果代码执行出现了异常，为了能够可靠地关闭操作对象，需要使用finally…结构确保释放操作对象。

12.2.5.2 规则：不要使用“except:”语句来捕获所有异常

说明： 在异常这方面, Python非常宽容,“except:”语句真的会捕获包括Python语法错误在内的任何错误。使用 “except:”很容易隐藏真正的bug，我们在使用try…except…结构对代码作保护时，应该明确期望处理的异常。 Exception类是大多数运行时异常的基类，一般也应当避免在except语句中使用。通常，try只应当包含必须要在当前位置处理异常的语句，except只捕获必须处理的异常。比如对于打开文件的代码， try应当只包含open语句，except只捕获FileNotFoundError异常。对于其他预料外的异常，则让上层函数捕获，或者透传到程序外部来充分暴露问题。

12.2.5.3 规则：不在 except 分支里面的 raise 都必须带异常

说明： raise关键字单独使用只能出现在try-except语句中，重新抛出except抓住的异常。

12.2.6 并发与并行

12.2.6.1 规则：多线程适用于阻塞式 IO 场景，不适用于并行计算场景

12.3 其他注意事项

12.3.1 devicemgr.query_neid(nename)不推荐使用，推荐使用 devicemgr.query_neids_by_operatename(operatename)

版权所有 © 华为技术有限公司

12.3.2 devicemgr.query_status(neid='', nename='', nemanageip='')不推荐使用，推荐使用 devicemgr.query_status_by_neid(neid)

版权所有 © 华为技术有限公司

13 附录

13.1 业务安全
13.2 CLI扩展项
13.3 Jinja2模板过滤器及方法

13.1 业务安全

下面描述开放可编程业务安全注意事项：

- 映射模板定制：

  支持nodelete标签。nodelete标签用于场景：当业务删除或者更新时会触发分解删除设备配置，该标签会决定删除设备配置是否执行。

  用户可以基于业务逻辑打上“no_delete”或“no_delete_nested”属性，标记该属性的数据是不会随着业务实例的删除删除掉。

  - no_delete：表示打上标记的本层节点不会被删除，子节点是可以被删除的。
  - no_delete_nested：表示打上标记的本层节点以及子节点都不会被删除。

  该标签不会删除数据，但会删除数据源。数据源（配置源）：SSP包中的SSP-实例1 分解接口配置interface 0/0/1和里面的接口属性attr1（value1）和attr2（value2），在网元配置interface 0/0/1以及属性attr1和attr2上面会打上SSP-实例1的path作为标记，该标记成为数据源。
- 数据一致性定制：如果用户定制了对账可删除的开关，就会在数据对账时，删除设备配置。
- 对账时是否支持删除转发器的配置，true值为支持，false值为不支持。

  例子：

  ```
  syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
  ```
- 回退逻辑：
  - 支持事务的设备回退能力：利用设备的事务能力，执行命令行报错时，执行cancel命令行（cancel命令行可以在SND包getCliDriverInfo中配置）。
  - 不支持事务的设备回退能力：根据设备的返回结果，识别出已经执行成功的命令行；将下发成功的命令行生成逆向报文下发给设备。

 版权所有 © 华为技术有限公司

13.2 CLI 扩展项

13.2.1 基础扩展项简表

表 13-1 基础扩展项 annotations

扩展项 annotations	作用节点	方向	功能描述	使用场景
insert-before(after) "xxx"	leaf	双向	将非key的leaf名称和值跟在list名后，并且通过before/after和"xxx"来指定其位置。"xxx"指其他的leaf名称。	场景1：指定某一个非key leaf显示在创建list的命令行中某一个key的前面。 场景2：指定某一个非key leaf显示在创建list的命令行中某一个key的后面。 场景3：指定某一个非key leaf显示在之前已经插入的非key leaf前后。
render-view true|false	container/list	双向	指定container/list是否进入视图。 list：默认情况下list是要进入视图的，如果加上render-view false，则抑制list进入视图。 container：默认情况下会将container节点名作为一个命令前缀，加上render-view true后，可以使container进入子视图。	场景1：抑制list进入子视图，使list下非key的leaf节点直接加在key节点之后。 场景2：无法用list建模，又需要进入子视图的场景。
render-name true|false	container/ leaf/ leaf-list	双向	指定节点的名字在CLI命令中显示或者不显示。	场景1：通过该扩展将container节点名称忽略掉。 场景2：配置命令中不需要节点名称，只需要参数值的场景。 场景3：需要将list的key节点名称显示出来的场景。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
optional-space	list/leaf	CLI->YANG	list：允许list名称和list的key合并，可以不用空格隔开。 leaf：允许leaf(非key)名称和其value合并，可以不用空格隔开。	场景1：针对list，逆向解析时，可以区分没有空格的场景。 场景2：针对接口等leaf节点，逆向解析时，可以区分没有空格的场景。
render-undo-value true|false	leaf	YANG->CLI	true：对于删除操作，生成的删除命令要包含leaf的值。 false：对于删除操作，生成的删除命令不包含leaf的值。 三方包中如果配置项defaultDeleteOperationWithValue项配置为了true，则生成的删除命令默认会包含leaf值。否则，删除命令默认不会包含leaf值。	场景1：在华为样式设备中，no l1 <string>，删除一个leaf节点，需要加上leaf本身的值，否则设备无法删除。 场景2：对于带值删除的设备（如Tellabs设备），如果期望删除生成的删除命令不含leaf值，则可以通过添加该annotation实现。
render-undo	leaf	双向	默认情况下<leaf>true</leaf>会转换成leaf true命令，<leaf>false</leaf>会转换成leaf false命令，加上render-undo之后，便会转换成leaf和no leaf命令。 不能与insert-before/after或render-single-line叠加使用。	场景1：对于数据类型是boolean的leaf节点，通过该扩展项可以生成leaf 和 no leaf命令，而不是 leaf true 和 leaf false。
render-undo-before	container/list/leaf / leaf-list	YANG->CLI	render-undo-before告诉CLI引擎，删除时，在元素前面嵌入no，而不是在行首。	场景1：一般在类Juniper样式(长命令)的设备中比较常见，router interface <name> [no] port，在华为样式(短命令)的设备使用的很少。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-nested-delete true|false	container/list	YANG->CLI	true：先递归删除容器节点中的子节点，再删除容器节点本身。 false：抑制递归删，直接删除容器，通常父层节点使用render-nested-delete true扩展字导致该容器需要递归删除。 **注意** 高危操作，请慎重使用。	场景1：先递归删除容器下的子节点，再删除容器自身。 场景2：抑制递归删，不删除子节点，直接删除自身。
quit-view "xxx"	container/list	双向	修改默认退出视图命令的行为，提示CLI转换引擎在当前的子视图中用引号中的退出命令替换默认退出命令。 "xxx"指退出子视图的具体命令。	场景1：用于退出子视图的命令不是默认退出子视图命令的场景。
allow-multi-token xx	leaf	CLI->YANG	支持leaf携带多个字符串参数，用空格隔开，数目不超过指定的参数(xx)。 若参数为-1，则表明参数数目无限制。	场景1：设备的某些配置命令的命令元素是多个字符串，用空格隔开。 场景2：设备的某些创建视图配置命令的命令元素是多个字符串，用空格隔开。
update-by-delete	leaf/leaf-list/list	YANG->CLI	设备上有些配置需要先删除后才允许更新。该annotation主要就是为了解决此类问题，它告诉CLI转换引擎，在生成更新命令之前，需要先生成一条删除命令。 涉及删除，请谨慎使用。	场景1：某些配置不支持直接更新，必须先删除再创建。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-single-line	container	双向	render-single-line将container中的所有leaf节点转换成一行命令，而不是每个leaf一行。 不能和render-undo叠加使用。 使用该扩展字的容器，其不应该包含会进入视图的节点。 子container不能是presence container。	场景1：包含多个参数的命令。
render-multi-line	leaf-list/list	双向	leaf-list: 默认leaf-list有多个值的情况下，会生成一条多个参数的命令，该扩展字会使每个值都生成一条命令。 list: 抑制list进入视图，与render-view false不同，key leaf后只能跟单个非key leaf，若有多个非key leaf，则分行。	场景1：对于list期望不进子视图、每个非key的leaf单独生成一条命令。 场景2：对于leaf-list，期望leaf-list的每个leaf值生成一条命令。
render-sequential-order	container	CLI->YANG	主要是告诉CLI解析引擎节只接受container顺序与数据模型中出现的顺序相同的leaf，一般跟render-name false一起使用，只有value，没有name的情况。 需要配合render-single-line使用，且两个扩展字的顺序固定，render-single-line在render-sequential-order上方。	场景1：命令行多参数不带leaf name场景，解析命令行时需要指定参数对应的leaf顺序。
render-optional	container	CLI->YANG	配合render-sequential-order一起使用。render-sequential-order表明该容器里元素必须按照正确的节点顺序出现；若有子元素使用render-optional扩展字，则该元素可以跳过。	场景1：命令行多参数场景，配置render-sequential-order命令行需要按照一定的顺序出现，但存在部分非必选参数。如c1 [l1 <string>] l2 <string> l3 <string> l4 <string>，命令行节点l1为非必选参数，需要使用render-optional进行说明。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-sequential-order-until "name"	container	CLI ->YANG	通过指定的子元素"name"来分割容器子leaf元素，前一部分必须按顺序解析，后一部分可以无序。	场景1：一容器的前一部分子元素需要有序，后一部分可以无序。
add-suffix-on-create "xxx"	list	双向	在创建list视图时，添加该annotation，会在命令中list那一行末尾加上指定的参数"xxx"，表明这是第一次创建。 只有创建时才会生效，更新不生效。	场景1：在创建视图时，如果list有该annotation，则会指明这是创建视图。
replace-value "xxx,yyy"	leaf	YANG->CLI	在设置leaf值时，如果值是"xxx"，则用"yyy"代替。	场景1：当我们在配置参数时，不方便直接指定leaf值时，可以用该扩展字去替换leaf的值
delete-siblings-on-delete	container/list/leaf/leaf-list	YANG->CLI	当删除某一使用了该扩展字的节点时，则会告诉上游系统去删除控制器中该节点的所有兄弟节点的数据。	场景1：当删除命令行部分参数时期望同时重置兄弟节点的参数，可以用该扩展字。
delete-parents-on-delete	leaf	YANG->CLI	删除某一带有该扩展字节点时，则会告诉上游系统，删除该节点的父节点数据。 只能是leaf节点，且有父节点。	场景1：当删除命令行某个参数时期望同时删除整个父节点，可以用该扩展字。

 版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-undo-name true|false	container/leaf/list	YANG->CLI	在删除操作时，不生成节点的节点名。	场景1：特定设备删除命令不能加leaf节点的名称，否则报错。Yang模型：父节点container，子节点leaf(多个)，container下使用render-single-line，leaf下使用render-undo-name和render-undo-value；进行删除container操作。 场景2：在递归删除一个container/list时，默认转换规则会先递归生成该container下的子节点的删除命令，然后再生成一条该container本身的删除命令。此时，如果不希望生成该container本身的删除命令，可以通过添加该annotation实现。
delete-by-set-default-value	leaf	YANG->CLI	在对设置有默认值的leaf节点进行删除时，生成update为默认值命令行。	场景1：删除某个参数期望重置为默认值。
delete-no-support	container/leaf	YANG->CLI	对带有该annotation的节点进行删除时，生成空命令行。	场景1：某些命令不支持删除，可以使用该扩展项。
leaf-dependency "xxx,xxx"	container/list	YANG->CLI	当container或list下节点需要依赖其他节点一起更新的时候，使用该annotation。 xxx指container或list下的节点名。参数为all时，代表container或list下所有节点都要依赖其他节点一起更新。	场景1：多参数命令行，当更新其中部分参数时，设备的命令行包含所有参数。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-flat-container	container	双向	render-flat-container将配置render-single-line的container的所有子container、所有leaf节点跟父节点转换成一行命令。 不能和render-undo叠加使用。使用该扩展字的容器，其不应该包含会进入视图的节点。	场景1：container有子container，期望container及子container所有节点都在一条命令行上，可以使用该扩展项。
render-container-together	presence container	双向	presence container和leaf节点不分行展示。	场景1：presence container不期望单独生成presence container的命令。
render-identifier	container/list/leaf/leaf-list	双向	生成/解析命令行使用指定的标识符替换定义的yang标识符名称。	场景1：可以使不符合yang规范的标识符作为命令行参数，如命令关键字8021p、6to4。
render-undo-name-and-value	leaf	YANG->CLI	合并render-undo-name和render-undo-value，生成的删除命令包含/不包含leaf的名字和值	场景1：用于undo命令行不需要对应配置项的名字和值场景。
render-order-dependency	container/leaf	YANG->CLI	指定下发到设备的命令行顺序，适用于所有操作类型。有5个子annotations可以指定适用的操作类型：operation-create,operation-update,operation-delete,operation-all,operation-create-delete，默认是operation-all。	场景1：对下发设备命令行有顺序要求的场景。
delete-on-empty	container	YANG->CLI	leaf节点全部删除的时候，联动删除数据库的container。	场景1：设备命令行参数全删除后，该命令行也会删除，AOCDB还有presence container，也需要删除，否则差异比较不一致。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
reset-all-sibling	使用render-sequential-order的container	YANG->CLI	对leaf节点delete and update的时候会联动reset container下所有其他的leaf，有默认值的reset为默认值，没有的置空。	场景1：当删除命令行部分参数时期望同时重置所有兄弟节点的参数，可以用该扩展字。
reset-siblings	使用render-sequential-order-until的container	YANG->CLI	对leaf节点delete and update的时候会联动reset container下render-sequential-order-until指定的leaf，有默认值的reset为默认值，没有的置空。	场景1：当删除命令行部分参数时期望同时重置部分兄弟节点的参数，可以用该扩展字。
cli-case-sensitive/insensitive	container/list/leaf/leaf-list	CLI->YANG	1、当解析CLI命令行时，Yang模型匹配节点名称是否大小写敏感。默认大小写敏感。 2、配置有cli-case-insensitive的节点作用范围是整个子树，可用cli-case-sensitive打破作用范围。	场景1：设备上的命令行关键字不区分大小写。
render-hide-in-view	带有render-view true扩展项的Container下的leaf	双向	用于设置leaf节点作为进入子视图的语句。	场景1：container需要加子leaf作为进子视图的参数。
reset-container	container/list/leaf	YANG->CLI	1.当配置于container，list节点时，当删除或者更新其直接子时，其余的子节点需要被重置。如果有默认值，重置为默认值，否则置空。 2.当配置于leaf节点时，当删除或者更新该叶子节点时，其兄弟节点需要被重置。 不能与render-nested-delete或delete-parents-on-delete叠加使用。	场景1：当删除命令行部分参数时期望同时重置所有兄弟节点的参数，可以用该扩展字。

版权所有 © 华为技术有限公司

扩展项 annotations	作用节点	方向	功能描述	使用场景
render-batch-list-syntax	只包含key leaf的list	双向	将数据库存的离散值整合为范围的命令行。 注意：一次事务中如含有不同的操作类型，则不会生效。	场景1：适用于vlan batch 1 to 5 范围的命令行。
incomplete-command	container/leaf	YANG->CLI	创建/更新操作使用，用于leaf时， 确保使用该扩展项的leaf不作为最后的配置节点；用于container时，需要用于presence container、且使用了render-single-line扩展项，确保命令行presence container不作为最后的配置项。	场景1：命令行某些参数创建、更新不能作为最后一个参数。
incomplete-no	leaf	YANG->CLI	用于delete操作， 确保使用该扩展项的leaf不作为最后的配置节点。	场景1：命令行某些参数删除时不能作为最后一个参数。
render-order-by-user	配置有ordered-by user的list节点	YANG->CLI	对于生成list实例有顺序要求的，在生成命令行时需要将配置先删除再重新新增。	场景1：对于SND包中的yang，如果有节点设置order-by-user属性，并且生成list实例有顺序要求，在生成命令行时需要将配置先删除再重新新增，可以在该节点使用render-order-by-user扩展项，位置信息需要从SSP包分解到SND包。

13.2.2 基础扩展项详细介绍

13.2.2.1 insert-before(after) "xxx" leaf

名称	描述
标签名	insert-before(after) "xxx"
作用节点	leaf
父节点	list
支持方向	双向

版权所有 © 华为技术有限公司

名称	描述
支持数据类型	无限制
是否可继承	否
功能描述	list的默认规则时list名后紧跟该list的key，然后进入子视图，如果期望将非key的leaf跟在list名后，可以通过该annotation实现，并且通过before/after和"xxx"来指定其位置
使用场景	场景一：指定某一个非key leaf显示在创建list的命令行中某一个key的前面 场景二：指定某一个非key leaf显示在创建list的命令行中某一个key的后面 场景三：指定某一个非key leaf显示在之前已经插入的非key leaf前后
备注	xxx指定leaf中的某个key或者普通leaf，若是普通leaf则也需要使用该扩展字

13.2.2.1.1 使用场景举例

场景一：命令行里，指定一个leaf在某个key前显示

YANG模型：使用insert-before	YANG模型：不使用insert-before

list l1 {
key "k1 k2"
leaf k1 {
type string;
}
leaf k2 {
type string;
}
leaf v1 {
hw:insert-before 'k2';
type string;
}
}
	list2 l2 {
	key "k1 k2"
	leaf k1 {
	type string;
	}
	leaf k2 {
	type string;
	}
	leaf v1 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

CLI to YANG结果 输入CLI 命令： l1 test1 v1 abcd test2\n 预期结果： <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data>	CLI to YANG结果 输入CLI 命令： l1 test1 test2\n v1 abcd\n 预期结果： <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data>

CLI to YANG结果 YANG Path:/l1 操作类型：create YANG Data: <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data> 预期结果： l1 test1 v1 abcd test2\n	CLI to YANG结果 YANG Path: /l1 YANG Data: <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data> 预期结果： l1 test1 test2\n v1 abcd\n

场景二：命令行里，指定一个leaf在某个key后显示

 版权所有 © 华为技术有限公司

YANG模型：使用insert-after	YANG模型：不使用insert-after

list l1 {
key "k1 k2"
leaf k1 {
type string;
}
leaf k2 {
type string;
}
leaf v1 {
hw:insert-after 'k2';
type string;
}
}
	list2 l2 {
	key "k1 k2"
	leaf k1 {
	type string;
	}
	leaf k2 {
	type string;
	}
	leaf v1 {
	type string;
	}
	}

CLI to YANG结果	CLI to YANG结果
输入CLI 命令： l1 test1 test2 v1 abcd\n 预期结果： <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data>	输入CLI 命令： l1 test1 test2\n v1 abcd\n 预期结果： <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data>

 版权所有 © 华为技术有限公司

CLI to YANG结果 YANG Path:/l1 操作类型：create YANG Data: <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data> 预期结果： l1 test1 test2 v1 abcd\n	CLI to YANG结果 YANG Path: /l1 YANG Data: <data> <l1> <k1>test1</k1> <k2>test2</k2> <v1>abcd</v1> </l1> </data> 预期结果： l1 test1 test2\n v1 abcd\n

场景三：命令行里，一个普通leaf插入在另一个普通leaf前后

YANG模型：使用insert-before(after)插入在普通leaf前后
list l1 {
key k1;
leaf k1 {
type string;
}
leaf v1 {
hw:insert-before 'k1';
type string;
}
leaf v2{
hw:insert-before "v1";
type string;
}
leaf v3{
hw:insert-after "v1";
type string;
}
}
	YANG模型：不使用insert-before(after)插入在普通leaf前后
	list2 l1 {
	key "k1 k2"
	leaf k1 {
	type string;
	}
	leaf v1 {
	hw:insert-before 'k1';
	type string;
	}
	leaf v2 {
	type string;
	}
	leaf v3 {
	type string;
	}
	}

 版权所有 © 华为技术有限公司

CLI to YANG结果 输入CLI 命令： l1 v2 value2 v1 value1 v3 value3 key1\n exit\n 预期结果： <data> <l1> <k1>test1</k1> <v1>value1</v1> <v2>value2</v2> <v3>value3</v3> </l1> </data>	CLI to YANG结果 输入CLI 命令： l1 v1 value1 key1\n v2 value2\n v3 value3\n exit\n 预期结果： <data> <l1> <k1>test1</k1> <v1>value1</v1> <v2>value2</v2> <v3>value3</v3> </l1> </data>

CLI to YANG结果 YANG Path:/l1 操作类型：create YANG Data: <data> <l1> <k1>key1</k1> <v1>value1</v1> <v2>value2</v2> <v3>value3</v3> </l1> </data> 预期结果： l1 v2 value2 v1 value1 v3 value3 key1\n exit\n	CLI to YANG结果 YANG Path: /l1 YANG Data: <data> <l1> <k1>key1</k1> <v1>value1</v1> <v2>value2</v2> <v3>value3</v3> </l1> </data> 预期结果： l1 v1 value1 key1\n v2 value2\n v3 value3\n exit\n

13.2.2.2 render-view true|false container/list

名称	描述
标签名	render-view true|false
作用节点	container/list

版权所有 © 华为技术有限公司

名称	描述
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	指定container/list是否进入视图。 list：默认情况下list是要进入视图的，如果加上render-view false，则抑制list进入视图。 container：默认情况下会将container节点名作为一个命令前缀，加上render-view true后，可以使container进入子视图。
使用场景	场景一：抑制list进入子视图，使list下非key的leaf节点直接加在key节点之后 场景二：无法用list建模，又需要进入子视图的场景
备注	NA

13.2.2.2.1 使用场景举例

场景一

YANG模型：list下使用render-view	YANG模型：list下不使用render-view

list l2 {
hw:render-view "false";
key k1;
leaf k1 {
type string;
}
leaf v1 {
type string;
}
leaf v2 {
type string;
}
}
	list l2 {
	key k1;
	leaf k1 {
	type string;
	}
	leaf v1 {
	type string;
	}
	leaf v2 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果 Path:/l1 Data: <data> <l2> <k1>test</k1> <v1>test1</v1> <v2>test2</v2> </l2> </data> 操作类型：create 预期结果： l2 test v1 test1 v2 test2 解析：v1和v2作为非key节点，当list上加入了render-view false后，他们对应的命令直接加在l2 test的尾部，而不是进入子视图。	YANG to CLI结果 Path:/l1 Data: <data> <l2> <k1>test</k1> <v1>test1</v1> <v2>test2</v2> </l2> </data> 操作类型：create 预期结果： l2 test v1 test1 v2 test2 解析：v1和v2作为非key节点，默认转换逻辑是进入子视图后，每个leaf都单独生成一条命令。

场景二

Yang模型：container下使用render-view
container router {
container rsvp {
hw:render-view "true";
leaf hello-timeout {
type uint16;
}
}
}
	Yang模型：container不使用render-view
	container router {
	container rsvp {
	leaf hello-timeout {
	type uint16;
	}
	}
	}

 版权所有 © 华为技术有限公司

Path:/router Data: <router> <rsvp> <hello-timeout>5</hello-timeout> </rsvp> </router> 操作类型：create 生成的CLI命令如下： router rsvp hello-timeout 5	Path:/router Data: <router> <rsvp> <hello-timeout>5</hello-timeout> </rsvp> </router> 操作类型：create 生成的CLI命令如下： router rsvp hello-timeout 5

13.2.2.3 render-name true|false container/leaf/leaf-list

名称	描述
标签名	render-name true|false
作用节点	container/ leaf/leaf-list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	指定节点的名字在CLI命令中显示或者不显示。
使用场景	场景一：根节点为leaf的，目前ODL不支持，需要在外面套一层container，然后通过该扩展将container节点名称忽略掉 场景二：配置命令中不需要节点名称，只需要参数值的场景 场景三：需要将list的key节点名称显示出来的场景
备注	NA

13.2.2.3.1 使用场景举例

场景一：

 版权所有 © 华为技术有限公司

Yang模型：使用render-name
container routerTest{
hw:render-name "false";
leaf router-id{
type inet:ipv4-address;
}
}
	Yang模型：不使用render-name或者render-name true
	container routerTest{
	leaf router-id{
	type inet:ipv4-address;
	}
	}

Path:/routerTest Data: <routerTest> <router-id>10.2.3.4</router-id> </routerTest> 操作类型：create 生成的CLI命令如下： router-id 10.2.3.4	Path:/routerTest Data: <routerTest> <router-id>10.2.3.4</router-id> </routerTest> 操作类型：create 生成的CLI命令如下： routerTest router-id 10.2.3.4

场景二:

Yang模型：使用render-name true
list l1 {
key k1;
leaf k1 {
hw:render-name "true"
type string;
}
leaf v1 {
type string;
}
}
	Yang模型：不使用render-name或者render-name false
	list l1 {
	key k1;
	leaf k1 {
	type string;
	}
	leaf v1 {
	type string;
	}
	}

 版权所有 © 华为技术有限公司

Path:/l1 Data: <l1> <k1>key1</k1> <v1>leaf1</v1> </l1> 操作类型：create 生成的CLI命令如下： l1 k1 key1 v1 leaf1	Path:/l1 Data: <l1> <k1>key1</k1> <v1>leaf1</v1> </l1> 操作类型：create 生成的CLI命令如下： l1 key1 v1 leaf1

13.2.2.4 optional-space list/leaf

名称	描述
标签名	optional-space
作用节点	list/leaf
父节点	无要求
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	否
功能描述	list：允许list名称和list的key合并，可以不用空格隔开 leaf：允许leaf(非key)名称和其value合并，可以不用空格隔开
使用场景	场景一：针对list，逆向解析时，可以区分没有空格的场景。 场景二：针对接口等leaf节点，逆向解析时，可以区分没有空格的场景。

13.2.2.4.1 使用场景举例

场景一

 版权所有 © 华为技术有限公司

YANG模型：list下使用optional-space	YANG模型：list不使用optional-space

list Ethernet {
hw:optional-space;
key name;
leaf name {
type string {
pattern "[0-9]+.*";
}
}
}
	list Ethernet {
	key name;
	leaf name {
	type string {
	pattern "[0-9]+.*";
	}
	}
	}

CLI to YANG结果 CLI 命令： Ethernet0/1/5 预期结果： <data> < Ethernet> <name>0/1/5</name> </ Ethernet> </data>	CLI to YANG结果 CLI 命令： Ethernet0/1/5 预期结果： 转换报错

场景二：

YANG模型：非key leaf下使用optional-space	YANG模型：非key leaf下不使用optional-space

list l1 {
key v1;
leaf v1 {
type string;
}
leaf v2 {
hw:optional-space;
type string;
}
}
	list l1 {
	key v1;
	leaf v1 {
	type string;
	}
	leaf v2 {
	type string;
	}
	}

 版权所有 © 华为技术有限公司

CLI to YANG结果 输入CLI 命令： l1 vv v2ggg 预期结果： <data> < l1> <v1>vv</v1> <v2>ggg</v2> </l1> </data>	CLI to YANG结果 输入CLI 命令： l1 vv v2ggg 预期结果： 转换报错

13.2.2.5 render-undo-value true|false leaf

名称	描述
标签名	render-undo-value true|false
作用节点	leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	true：对于删除操作，生成的删除命令要包含leaf的值 false：对于删除操作，生成的删除命令不包含leaf的值
使用场景	场景一：在华为样式设备中，no l1 <string>，删除一个leaf节点，需要加上leaf本身的值，否则设备无法删除。 场景二：对于带值删除的设备（如Tellabs设备），如果期望删除生成的删除命令不含leaf值，则可以通过添加该annotation实现。
备注	三方包中如果defaultDeleteOperationWithValue项配置为了true，则生成的删除命令默认会包含leaf值。否则，删除命令默认不会包含leaf值。

13.2.2.5.1 使用场景举例

命令行手册语法：no l1 <string>	命令行手册语法：no l1

版权所有 © 华为技术有限公司

YANG模型：使用render-undo-value true
leaf l1 {
hw:render-undo-value "true";
type string;
}
	YANG模型：使用render-undo-value false
	leaf l1 {
	hw:render-undo-value "false";
	type string;
	}

删除操作 Path:/l1 预期结果： no l1 v1 解析：leaf l1中加了render-undo-value true。因此在删除的时候，需要输入l1的值。	删除操作 Path:/l1 预期结果： no l1 解析：leaf l1中加了render-undo-value false。因此在删除的时候，不需要输入l1的值。

13.2.2.6 render-undo leaf

名称	描述
标签名	render-undo
作用节点	leaf
父节点	无要求
支持方向	双向
支持数据类型	boolean
是否可继承	否
功能描述	默认情况下<leaf>true</leaf>会转换成leaf true命令，<leaf>false</leaf>会转换成leaf false命令，加上render-undo之后，便会转换成leaf和no leaf命令。
使用场景	需要明确指出leaf节点配与不配的场景
备注	不能与insert-before/after或render-single-line叠加使用。

13.2.2.6.1 使用场景举例

Yang模型：使用render-undo
leaf keepalive {
hw:render-undo;
type boolean;
}
	Yang模型：不使用render-undo
	leaf keepalive {
	type boolean;
	}

版权所有 © 华为技术有限公司

Path:/keepalive Data: <keepalive>false</keepalive> 操作类型：create 生成的CLI命令如下： no keepalive Path:/keepalive Data: <keepalive>true</keepalive> 操作类型：create 生成的CLI命令如下： keepalive	Path:/keepalive Data: <keepalive>false</keepalive> 操作类型：create 生成的CLI命令如下： keepalive false Path:/keepalive Data: <keepalive>true</keepalive> 操作类型：create 生成的CLI命令如下： keepalive true

13.2.2.7 render-undo-before container/list/leaf/leaf-list

名称	描述
标签名	render-undo-before
作用节点	container/list/ leaf /leaf-list
父节点	无要求
支持方向	YANG -> CLI
支持数据类型	无限制
是否可继承	否
功能描述	render-undo-before告诉CLI引擎，删除时，在元素前面嵌入no，而不是在行首。
使用场景	场景一：一般在类Juniper样式(长命令)的设备中比较常见，router interface <name> [no] port，在华为样式(短命令)的设备使用的很少。
备注	只适用于删除的情况

13.2.2.7.1 使用场景举例

命令行手册语法：router interface <name> [no] port	命令行手册语法： [no] router interface <name> port

版权所有 © 华为技术有限公司

YANG模型：使用render-undo-before
container router {
list interface {
hw:render-undo-before;
ordered-by user;
key "name";
leaf name {
type types:quoted-string;
}
leaf address {
type string;
}
leaf port {
hw:render-undo-before;
type string;
}
leaf shutdown {
type empty;
}
}
	YANG模型：不使用render-undo-before
	container router {
	list interface {
	ordered-by user;
	key "name";
	leaf name {
	type types:quoted-string;
	}
	leaf address {
	type string;
	}
	leaf port {
	type string;
	}
	leaf shutdown {
	type empty;
	}
	}

CLI to YANG结果 Path:/router/interface/name=if6/port 对port执行删除操作 预期结果： router interface "if6" no port	CLI to YANG结果 Path:/router/interface/name=if6/port 对port执行删除操作 预期结果： no router interface "if6" port

13.2.2.8 render-nested-delete true|false container/list

名称	描述
标签名	render-nested-delete true|false
作用节点	container/list
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制

版权所有 © 华为技术有限公司

名称	描述
是否可继承	向下传递，子层容器也起作用
功能描述	true:先递归删除容器节点中的子节点，再删除容器节点本身。 false:抑制递归删，直接删除容器。通常父层节点使用render-nested-delete true扩展字导致该容器及子节点会被递归删除，在不期望删除节点前抑制递归删除。
使用场景	场景一：先递归删除容器下的子节点，再删除容器自身 场景二：抑制递归删，不删除子节点，直接删除自身
备注	**该annotation属于高危操作，请慎重使用！**

13.2.2.8.1 使用场景举例

场景一

YANG模型：使用render-nested-delete	YANG模型：不使用render-nested-delete

container c1 {
hw:render-nested-delete true;
leaf l3 {
type string;
}
}
	container c1 {
	leaf l3 {
	type string;
	}
	}

YANG to CLI结果 Path:/c1 Data: <data> <c1> <l3>test</13> </c1> </data> 操作类型：delete 预期结果： no c1 l3 test no c1 分析：因为container c1下加了render-nested-delete true，所以在删除c1前需要先递归删除它的子节点，即这里的l3.	YANG to CLI结果 Path:/c1 Data: <data> <c1> <l3>test</13> </c1> </data> 操作类型：delete 预期结果： no c1 分析：直接删除c1自身即可。

场景二：

 版权所有 © 华为技术有限公司

YANG模型：使用render-nested-delete true和false
container c1 {
hw:render-nested-delete true;
container c2{
hw:render-nested-delete false;
leaf v1{type string}
}
}
	YANG模型：不使用render-nested-delete false
	container c1 {
	hw:render-nested-delete true;
	container c2{
	leaf v1{type string}
	}
	}

YANG to CLI结果 Path:/c1 Data: <data> <c1> <c2> <v1>value1</v1> </c2> </c1> </data> 操作类型：delete 预期结果： no c1 c2 no c1 分析：直接删除c2，不递归删除c2的子leaf，因为render-nested-delete false生效	YANG to CLI结果 Path:/c1 Data: <data> <c1> <c2> <v1>value1</v1> </c2> </c1> </data> 操作类型：delete 预期结果： no c1 c2 v1 no c1 c2 no c1 分析：需要递归删除c2的子leaf。

13.2.2.9 quit-view "xxx" container/list

名称	描述
标签名	quit-view "xxx"
作用节点	container/list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否

版权所有 © 华为技术有限公司

名称	描述
功能描述	修改默认退出视图命令的行为，提示CLI转换引擎在当前的子视图中用注解中的退出命令替换默认退出命令。

13.2.2.9.1 使用场景举例

场景一

YANG模型：使用quit-view "xxx"	YANG模型：不使用quit-view "xxx"

list l1 {
hw:quit-view 'quit l1';
key k1;
leaf k1 {
type string;
}
}
	list l1 {
	key k1;
	leaf k1 {
	type string;
	}
	}

CLI to YANG结果	CLI to YANG结果
输入CLI 命令： l1 abc **quit** 预期结果： <data> <l1> <k1>abc</k1> </l1> </data>	输入CLI 命令： l1 abc 预期结果： <data> <l1> <k1>abc</k1> </l1> </data>

版权所有 © 华为技术有限公司

YANG to CLI结果： YANG Path:/l1[@k1=abc] YANG Data: <data> <l1> <k1>abc</k1> </l1> </data> 预期结果： l1 abc **quit** l1	YANG to CLI结果： YANG Path: /l1[@k1=abc] YANG Data: <data> <l1> <k1>abc</k1> </l1> </data> 预期结果： l1 abc

13.2.2.10 allow-multi-token xx leaf

名称	描述
标签名	allow-multi-token xx
作用节点	leaf
父节点	无要求
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	否
功能描述	支持leaf携带多个字符串参数，用空格隔开，数目不超过指定的参数(xx)。 若参数为-1，则表明参数数目无限制。
使用场景	场景一：设备的某些配置命令的命令元素是多个字符串，用空格隔开。 场景二：设备的某些创建视图配置命令的命令元素是多个字符串，用空格隔开。
备注	非key leaf和key leaf分别对应场景一、场景二。

13.2.2.10.1 使用场景举例

场景一

 版权所有 © 华为技术有限公司

YANG模型：使用allow-multi-token	YANG模型：不使用allow-multi-token

list l1 {
key k1;
leaf k1 {
type string;
}
leaf v1 {
hw:allow-multi-token 2;
type string;
}
leaf v2 {
type string;
}
}
	list l1 {
	key k1;
	leaf k1 {
	type string;
	}
	leaf v1 {
	type string;
	}
	leaf v2 {
	type string;
	}
	}

CLI to YANG结果	CLI to YANG结果
输入CLI 命令： l1 test v1 abcd efgh cdef v2 ghij 预期结果： <data> <l1> <k1>test</k1> <v1>abcd efgh cdef</v1> <v2>ghij</v2> </l1> </data>	输入CLI 命令： l1 test v1 abcd v2 ghij 预期结果： <data> <l1> <k1>test</k1> <v1>abcd</v1> <v2>ghij</v2> </l1> </data>

场景二

 版权所有 © 华为技术有限公司

YANG模型：使用allow-multi-token	YANG模型：不使用allow-multi-token

list l1 {
key k1;
leaf k1 {
hw:allow-multi-token 10;
type string;
}
leaf v1 {
type string;
}
leaf v2 {
type string;
}
}
	list l1 {
	key k1;
	leaf k1 {
	type string;
	}
	leaf v1 {
	type string;
	}
	leaf v2 {
	type string;
	}
	}

CLI to YANG结果	CLI to YANG结果
输入CLI 命令： l1 test1 test2 test3 v1 abcd v2 ghij 预期结果： <data> <l1> <k1>test1 test2 test3</k1> <v1>abcd</v1> <v2>ghij</v2> </l1> </data>	输入CLI 命令： l1 test1 v1 abcd v2 ghij 预期结果： <data> <l1> <k1>test1</k1> <v1>abcd</v1> <v2>ghij</v2> </l1> </data>

13.2.2.11 update-by-delete leaf/leaf-list/list

名称	描述
标签名	update-by-delete
作用节点	leaf/leaf-list/list
父节点	无要求
支持方向	YANG->CLI

 版权所有 © 华为技术有限公司

名称	描述
支持数据类型	无限制
是否可继承	否
功能描述	设备上有些配置需要先删除后才允许更新。该annotation主要就是为了解决此类问题，它告诉CLI转换引擎，在生成更新命令之前，需要先生成一条删除命令。
使用场景	某些配置不支持直接更新，必须先删除再创建
备注	涉及业务安全性，设备侧的配置可能会被删除，开发人员谨慎使用

13.2.2.11.1 使用场景举例

场景一

YANG模型：使用update-by-delete	YANG模型：不使用update-by-delete

list l1 {
key k1;
leaf k1 {
type string;
}
leaf v1 {
hw: update-by-delete;
type string;
}
}
	list l1 {
	key k1;
	leaf k1 {
	type string;
	}
	leaf v1 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果 YANG Path: /l1[@k1=test1] YANG Data: <data> <l1> <k1> test1</k1> <v1> abcd</v1> </l1> </data> 操作类型：merge 预期结果： l1 test1 no v1 v1 abcd	YANG to CLI结果 YANG Path: /l1[@k1=test1] YANG Data: <data> <l1> <k1> test1</k1> <v1> abcd</v1> </l1> </data> 操作类型：merge 预期结果： l1 test1 v1 abcd

13.2.2.12 render-single-line container

名称	描述
标签名	render-single-line
作用节点	container
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	render-single-line将container中的所有leaf节点转换成一行命令，而不是每个leaf一行。
使用场景	包含多个参数的命令
备注	不能和render-undo叠加使用。 使用该扩展字的容器，其不应该包含会进入视图的节点。

13.2.2.12.1 使用场景举例

场景一

 版权所有 © 华为技术有限公司

Yang模型：使用render-single-line
container bfd {
hw:render-single-line;
leaf interval {
hw:render-name false;
type uint16;
}
leaf multiplier {
hw:render-name false;
type uint16;
}
}
	Yang模型：不使用render-single-line
	container bfd {
	leaf interval {
	type uint16;
	}
	leaf multiplier {
	type uint16;
	}
	}

Path:/bfd Data: <bfd> <interval>15</interval> <multiplier>3</multiplier> </bfd> 操作类型：create 生成的CLI命令如下： bfd 15 3	Path:/bfd Data: <bfd> <interval>15</interval> <multiplier>3</multiplier> </bfd> 操作类型：create 生成的CLI命令如下： bfd interval 15 bfd multiplier 3

13.2.2.13 render-multi-line leaf-list/list

名称	描述
标签名	render-multi-line
作用节点	leaf-list/list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	leaf-list: 默认leaf-list有多个值的情况下，会生成一条多个参数的命令，该扩展字会使每个值都生成一条命令。 list: 抑制list进入视图，和render-view false区分开来，key leaf后只能跟单个非key leaf,若有多个非key leaf，则分行。

版权所有 © 华为技术有限公司

名称	描述
使用场景	参考功能描述
备注	NA

13.2.2.13.1 使用场景举例

场景一：leaf-list使用render-multi-line生成多条命令

Yang模型：使用render-multi-line	Yang模型：不使用render-multi-line

container router {
list isis {
key tag;
leaf tag{
type string;
}
leaf-list summary-address {
hw:render-multi-line;
type inet:ipv4-prefix;
}
}
}
	container router {
	list isis {
	key tag;
	leaf tag{
	type string;
	}
	leaf-list summary-address {
	type inet:ipv4-prefix;
	}
	}
	}

版权所有 © 华为技术有限公司

Path:/router Data: <router> <isis> <tag>100</tag> <summary-address>10.2.3.4/32</summary-address> <summary-address>10.2.3.5/32</summary-address> <summary-address>10.2.3.6/32</summary-address> </isis> </router> 操作类型：create 生成的CLI命令如下： router isis 100 summary-address 10.2.3.4/32 summary-address 10.2.3.5/32 summary-address 10.2.3.6/32	Path:/router Data: <router> <isis> <tag>100</tag> <summary-address>10.2.3.4/32</summary-address> <summary-address>10.2.3.5/32</summary-address> <summary-address>10.2.3.6/32</summary-address> </isis> </router> 操作类型：create 生成的CLI命令如下： router isis 100 summary-address 10.2.3.4/32 10.2.3.5/32 10.2.3.6/32

场景二：list下使用render-multi-line

YANG模型：使用render-multi-line
list l1{
hw:render-multi-line;
key k1;
leaf k1{
type string;
}
leaf v1{
type string;
}
leaf v2{
type string;
}
}
	YANG模型：不使用render-multi-line
	list l1{
	key k1;
	leaf k1{
	type string;
	}
	leaf v1{
	type string;
	}
	leaf v2{
	type string;
	}
	}

 版权所有 © 华为技术有限公司

YANG to CLI结果 <data> <l1> <k1>key1</k1> <v1>val1</v1> <v2>val2</v2> </l1> </data> 预期结果： l1 key1 v1 val1 l2 key1 v2 val2	YANG to CLI结果 <data> <l1> <k1>key1</k1> <v1>val1</v1> <v2>val2</v2> </l1> </data> 预期结果： l1 key v1 val1 v2 val2

13.2.2.14 render-sequential-order container

名称	描述
标签名	render-sequential-order
作用节点	container
父节点	无要求
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	否
功能描述	主要是告诉CLI解析引擎只接受container顺序与数据模型中出现的顺序相同的leaf，一般跟render-name false一起使用，只有value，没有name的情况。
使用场景	场景一：container下有多个leaf，设备命令行不显示leaf name，按照leaf在yang模型中的顺序解析命令行中的参数
备注	需要配合render-single-line使用，且两个扩展字的顺序固定，render-single-line在render-sequential-order上方

13.2.2.14.1 使用场景举例

场景一

 版权所有 © 华为技术有限公司

YANG模型：使用render-sequential-order
container c1 {
hw:render-single-line;
hw:render-sequential-order;
leaf l1 {
hw:render-name "false";
type string;
}
leaf l2 {
hw:render-name "false";
type boolean;
}
}
	YANG模型：不使用render-sequential-order
	container c1 {
	hw:render-single-line;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

YANG to CLI结果 输入CLI 命令： c1 val1 false 预期结果： <data> <c1> <l1>val1</l1> <l2>false</l2> </c1> </data> 解析：render-order主要和render-name false叠加使用。render-name false可以在输入CLI命令时省去节点名。而render-sequential-order保证了输入节点的顺序必须完全按照Yang模型定义顺序，才能正确解析。 注：如果l1和l2类型相同且输入无误时，顺序打乱会导致l1和l2赋值错误，但是程序不会报错。	YANG to CLI结果 输入CLI 命令： c1 l2 val2 l1 val1 预期结果： <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 解析：不使用render-sequential-order和render-name false的时候，需要在设置节点的时候显式的输入l1和l2的名字，但是顺序可以打乱。

13.2.2.15 render-optional "xxx" container

名称	描述
标签名	render-optional

版权所有 © 华为技术有限公司

名称	描述
作用节点	container
父节点	无要求
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	否
功能描述	配合render-sequential-order一起使用。render-sequential-order表明该容器里元素必须按照正确的节点顺序出现；若有子元素使用render-optional扩展字，则该元素可以跳过
使用场景	场景一：命令行多参数场景，配置render-sequential-order命令行需要按照一定的顺序出现，但存在部分非必选参数。如c1 [l1 <string>] l2 <string> l3 <string> l4 <string>，命令行节点l1为非必选参数，需要使用render-optional进行说明。
备注	需要与render-single-line和render-sequential-order叠加使用。

13.2.2.15.1 使用场景举例

命令行手册语法：c1 [l1 <string>] l2 <string> l3 <string> l4 <string>	命令行手册语法：c1 l1 <string> l2 <string> l3 <string> l4 <string>

YANG模型：使用render-optional
container c1 {
hw:render-single-line;
hw:render-optional 'l1';
hw:render-sequential-order;
leaf l1 {
type string;
}
leaf l2 {
type string;
}
leaf l3 {
type string;
}
leaf l4 {
type string;
}
}
	YANG模型：不使用render-optional
	container c1 {
	hw:render-single-line;
	hw:render-sequential-order;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	leaf l3 {
	type string;
	}
	leaf l4 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

CLI to YANG结果 c1 l2 val2 l3 val3 l4 val4 预期结果： <data> <c1> <l2>val2</l2> <l3>val3</l3> <l4>val4</l4> </c1> </data> 命令解析：render-sequential-order告诉CLI解析引擎只接受container顺序与数据模型中出现的顺序相同的leaf；render-optional使leaf l1变成可选参数，可以去除。	CLI to YANG结果 c1 l1 val1 l2 val2 l3 val3 l4 val4 预期结果： <data> <c1> <l1>val1</l1> <l2>val2</l2> <l3>val3</l3> <l4>val4</l4> </c1> </data> 命令解析：l1 val1必须存在，否则报错

13.2.2.16 render-sequential-order-until "name" container

名称	描述
标签名	render-sequential-order-until "name"
作用节点	container
父节点	无限制
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	否
功能描述	通过指定的子元素"name"来分割容器子leaf元素，前一部分必须按顺序解析，后一部分可以无序。
使用场景	容器的前一部分子元素需要有序，后一部分可以无序。
备注	和render-sequential-order有区别。一般配合render-optional、render-single-line和render-name false使用。

版权所有 © 华为技术有限公司

13.2.2.16.1 使用场景举例

YANG模型：使用render-sequential-order-until	YANG模型：不使用render-sequential-order-until

container c1 {
hw:render-sequential-order-until "l2";
hw:render-single-line;
leaf l1 {
hw:render-name "false"
type boolean;
}
leaf l2 {
hw:render-name "false"
type string;
}
leaf l3 {
type uint8;
}
leaf l4 {
type string;
}
}
	container c1 {
	leaf l1 {
	type boolean;
	}
	leaf l2 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果 输入CLI 命令： c1 true value2 l4 value4 l3 22 预期结果： <data> <c1> <l1>true</l1> <l2>value2</l2> <l3>22</l3> <l4>value4</l4> </c1> </data> 解析：render-sequential-order-until "l2"指明l1和l2必须按顺序解析, l3和l4可以无序。 注：如果l1和l2类型相同且输入无误时，顺序打乱会导致l1和l2赋值错误，但是程序不会报错。	YANG to CLI结果 输入CLI 命令： c1 l2 value2 l1 false 预期结果： <data> <c1> <l1>false</l1> <l2>value2</l2> </c1> </data> 解析：不使用render-sequential-order-until和render-name false，需要在设置节点的时候显式的输入l1和l2的名字，顺序可以打乱。

13.2.2.17 render-suffix-on-create "xxx" list

名称	描述
标签名	add-suffix-on-create "xxx"
作用节点	list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	在创建list视图时，添加该annotation，会在命令中list那一行末尾加上指定的参数"xxx"，表明这是第一次创建。
使用场景	在创建视图时，如果list有该annotation，则会指明这是创建视图。
备注	只有创建时才会生效，更新不生效

版权所有 © 华为技术有限公司

13.2.2.17.1 使用场景举例

YANG模型：使用add-suffix-on-create	YANG模型：不使用add-suffix-on-create

list l1{
hw:add-suffix-on-create "create";
key k1;
leaf k1{
type string;
}
leaf v1{
type string;
}
leaf v2{
type string;
}
}
	list l1{
	key k1;
	leaf k1{
	type string;
	}
	leaf v1{
	type string;
	}
	leaf v2{
	type string;
	}
	}

YANG to CLI结果 <data> <l1> <k1>key1</k1> <v1>val1</v1> <v2>val2</v2> </l1> </data> 预期结果： l1 key1 create v1 val1 v2 val2	YANG to CLI结果 <data> <l1> <k1>key1</k1> <v1>val1</v1> <v2>val2</v2> </l1> </data> 预期结果： l1 key v1 val1 v2 val2

13.2.2.18 replace-value "xxx,yyy" leaf

名称	描述
标签名	replace-value
作用节点	leaf
父节点	无限制
支持方向	YANG->CLI
支持数据类型	无限制

版权所有 © 华为技术有限公司

名称	描述
是否可继承	否
功能描述	在设置leaf值时，如果值是"xxx"，则用"yyy"代替
使用场景	当我们在配置参数时，不方便直接指定leaf值时，可以用该扩展字去替换leaf的值
备注	NA

13.2.2.18.1 使用场景举例

YANG模型：使用replace-value	YANG模型：不使用replace-value

container c1{
leaf f1{
type string;
hw:replace-value 'one,two';
}
}
	container c1{
	leaf f1{
	type string;
	}
	}

CLI命令行： c1 one\n YANG to CLI结果 <data> <c1> <f1>two</f1> </c1> </data>	CLI命令行： c1 one\n YANG to CLI结果 <data> <c1> <f1>one</f1> </c1> </data>

13.2.2.19 delete-siblings-on-delete container/list/leaf/leaf-list

名称	描述
标签名	delete-siblings-on-delete
作用节点	container/list/leaf/leaf-list
父节点	container/list
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否

版权所有 © 华为技术有限公司

名称	描述
功能描述	当删除某一使用了该扩展字的节点时，则会告诉上游系统去删除控制器中该节点的所有兄弟节点的数据。
使用场景	NA

13.2.2.19.1 使用场景举例

NA

13.2.2.20 delete-parents-on-delete leaf

名称	描述
标签名	delete-parents-on-delete
作用节点	leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	删除某一带有该扩展字节点时，则会告诉上游系统，删除该节点的父节点数据
备注	只能是leaf节点，且有父节点

13.2.2.20.1 使用场景举例

NA

13.2.2.21 render-undo-name true|false container/list/leaf

名称	描述
标签名	render-undo-name true|false
作用节点	container/leaf/list
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否

版权所有 © 华为技术有限公司

名称	描述
功能描述	在删除操作时，不生成节点的节点名
使用场景	场景一：特定设备删除命令不能加leaf节点的名称，否则报错。Yang模型：父节点container，子节点leaf(多个)，container下使用render-single-line，leaf下使用render-undo-name和render-undo-value；进行删除container操作。 场景二：在递归删除一个container/list时，默认转换规则会先递归生成该container下的子节点的删除命令，然后再生成一条该container本身的删除命令。此时，如果不希望生成该container本身的删除命令，可以通过添加该annotation实现。
备注	该扩展字使用的场景比较特殊，需要在特定的场景一、二，且和其他扩展项配合使用

13.2.2.21.1 使用场景举例

场景一：删除container

YANG模型：使用render-undo-name	YANG模型：不使用render-undo-name

container c1{
hw:render-single-line;
leaf v1{
hw:render-undo-name false;
hw:render-undo-value false;
type string;
}
leaf v2{
hw:render-undo-name false;
hw:render-undo-value false;
type string;
}
}
	container c1{
	hw:render-single-line;
	leaf v1{
	type string;
	}
	leaf v2{
	type string;
	}
	}

版权所有 © 华为技术有限公司

使用cli-no-name-on-delete YANG to CLI结果 Path:/l1 Data: <c1> <v1>value1</v1> <v2>value2</v2> </c1> 操作类型：delete 预期结果： no c1 说明：在删除c1的时候，因为render-single-line会带上其子层节点，子层leaf使用了render-undo-name/value直接隐藏了本身	不使用cli-no-name-on-delete YANG to CLI结果 Path:/l1 Data: <c1> <v1>value1</v1> <v2>value2</v2> </c1> 操作类型：delete 预期结果： no c1 v1 value1 v2 value2 说明：在删除c1时，后面会跟上leaf节点名称和value。但特定设备可能不支持这样的命令，会报错

场景二：与render-nested-delete同时使用，不生成删除该container本身的命令

YANG模型：使用render-undo-name
container c2 {
hw:render-nested-delete;
hw:render-undo-name;
leaf l2 {
type string;
}
leaf l3 {
type string;
}
}
	YANG模型：不使用render-undo-name
	container c2 {
	hw:render-nested-delete;
	hw:render-undo-name;
	leaf l2 {
	type string;
	}
	leaf l3 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果 Path:/c2 Data: <c2> <l2>abc</l2> <l3>def</l3> </c2> 预期结果： no c2 l2 no c2 l3 说明：container c2上加上了render-undo-name，不再生成删除c2自身的命令no c2，只生成了删除leaf l2和leaf l3的命令。	YANG to CLI结果 Path:/c2 Data: <c2> <l2>abc</l2> <l3>def</l3> </c2> 预期结果： no c2 l2 no c2 l3 no c2 说明：container c2没有加render-undo-name，不仅生成了删除leaf l2和leaf l3的命令，而且生成了删除container c2的命令，即no c2

13.2.2.22 delete-by-set-default-value leaf

名称	描述
标签名	delete-by-set-default-value
作用节点	leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	在对设置有默认值的leaf节点进行删除时，生成update为默认值命令行
使用场景	删除操作
备注	NA

版权所有 © 华为技术有限公司

13.2.2.22.1 使用场景举例

YANG模型：使用delete-by-set-default-value
container c1 {
leaf f1 {
hw:delete-by-set-default-value;
type string;
default "disable";
}
}
	YANG模型：不使用delete-by-set-default-value
	container c1 {
	leaf f1 {
	type string;
	default "disable";
	}
	}

CLI命令行： c1 f1 disable	CLI命令行： no c1 f1

13.2.2.23 delete-no-support container/leaf

名称	描述
标签名	delete-no-support
作用节点	container/leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	是
功能描述	对带有该注解的节点进行删除时，生成空命令行
使用场景	删除操作
备注	NA

版权所有 © 华为技术有限公司

13.2.2.23.1 使用场景举例

YANG模型：使用delete-no-support
container c1 {
leaf f1 {
hw: delete-no-support;
type string;
}
}
	YANG模型：不使用delete-no-support
	container c1 {
	leaf f1 {
	type string;
	}
	}

CLI命令行：	CLI命令行： no c1 f1

13.2.2.24 leaf-dependency "xxx,xxx" container/list

名称	描述
标签名	leaf-dependency
作用节点	container/list
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	当container或list下节点需要依赖其他节点一起更新的时候，使用这个注解
使用场景	多参数命令行，当其中的某个参数更新时命令行需要带上其他的参数
备注	参数为all，代表container或list下所有节点都要依赖其他节点一起更新

版权所有 © 华为技术有限公司

13.2.2.24.1 使用场景举例

YANG模型：使用leaf-dependency
container c1 {
hw: leaf-dependency "f1,f2";
leaf f1 {
type string;
}
leaf f2 {
type string;
}
Leaf f3 {
type string;
}
}
	YANG模型：不使用leaf-dependency
	container c1 {
	leaf f1 {
	type string;
	}
	leaf f2 {
	type string;
	}
	Leaf f3 {
	type string;
	}
	}

CLI命令行： 场景1：更新f1或f2 c1 f1 value1 f2 value2 f3 value3 场景2：更新f3 c1 f3 value3	CLI命令行： 场景1：更新f1或f2 c1 f1 value1 或 c1 f2 value2 场景2：更新f3 c1 f3 value3

13.2.2.25 render-flat-container container

名称	描述
标签名	render-flat-container
作用节点	container
父节点	配置render-single-line的container
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	render-flat-container将配置render-single-line的container的所有子container、所有leaf节点跟父节点转换成一行命令。
使用场景	包含多个参数的命令

版权所有 © 华为技术有限公司

名称	描述
备注	不能和render-undo叠加使用。 使用该扩展字的容器，其不应该包含会进入视图的节点。

13.2.2.25.1 使用场景举例

YANG模型：使用render-flat-container	YANG模型：不使用render-flat-container

container c1 {
hw:render-single-line;
container c2 {
hw: render-flat-container;
leaf l1 {
type string;
}
leaf l2 {
type string;
}
}
leaf l3 {
type string;
}
}
	container c1 {
	hw:render-single-line;
	container c2 {
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}
	leaf l3 {
	type string;
	}
	}

Data: <c1> <c2> <l1>value1</l1> <l2>value2</l2> </c2> <l3>value3</l3> </c1> 操作类型：create 生成的CLI命令如下： c1 c2 l1 value1 l2 value2 l3 value3	Data: <c1> <c2> <l1>value1</l1> <l2>value2</l2> </c2> <l3>value3</l3> </c1> 操作类型：create 生成的CLI命令如下： c1 c2 l1 value1 c2 l2 value2 l3 value3

 版权所有 © 华为技术有限公司

13.2.2.26 render-container-together container

名称	描述
标签名	render-container-together
作用节点	presence container
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	presence container和leaf节点不分行展示
使用场景	presence container默认会单独生成一条命令行，该扩展项用在presence container不需要单独生成一条命令行的场景
备注	NA

13.2.2.26.1 使用场景举例

YANG模型：使用render-container-together	YANG模型：不使用render-container-together

container c1{
presence true;
cli:render-container-together;
leaf l1 {
type string;
}
leaf l2 {
type string;
}
}
	container c1{
	presence true;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

Data: <c1> <l1>value1</l1> <l2>value2</l2> </c1> 操作类型：create 生成的CLI命令如下： c1 l1 value1 c1 l2 value2	Data: <c1> <l1>value1</l1> <l2>value2</l2> </c1> 操作类型：create 生成的CLI命令如下： c1 c1 l1 value1 c1 l2 value2

13.2.2.27 render-identifier container/list/leaf/leaf-list

名称	描述
标签名	render-identifier
作用节点	container/list/leaf/leaf-list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	生成/解析命令行使用指定的标识符替换定义的yang标识符名称
使用场景	某些标识名不能用于yang模型命名
备注	NA

13.2.2.27.1 使用场景举例

YANG模型：使用render-identifier
container cmd8021p {
cli:render-identifier "8021p";
leaf l1{
type string;
}
}
	YANG模型：不使用render-identifier
	container cmd8021p {
	leaf l1 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

Data: <data> < cmd8021p > <l1>value1</l1> </ cmd8021p > </data> 操作类型：create 生成的CLI命令如下： 8021p l1 value1	Data: <data> < cmd8021p > <l1>value1</l1> </ cmd8021p > </data> 操作类型：create 生成的CLI命令如下： cmd8021p l1 value1

13.2.2.28 render-undo-name-and-value true|false leaf

名称	描述
标签名	render-undo-name-and-value
作用节点	leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	true：对于删除操作，生成的删除命令要包含leaf的名字和值 false：对于删除操作，生成的删除命令不包含leaf的名字和值
使用场景	用于undo命令行不需要对应配置项的名字和值场景
备注	用于Delete操作

版权所有 © 华为技术有限公司

13.2.2.28.1 使用场景举例

YANG模型：使用render-undo-name-and-value
container c1 {
cli:render-single-line;
leaf l1 {
cli:render-undo-name-and-value false;
type string;
}
leaf l2 {
cli:render-undo-name-and-value false;
type string;
}
}
	YANG模型：不使用render-undo-name-and-value
	container c1 {
	cli:render-single-line;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

Data: <data> <c1> <l1>value1</l1> <l2>value2</l2> </ c1 > </data> 操作类型：delete 生成的CLI命令如下： no c1	Data: <data> <c1> <l1>value1</l1> <l2>value2</l2> </ c1 > </data> 操作类型：delete 生成的CLI命令如下： no c1 l1 l2

13.2.2.29 render-order-dependency container/leaf

名称	描述
标签名	render-order-dependency
作用节点	container/leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否

版权所有 © 华为技术有限公司

名称	描述
功能描述	指定下发到设备的命令行顺序，适用于所有操作类型。有5个子annotations可以指定适用的操作类型：operation-create,operation-update,operation-delete,operation-all,operation-create-delete，默认是operation-all。
使用场景	对下发设备命令行有顺序要求的场景。
备注	NA

13.2.2.29.1 使用场景举例

YANG模型：使用render-order-dependency	YANG模型：不使用render-order-dependency

container cd0 {
container c1 {
leaf l1 { type string; }
leaf l2 { type string; }
}
container c2 {
cli:render-order-dependency "/cd0/c3";
leaf l3 { type string; }
leaf l4 { type string; }
}
container c3 {
cli:render-order-dependency "/cd0/c1" {
cli:operation-delete;
}
leaf l5 { type string; }
leaf l6 { type string; }
}
}
	container cd0 {
	container c1 {
	leaf l1 { type string; }
	leaf l2 { type string; }
	}
	container c2 {
	leaf l3 { type string; }
	leaf l4 { type string; }
	}
	container c3 {
	leaf l5 { type string; }
	leaf l6 { type string; }
	}
	}

版权所有 © 华为技术有限公司

Data: <data> <cd0> <c1> <l1>value1</l2> </c1> <c2> <l4>value4</l2> </c2> <c3> <l5>value5</l5> </c3> </ cd0 > </data> 操作类型：delete 生成的CLI命令如下： no cd0 c1 l1 no cd0 c3 l5 no cd0 c2 l4	Data: <data> <cd0> <c1> <l1>value1</l2> </c1> <c2> <l4>value4</l2> </c2> <c3> <l5>value5</l5> </c3> </ cd0 > </data> 操作类型：delete 生成的CLI命令如下： no cd0 c1 l1 no cd0 c2 l4 no cd0 c3 l5

13.2.2.30 delete-on-empty container

名称	描述
标签名	delete-on-empty
作用节点	container
父节点	presence container
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	leaf节点全部删除的时候，联动删除数据库的presence container, 保持控制器和设备一致
使用场景	用于leaf节点全部删除，container自动联动删除的场景
备注	NA

版权所有 © 华为技术有限公司

13.2.2.30.1 使用场景举例

YANG模型：使用delete-on-empty
container c2 {
presence true;
cli:delete-on-empty;
leaf l1 {
type string;
}
leaf l2 {
type string;
}
}
	YANG模型：不使用delete-on-empty
	container c2 {
	presence true;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

Data: <data> <c2> <l1>value1</l1> <l2>value2</l2> </ c2 > </data> 操作类型：delete 生成的CLI命令如下： no c2 l1 no c2 l2 数据库联动删除：c2	Data: <data> <c2> <l1>value1</l1> <l2>value2</l2> </ c2 > </data> 操作类型：delete 生成的CLI命令如下： no c2 l1 no c2 l2 无数据库联动

13.2.2.31 reset-all-siblings container

名称	描述
标签名	reset-all-siblings
作用节点	使用render-sequential-order的container
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	对leaf节点delete and update的时候会联动reset container下所有其他的leaf，有默认值的reset为默认值，没有的置空。

版权所有 © 华为技术有限公司

名称	描述
使用场景	多参数命令行，某个参数的更新或者删除会重置其他参数
备注	配置有mandatory相邻leaf节点不会被重置

13.2.2.31.1 使用场景举例

YANG模型：使用reset-all-siblings
container c1 {
container c2 {
cli:render-single-line;
cli:render-sequential-order;
cli:reset-all-siblings;
leaf l1 {
type string;
}
leaf l2 {
type string;
default "def-l2-value";
}
}
}
	YANG模型：不使用reset-all-siblings
	container c1 {
	container c2 {
	cli:render-single-line;
	cli:render-sequential-order;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	default "def-l2-value";
	}
	}
	}

原始配置Data: <data> <c1> <c2> <l1>value1</l1> <l2>value2</l2> </c2 > <c1> </data> 删除l1 value1 操作类型：delete 生成的CLI命令如下： no c1 c2 l1 数据库联动reset l2为默认值def-l2-value， l1值为空， l2值为def-l2-value	原始配置Data: <data> <c1> <c2> <l1>value1</l1> <l2>value2</l2> </c2 > <c1> </data> 删除l1 value1 操作类型：delete 生成的CLI命令如下： no c1 c2 l1 l1值为空， l2值为value2

 版权所有 © 华为技术有限公司

13.2.2.32 reset-siblings container

名称	描述
标签名	reset-siblings
作用节点	使用render-sequential-order-until的container
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	对leaf节点delete and update的时候会联动reset container下render-sequential-order-until指定的leaf，有默认值的reset为默认值，没有的置空。
使用场景	多参数命令行，某个参数的更新或者删除会重置其他部分参数
备注	配置有mandatory相邻leaf节点不会被重置

13.2.2.32.1 使用场景举例

YANG模型：使用reset-siblings	YANG模型：不使用reset-siblings

container c9 {
cli:render-single-line;
cli:render-sequential-order-until "l3";
cli:reset-siblings;
leaf l1 {
type string;
}
leaf l2 {
type string;
}
leaf l3 {
type string;
}
leaf l4 {
type string;
}
}
	container c9 {
	cli:render-single-line;
	cli:render-sequential-order-until "l3";
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	leaf l3 {
	type string;
	}
	leaf l4 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

原始配置Data: <data> <c9> <l1>value1</l1> <l2>value2</l2> <l3>value3</l3> <l4>value4</l4> </c9 > </data> 删除l1 value1 操作类型：delete 生成的CLI命令如下： no c9 l1 数据库联动reset l2、l3为空， l1、l2、l3为空值为空， l4值为value4	原始配置Data: <data> <c9> <l1>value1</l1> <l2>value2</l2> <l3>value3</l3> <l4>value4</l4> </c9 > </data> 删除l1 value1 操作类型：delete 生成的CLI命令如下： no c9 l1 l1值为空， l2值为value2，l3值为value3, l4值为value4

13.2.2.33 cli-case-sensitive/insensitive container/list/leaf/leaf-list

名称	描述
标签名	cli-case-sensitive/insensitive
作用节点	container/list/leaf/leaf-list
父节点	无要求
支持方向	CLI->YANG
支持数据类型	无限制
是否可继承	是
功能描述	1、定义当解析CLI命令行时，Yang模型匹配节点名称是否大小写敏感。默认大小写敏感。 2、配置有cli-case-insensitive的节点作用范围是整个子树，可用cli-case-sensitive打破作用范围。
使用场景	设备命令行的关键字对大小写不敏感
备注	NA

 版权所有 © 华为技术有限公司

13.2.2.33.1 使用场景举例

YANG模型：使用cli-case-sensitive、cli-case-insensitive
container c1 {
container c1a {
hw:cli-case-insensitive;
container cb {
container cc {
hw:cli-case-sensitive;
leaf la1 {
type string;
}
}
}
}
}
	YANG模型：不使用cli-case-sensitive、cli-case-insensitive
	container c1 {
	container c1a {
	container cb {
	container cc {
	leaf la1 {
	type string;
	}
	}
	}
	}
	}

CLI to YANG结果 输入CLI 命令： c1 C1A cB cc la1 la1VAL 预期结果： <data> <c1> <c1a> <cb> <cc> <la1>la1VAL</la1> </cc> </cb> </c1a> </c1> </data> 说明：c1a、cb大小写不敏感，c1、cc、la1大小写敏感。	CLI to YANG结果 输入CLI 命令： c1 C1A cB cc la1 la1VAL 预期结果： <data/> 说明：默认大小写敏感，匹配不上。

13.2.2.34 render-hide-in-view leaf

名称	描述
标签名	render-hide-in-view

版权所有 © 华为技术有限公司

名称	描述
作用节点	leaf
父节点	带有render-view true扩展项的Container
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	用于设置leaf节点作为进入子视图的语句
使用场景	使用container和子leaf进入子视图场景
备注	NA

13.2.2.34.1 使用场景举例

YANG模型：使用render-hide-in-view	YANG模型：不使用render-hide-in-view

container c1 {
hw:render-view true;
leaf l1 {
hw:render-hide-in-view;
type string;
}
leaf l2 {
type string;
}
}
	container c1 {
	hw:render-view true;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：create 预期结果： c1 l1 val1 l2 val2	YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：create 预期结果： c1 l1 val1 l2 val2

13.2.2.35 reset-container container/list/leaf

名称	描述
标签名	reset-container
作用节点	container/list/leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	1.当配置于container，list节点时，当删除或者更新其孩子时，其余的孩子节点需要被重置。如果有默认值，重置为默认值，否则置空。 2.当配置于leaf节点时，当删除或者更新该叶子节点时，其兄弟节点需要被重置。
使用场景	多参数命令行，某个参数的更新或者删除会重置其他参数
备注	不能与delete-parents-on-delete同时使用

 版权所有 © 华为技术有限公司

13.2.2.35.1 使用场景举例

YANG模型：使用reset-container	YANG模型：不使用reset-container

container c1 {
leaf l1 {
cli:reset-container;
type string;
}
leaf l2 {
type string;
}
}
	container c1 {
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1_new</l1> </c1> </data> 原始数据： <data> <c1> <l1>val1_old</l1> <l2>val2_old</l2> </c1> </data> 操作类型：merge 预期结果： c1 l1 val1_new AOC DB更新：reset l2 (空或者默认值) 回滚命令： c1 l1 val1_old c1 l2 val2_old	YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1_new</l1> </c1> </data> 原始数据： <data> <c1> <l1>val1_old</l1> <l2>val2_old</l2> </c1> </data> 操作类型：create 预期结果： c1 l1 val1_new 回滚命令： c1 l1 val1_old

13.2.2.36 render-batch-list-syntax list/leaf-list

名称	描述
标签名	render-batch-list-syntax

版权所有 © 华为技术有限公司

名称	描述
作用节点	list/leaf-list
父节点	无要求
支持方向	双向
支持数据类型	无限制
是否可继承	否
功能描述	将数据库存的离散值整合为范围的命令行
使用场景	vlan batch 1 to 3 5这类范围类的命令行
备注	需要在配置文件配置rangeCommand、rangeIndicator、rangeSeparator参数项

13.2.2.36.1 使用场景举例

YANG模型：使用render-batch-list-syntax	YANG模型：不使用render-batch-list-syntax

list vlan {
cli:render-batch-list-syntax true;
key id;
leaf id {
type uint16 {
range "1..4096";
}
}
}
	list vlan {
	cli:render-batch-list-syntax true;
	key id;
	leaf id {
	type uint16 {
	range "1..4096";
	}
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果: Path:/vlan Data: <data> <vlan> <l1>1</l1> <l2>2</l2> </vlan> </data> 操作类型：create 预期结果： vlan batch 1 to 2	YANG to CLI结果: Path:/vlan Data: <data> <vlan> <l1>1</l1> <l2>2</l2> </vlan> </data> 操作类型：create 预期结果： vlan 1 vlan 2

13.2.2.37 incomplete-command container/leaf

名称	描述
标签名	incomplete-command
作用节点	container/leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	用于创建/更新操作， 确保使用该扩展项的leaf不作为最后的配置节点
使用场景	多参数命令行，对输入的参数的数量有要求
备注	NA

 版权所有 © 华为技术有限公司

13.2.2.37.1 使用场景举例

YANG模型：使用incomplete-command	YANG模型：不使用incomplete-command

container c1{
leaf l1 {
hw:incomplete-command;
type string;
}
leaf l2 {
hw:incomplete-command;
type string;
}
leaf l3 {
type string;
}
}
	container c1 {
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	leaf l3 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：create 预期结果: 报错，生成不了命令行 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> <l3>val3</l3> </c1> </data> 操作类型：create 预期结果: c1 l1 val1 c1 l2 val2 c1 l3 val3	YANG to CLI结果: Path:/c1 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：create 预期结果: c1 l1 val1 c1 l2 val2 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> <l3>val3</l3> </c1> </data> 操作类型：create 预期结果: c1 l1 val1 c1 l2 val2 c1 l3 val3

13.2.2.38 incomplete-no leaf

名称	描述
标签名	incomplete-no
作用节点	leaf
父节点	无要求
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否

版权所有 © 华为技术有限公司

名称	描述
功能描述	用于delete操作， 确保使用该扩展项的leaf不作为最后的配置节点
使用场景	多参数命令行，删除时对输入的命令行参数数量有要求
备注	NA

13.2.2.38.1 使用场景举例

YANG模型：使用incomplete-no	YANG模型：不使用incomplete-no

container c1 {
leaf l1 {
hw:incomplete-no;
type string;
}
leaf l2 {
type string;
}
leaf l3 {
type string;
}
}
	container c1 {
	hw:render-view true;
	leaf l1 {
	type string;
	}
	leaf l2 {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果: Path:/c1 原始数据: Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> <l3>val3</l3> </c1> </data> Data: <data> <c1> <l1>val1</l1> </c1> </data> 操作类型：delete 预期结果: 报错，生成不了命令行 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：delete 预期结果: undo c1 l1 undo c1 l2	YANG to CLI结果: Path:/c1 原始数据: Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> <l3>val3</l3> </c1> </data> Data: <data> <c1> <l1>val1</l1> </c1> </data> 操作类型：delete 预期结果: undo c1 l1 Data: <data> <c1> <l1>val1</l1> <l2>val2</l2> </c1> </data> 操作类型：delete 预期结果: undo c1 l1 undo c1 l2

13.2.2.39 render-order-by-user list

名称	描述
标签名	render-order-by-user
作用节点	配置有ordered-by user的list节点
父节点	无要求

 版权所有 © 华为技术有限公司

名称	描述
支持方向	YANG->CLI
支持数据类型	无限制
是否可继承	否
功能描述	对于生成list实例有顺序要求的，在生成命令行时需要将配置先删除再重新新增。
使用场景	命令行有先后顺序要求
备注	NA

13.2.2.39.1 使用场景举例

YANG模型：使用render-order-by-user	YANG模型：不使用render-order-by-user

list li1 {
ordered-by user;
hw: render-order-by-user;
key "li1-key";
leaf li1-key {
type string;
}
}
	list li1 {
	key "li1-key";
	leaf li1-key {
	type string;
	}
	}

版权所有 © 华为技术有限公司

YANG to CLI结果: Path:/li1 原始数据: Data: <data> <li1> <li1-key>val1</li1-key> </li1> <li1> <li1-key>val2</li1-key> </li1> <li1> <li1-key>val5</li1-key> </li1> </c1> </data> Data: val3 before val5 <data> <li1> <li1-key>val3</li1-key> </li1> </data> 操作类型：merge 预期结果: undo li1 val5 li1 val3 li1 val5	YANG to CLI结果: Path:/li1 原始数据: Data: <data> <li1> <li1-key>val1</li1-key> </li1> <li1> <li1-key>val2</li1-key> </li1> <li1> <li1-key>val5</li1-key> </li1> </c1> </data> Data: val3 before val5 <data> <li1> <li1-key>val3</li1-key> </li1> </data> 操作类型：merge 预期结果: li1 val3

13.2.3 CLI 自定义扩展

13.2.3.1 背景

为了使CliDriver最大限度地成为一个通用引擎，而不是一个业务引擎，为了减少不必要的特例代码、兼容性代码，所以开发以下自定义扩展项，可以将YANG与CLI之间的正逆向转换逻辑转移到Specific Ne Driver（以下简称SND）包中实现。用户可以根据不同设备的不同CLI特性，通过编码的方式，定制出其中的转换逻辑，实现CliDriver在解析Cli与YANG功能上的灵活性。

13.2.3.2 扩展

 版权所有 © 华为技术有限公司

13.2.3.2.1 cli-custom-transform

范围

container，list，leaf。

功能

由SND包定制被标记cli-custom-transform的container节点、list节点以及leaf节点的正向YangToCli的转换逻辑。此扩展可能有3个参数，可以用逗号进行组合，分别是create, update和delete，参数指定了使此扩展生效的操作类型。如果当前节点有扩展项：

1. 对于create操作

   a. path指向当前节点，那么当前节点的转换逻辑由SND包给出。

   b. path指向父节点，那么父节点其他路径的节点不受影响，当前路径的转换逻辑由SND包给出。

   c. path指向子节点，那么子节点的转换逻辑由SND包给出。

   参考下面**例子**，给出2个例子如下：

   - 举例1：现有YANG文件结构c0/c1/c2/l2, c0/c1/l1, c0/c4/l4，标红的c2有create扩展项，path指向c1，路径c0/c4/l3与c1无关，就不会render。如果l1和l2都有值，那么l1的路径因为没有transform注解，会由CLIDriver正常render；而l2会由SND包render。
   - 举例2：现有YANG文件结构c0/c1/c2/c3/l3，标红的c2和c3都有create扩展项，path指向l3，那么CLIDriver将从C0开始从顶至下检查，发现C2有transform注解，那么SND中c2路径对应的自定义代码将会执行，c2以下的都不再解析。因为c0和c1是container类型父节点，所以它们不会由CLI生成前缀，而是由c2的自定义逻辑给出。

2. 对于delete操作

   a. path指向父节点，那么没有任何影响。

   b. path指向当前节点或子节点，那么转换逻辑由SND包给出。

3. 对于update操作

   a. 如果当前节点有扩展项，path指向当前节点，那么当前节点的转换逻辑由SND包给出。

   b. 如果当前节点有扩展项，path指向父节点，那么父节点其他路径的节点不受影响，当前路径的转换逻辑由SND包给出。

   c. 如果当前节点有扩展项，path指向子节点，那么子节点的转换逻辑由SND包给出。

例子

```
container c0 {
  container c4 {
    leaf l4 {
      type string;
    }
  }
  container c1 {
    leaf l1 {
      type string;
    }
    container c2 {
```

 版权所有 © 华为技术有限公司

```
    cli:cli-custom-transform 'create,update,delete'
      leaf l2 {
        type string;
      }
      container c3 {
      cli:cli-custom-transform 'create,update,delete'
        leaf l3 {
          type string;
        }
      }
    }
  }
}
```

在上方的YANG文件中，c2和c3 都标记了cli-custom-tranform扩展，c3是c2的子节点，c2是c1的子节点，c0是最顶层节点，c1和c4是兄弟节点。根据扩展参数，操作类型为create、update和delete，并且path末节点为c2/c3的父节点、c2/c3本身以及c2/c3的子节点的各种情况，其正向YangToCli转换逻辑都由SND包负责给出。

假设SND:

```
if path=/c0/c1/c2       return c1 c2custom  # c1前缀由SND负责给出
if path=/c0/c1/c2/c3    return c1 c2 c3custom  # 需给出c1和c2前置container
```

假设path:

```
path=/c0/c1
```

分析：path的末节点为c1，所以c4和l4不受影响。 c1及其所有子节点会进行正向YangToCli转换。l1的路径上没有cli-custom-transform注解，因此l1的转换由CLIDriver给出；c2有cli-custom-transform的注解，那么SND包会执行对应c2的自定义转换逻辑，即return c1 c2custom。c2的逻辑给出后，SND包不再执行c2子节点的转换逻辑，即C3不会再被SND包处理。

正常出参：

```
c1 l1 l1test
c1 c2 l2 l2test
c1 c2 c3 l3 l3test
```

自定义出参：

```
c1 l1 l1test
c1 c2custom
```

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:c1/cli-custom:c2" and request.oper_type == OperType.Value('CREATE'):
        out.data = "c1 c2custom"
    return out
```

上面是SND包的正向自定义转换代码，代码在yangToCli方法里，通过匹配request.path和操作类型request.oper_type拦截正确的请求。out.data存放的是生成的CLI命令。

13.2.3.2.2 cli-custom-read

范围

container, list, leaf。

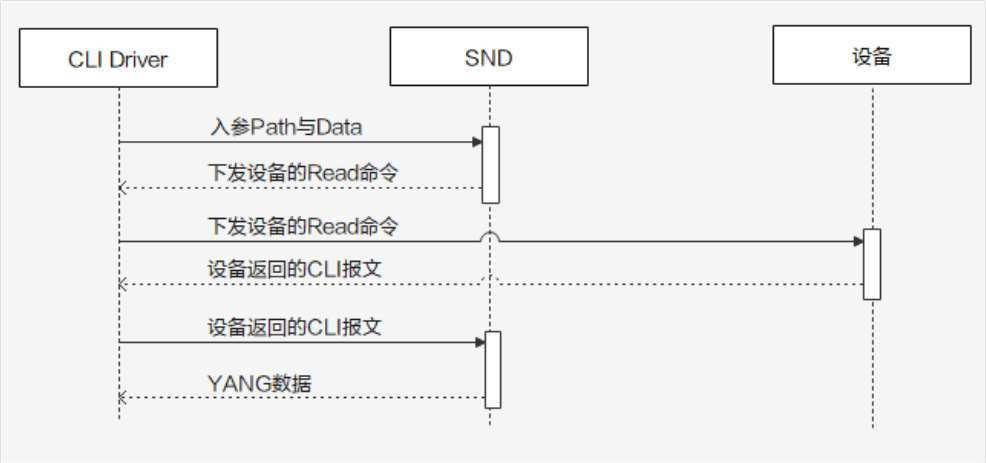
 版权所有 © 华为技术有限公司

功能

由SND包定制被标记cli-custom-read的节点的逆向CliToYang的转换逻辑。此扩展有三种参数，分别是two-step，one-step和one-step-no-parse，三个参数只能同时出现一个，不能叠加。它们区别如下：

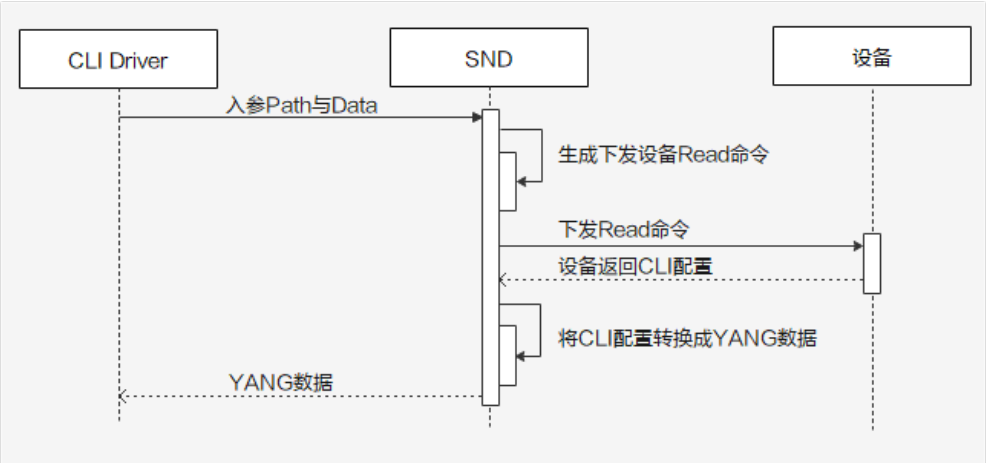
1. **two-step**

   read操作的正向生成命令行与逆向解析设备配置的两个转换逻辑都由SND负责给出，CLIDriver与SND包发生两次交互：CLIDriver调用SND包的yangToCli方法生成下发设备的CLI read命令，由CLIDriver将此命令下发设备，CLIDriver获取设备返回的配置报文后，再调用SND包的cliToYang方法，将配置报文转换成Yang数据，SND包最终将YANG数据返回给CLIDriver。

2. **one-step**

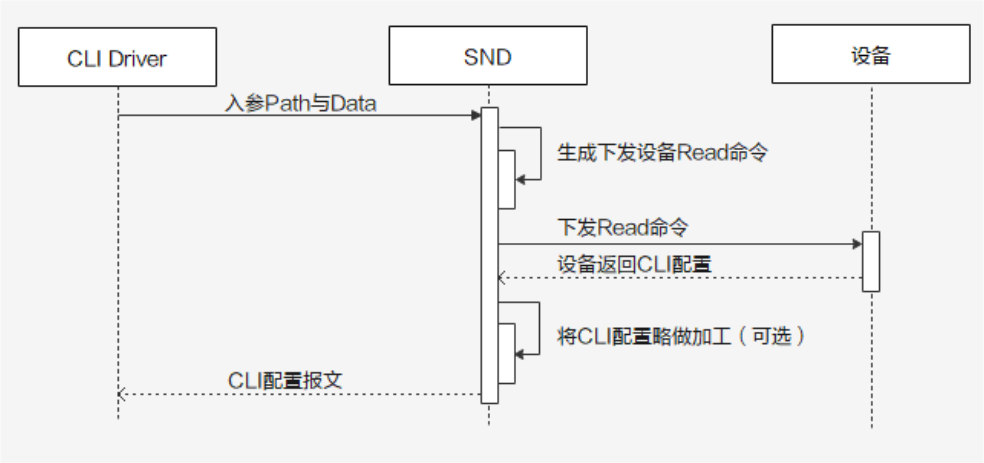
   CLIDriver直接向SND包发送read命令的path参数，调用SND包的cliToYang方法，由SND包负责生成下发设备的read命令，SND将命令下发到设备，获取到设备返回的配置报文后，将报文转换成Yang数据，最终返回给CLIDriver。在one-step流程中，CLIDriver与SND包只发生一次交互，只涉及cliToYang一个方法，并且与设备交互的是SND包而不是CLIDriver。

3. **one-step-no-parse**

   与one-step类似，CLIDriver向SND包发送read命令的path参数，调用SND包的cliToYang方法 。但是，当SND包下发命令到设备上，获取设备返回的配置报文后，可以对配置报文略做处理，随后将配置报文直接返回给CLIDriver，由CLIDriver负责CLI到YANG进行转换。SND包只负责下发命令的生成、调用设备、获取并加工返回的配置报文即可。

版权所有 © 华为技术有限公司



说明

- cli-custom-read 'one-step'和'one-step-no-parse'只能加在顶层节点上，非顶层节点不生效。
- cliToYang方法在返回out对象时，必须填写endOffset字段，表明入参字符串有多少被消费了，消费之后的offset值是多少。对于one-step与one-step-no-parse，endOffset返回-1；其他情况下，如果全部消费完，可以返回str(request.data).__len__()；如果没有全部消费完，endOffset根据实际消费的字符数加上入参request对象中的startOffset值得出。

例子

1. **two-step**

```
leaf router-id {
    cli:cli-custom-read 'two-step';
    type string;
}
```

在上方的YANG文件中，router-id leaf节点标记了cli-custom-read‘two-step’自定义扩展，那么对router-id的read操作的正向下发命令的生成操作由SND负责完成，对设备返回的CLI配置报文转换成YANG数据的操作也由SND包负责完成。

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "tellabs:router-id":
        out.data = "show running config | block router-id"
    return out
```

上面是在two-step中，SND包生成自定义read命令的代码。SND包通过request.path匹配到所需路径，生成自定义read的下发设备的命令。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "tellabs:router-id":
        lines = request.data.splitlines()
        if len(lines) > 1:
            router_id_content = str(lines[1]).split()[1]
            out.data = "<router-id xmlns=\"https://huawei.com/tellabs\">" + router_id_content + "</
router-id>"
    out.endOffset = str(request.data).__len__()
    return out
```

上面是在two-step中，SND包cliToYang方法将设备返回配置报文转换成YANG数据的操作。自定义代码通过request.path匹配路径，然后通过request.data获取设备配置报文，处理报文获取到所需router-id的数据，存放在router_id_content变量中，最后将YANG数据组装好，返回给CLIDriver。

2. **one-step**

版权所有 © 华为技术有限公司

```
leaf router-id {
    cli:cli-custom-read 'one-step';
    type string;
}
```

在上方的YANG文件中，router-id leaf节点标记有cli-custom-read 'one-step'，那么对router-id的read操作的所有CLI与YANG的转换逻辑由SND包的cliToYang方法给出。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    proxy = CommandProxy(request.neId, logger)
    proxy.send_command(["enable", "end", "configure terminal"])
    response_data = proxy.send_command(["show running-config | block router-id"], 120)
    response_body = response_data.response
    lines = response_body.splitlines()
    if len(lines) > 1:
        router_id_content = str(lines[1]).split()[1]
        out.data = "<router-id xmlns=\"https://huawei.com/tellabs\">" + router_id_content + "</router-
id>"
    out.endOffset = -1
    return out
```

在SND包中，我们从入参request的neId属性中获取网元ID，然后调用CommandProxy代理类将自定义的read命令下发至设备，获取设备返回的配置报文，并将配置报文转换成YANG数据。CLIDriver调用一次SND包，在cliToYang方法内完成read操作的CLI与YANG的全部自定义转换。

3. **one-step-no-parse**

```
leaf router-id {
    cli:cli-custom-read 'one-step-no-parse';
    type string;
}
```

在上方的YANG文件中，router-id标记有cli-custom-read 'one-step-no-parse'，那么相比one-step，SND包的cliToYang方法只要返回CLI配置报文即可，无需做CLI到YANG的转换操作。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    proxy = CommandProxy(request.neId, logger)
    proxy.send_command(["enable", "end", "configure terminal"])
    response_data = proxy.send_command(["show running-config | block router-id"], 120)
    response_body = response_data.response
    lines = response_body.splitlines()
    if len(lines) > 1:
        router_id_content = str(lines[1]).split()[1]
        out.data = "router-id" + router_id_content
    out.endOffset = -1
    return out
```

与one-step类似，one-step-no-parse在SND包cliToYang方法中编写自定义代码，通过CommandProxy代理类给设备下发自定义read命令，获取设备返回配置报文后，做适当数据处理，最后返回类似"router-id 10.2.3.4"这样的CLI配置报文给CLIDriver，CLIDriver负责根据YANG模型定义，将此CLI配置报文转换成YANG数据。one-step-no-parse让SND包自定义代码无需做CLI到YANG的转换操作。

13.2.3.2.3 cli-custom-processor

范围

顶层container, 顶层list。

 版权所有 © 华为技术有限公司

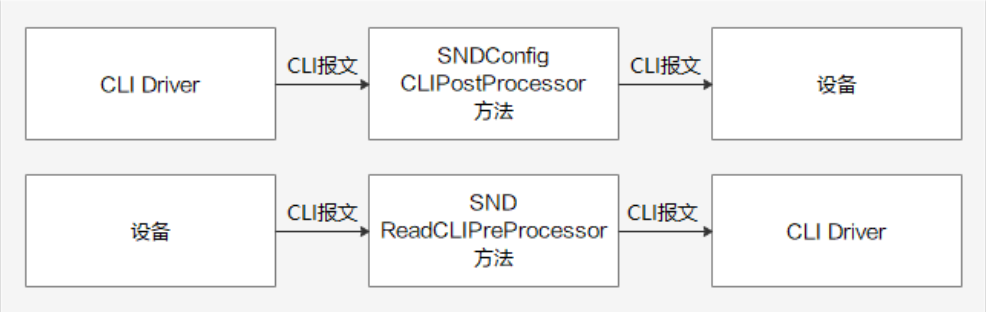
功能

这个扩展是前置后置处理，它有4个参数，分别是pre-read、post-config和post-config-by-line、post-config-rollback，可以用逗号分隔同时使用。

- pre-read就是将设备返回的CLI Config在返回给CLiDriver之前做一次前置自定义处理。一般使用在读的场景。
- post-config就是在CliDriver生成CLI命令后、下发设备前，对CLI做一个后置处理。一般使用在CUD下发场景，偶尔也用在读命令下发场景。
- post-config-by-line是将设备命令行按行后置处理。如果采用这样方式，用户自定代码不允许改变命令的行数和顺序。
- post-config-rollback当CLIDriver默认回滚机制无法满足一阶段设备的回滚特性时，用户可根据实际情况自定义回滚命令。

  流程如下：

  “CLIDriver正向 > CLI > (SND)后置处理 > 设备 > (SND)前置处理 > Config > CLIDriver逆向”。



此注解虽然配在顶层节点，由于父子继承关系，入参path如果指向任何子节点，都可以进入SND包前置后置处理方法中。用户可以针对不同的path及入参数据，采取不同的处理措施。

例子

YANG文件

```
container interfaces {
    cli:cli-custom-processor 'post-config,pre-read';
    list interface {
        container mpls {
            list l2vc {
                key role;
                leaf role {
                    typoe enumeration {
                        enum "primary";
                        enum "secondary";
                    }
                }
            }
        }
    }
}
```

在上方的YANG文件中，顶层节点interfaces由cli:cli-custom-processor 'post-config,pre-read'标记，所有的请求，如果入参path指向interfaces节点或它的子节点，其转换操作都会由SND包进行自定义前置处理或后置处理。当前的自定义需求是：

- 在下发命令时，如果CLI报文中l2vc节点的值包含primary，将primary删除。

版权所有 © 华为技术有限公司

- 在设备读取配置时，如果读取的l2vc节点的值为空，需要向CLI报文合适位置添加primary。

1. **pre-read**

```
def readCliPreProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
    cli_str = str(request.data)
    re_mpls = r'.*(undo\s+)?mpls l2vc.*'
    if re.search(re_mpls, cli_str):
        newlines = []
        lines = cli_str.splitlines()
        for line in lines:
            if re.search(re_mpls, line) and 'secondary' not in line:
                line += ' primary'
            newlines.append(line)
        cli_str = '\n'.join(newlines)
    out.data = cli_str
    return out
```

pre-read的自定义代码写在readCliPreProcessor方法中。自定义代码通过正则表达式匹配设备返回的CLI报文。如果l2vc节点的值为空，而且不包含secondary，就将primary添加到行尾。

2. **post-config**

```
def configCliPostProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
    cli_str = request.data
    re_mpls = r'.*(undo\s+)?mpls l2vc.+primary.*'
    if re.search(re_mpls, cli_str):
        newlines = []
        lines = cli_str.splitlines()
            for line in lines:
                if re.search(re_mpls, line):
                    line = line.replace('primary', '')
                newlines.append(line)
            cli_str = '\n'.join(newlines)
    out.data = cli_str
    return out
```

post-config的自定义代码写在configCliPostProcessor方法中。用户从request.data获取到需要后置处理的CLI报文，然后通过正则表达式等手段对CLI报文进行处理。如果报文中包含mpls l2vc primary，就将primary替换成空字符串。最后处理完，将新CLI报文赋值给out.data。

13.2.3.2.4 cli-custom-rpc

范围

rpc顶层节点。

功能

将rpc的YANG与CLI的正向与逆向转换过程分别委托给SND包中的yangToCli和cliToYang两个方法来实现：

1. 用户调用rpc的input接口，传入input的path与data，CLIDriver调用SND包的yangToCli方法，生成下发设备的命令。
2. CLIDriver下发命令到设备，获取设备的返回的报文。
3. CLIDriver调用SND包中的cliToYang方法，传入output的path以及设备返回的报文。

 版权所有 © 华为技术有限公司

例子

1. **YANG文件**

```
rpc resetIpStatistics {
    cli:cli-custom-rpc;
    input {
        leaf interface-name {
            type string;
        }
    }
    output {
        leaf response {
            type string;
        }
    }
}
```

在上方的YANG文件中，resetIpStatistics是rpc的顶层节点。cli-custom-rpc只能作用于顶层节点。入参path=/resetIpStatistics/input，调用SND 包的yangToCli方法，生成发向设备的CLI命令。出参path=/resetIpStatistics/output，调用SND包的cliToYang方法，生成YANG数据。

2. **SND包**

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:resetIpStatistics/cli-custom:input":
        out.data = "show running-config | block router-id"
    return out


def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:resetIpStatistics/cli-custom:output":
        out.data = "<resetIpStatistics xmlns=\"https://huawei.com/cli-custom\">" \
                   "<response>hello_world</response>" \
                   "</resetIpStatistics>"
        out.endOffset = str(request.data).__len__()
    return out
```

在上方的SND包代码中，RPC的正向自定义代码写在yangToCli方法里，request.path指向了RPC YANG的input节点，out.data的内容是希望下发设备的CLI命令。RPC的逆向自定义代码写在cliToYang方法里，request.path指向RPC Yang的output节点，out.data对应需要返回的YANG数据。

13.3 Jinja2 模板过滤器及方法

13.3.1 jinja 模板过滤器

Jinja开发手册请参考Jinja2官方文档，为了方便编码，aoc_api除了Jinja本身的过滤器支持之外，还新增了一些可能用到的方法及过滤器。

类型

API-5，模板类功能接口。

典型场景

用户使用jinja模板渲染引擎开发网络层到网元层的映射逻辑时，需要使用过滤器能力进行编码。

 版权所有 © 华为技术有限公司

接口功能

Jinja模板提供系列的过滤器能力，包括设置全局变量、转换neId、转换IP地址。

接口约束

1. 前提条件：无。
2. 注意事项：to_ne_id不能本地测试，因为本接口需要到AOC库查询设备信息。
3. 使用限制：无。
4. 接口之间的关联关系：无。

GET_GLOBAL 与 SET_GLOBAL 使用方法

使用GET_GLOBAL与SET_GLOBAL方法解决在Jinja内部需要使用全局变量的问题，使用方法：

```
<example>
   #初始化temp全局变量
   {{SET_GLOBAL('temp', 0)}}
   {%- for name in class%}
      <name>{{name}}</name>
      {{GET_GLOBAL('temp') + loop.index0}}
      {%- if loop.lase %}
         {{SET_GLOBAL('temp',GET_GLOBAL('temp' + loop.index0)}}
      {%- endif%}
   {%- endfor%}
   {{GET_GLOBAL('temp')}}
</example>
```

说明

使用GET_GLOBAL与SET_GLOBAL方法访问for循环索引的示例。

表 13-2 SET_GLOBAL 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
key	是	STRING	N/A	N/A	全局变量的key。
value	是	任意类型	N/A	N/A	全局变量的内容。

表 13-3 GET_GLOBAL 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
key	是	STRING	N/A	N/A	全局变量的key。

版权所有 © 华为技术有限公司

表 13-4 GET_GLOBAL 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
value	是	任意类型	N/A	N/A	全局变量的内容。

to_ne_id 过滤器使用方法

```
输入:{{HUAWEI_ROUTER| to_ne_id}}
输出:868e778e-153c-3afe-a02c-89678e31e3e4
```

说明

设备名和设备ID以具体环境为准。

表 13-5 to_ne_id 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
ne_name	是	STRING	N/A	N/A	设备的名称。

表 13-6 to_ne_id 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
ne_id	是	STRING	UUID	N/A	设备的ID。

ipaddr 过滤器使用方法

ipaddr过滤器封装了 netaddr Python库的部分功能。

- 计算ip地址：

```
输入:{{ '192.168.32.0/24' | ipaddr('0')}}
输出: 192.168.32.0/24
输入:{{ '192.168.32.0/24' | ipaddr('1')}}
输出: 192.168.32.1/24
输入:{{ '192.168.32.0/24' | ipaddr('-1')}}
输出: 192.168.32.255/24
输入: {{ '192.168.32.0/24' | ipaddr('100')}}
输出: 192.168.32.100/24
```

- 计算掩码：

```
输入: {{ '192.168.32.0/24' | ipaddr('netmask')}}
输出: 255.255.255.0
输入: {{ '192.168.32.0/25' | ipaddr('netmask')}}
输出: 255.255.255.128
```


版权所有 © 华为技术有限公司

```
输入: {{ '192.168.32.0/32' | ipaddr('netmask')}}
输出: 255.255.255.255
```

- 计算网关地址：

```
输入: {{ '192.168.32.1/32' | ipaddr('network')}}
输出: 192.168.32.1
输入: {{ '192.168.32.1/255.255.255.255' | ipaddr('network')}}
输出: 192.168.32.1
输入: {{ '192.168.32.1/24' | ipaddr('network')}}
输出: 192.168.32.0
```

- 计算广播地址：

```
输入: {{ '192.168.32.1/24' | ipaddr('broadcast')}}
输出: 192.168.32.255
```

- 计算主机掩码：

```
输入: {{ '192.168.32.1/24' | ipaddr('hostmask')}}
输出: 0.0.0.255
计算广播地址：
输入: {{ '192.168.32.1/25' | ipaddr('hostmask')}}
输出: 0.0.0.127
```

13.3.2 jinja 模板支持 no_delete 标签定制删除

当对某个业务层配置做删除动作并提交成功，NCE默认的删除行为是同时删除配置数据与数据源，jinja模板支持no_delete和no_delete_nested两种标签，提供了一种只删除数据源不删除数据的方法。

no_delete：表示打上标记的本层节点只会被删除数据源，不会被删除数据，子节点是可以被删除数据的。

no_delete_nested：表示打上标记的本层节点以及子节点都只会被删除数据源，不会被删除数据。

说明

- 数据源：数据源表示一个真实数据的引用。例如网元数据interface1，它如果被业务1和业务2同时分解过，那么它会存在两个引用（数据源），当业务1删除的时候，因为网元数据interface1还被业务2引用，该接口还存在不会实际删除。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的no_delete标签控制业务更新时的删除配置逻辑。

接口功能

Jinja模板提供的定制业务更新删除逻辑能力。

接口约束

1. 前提条件：无。
2. 注意事项：参考下文使用方法及说明。
3. 使用限制：参考下文使用方法及说明。
4. 接口之间的关联关系：无。

 版权所有 © 华为技术有限公司

使用方法

no_delete示例：

```
... ...
<ntpAuthKeyCfg xmlns:x="urn:huawei:ac" x:tag="no_delete">
... ...
```

no_delete_nested示例：

```
... ...
<system  xmlns:ns0="urn:huawei:ac" ns0:tag="no_delete_nested">
  <inform-timeout>2</inform-timeout>
  <inform-resend-times>2</inform-resend-times>
  <inform-pend-number>111</inform-pend-number>
</system>
... ...
```

说明

- 对于container节点，只支持no_delete_nested。
- no_delete下面可嵌套no_delete或者no_delete_nested，no_delete_nested下面不允许嵌套no_delete标签。
- neid不支持no_delete和no_delete_nested。

13.3.3 jinja 模板支持 create/merge/delete/replace 操作

用户在模板里面定制的网元数据的新增或删改操作。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的操作码标签直接操作网元数据，不经过差异对比。

接口功能

Jinja模板提供定制网元的新增或删改逻辑能力。

接口约束

1. 前提条件：无。
2. 注意事项：参考下文使用方法。
3. 使用限制：参考下文使用方法。
4. 接口之间的关联关系：无。

使用方法

create：创建一个不存在的节点，create是SSP默认操作；create通常不使用在container节点上。

示例：

版权所有 © 华为技术有限公司

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" ns0:operation="create">2</inform-timeout>
 ... ...
```

merge：修改一个已存在的节点或者创建一个不存在的节点，merge是SSP默认操作。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" ns0:operation="merge">2</inform-timeout>
 ... ...
```

delete：删除一个已存在的节点，如果节点不存在不报错；如果delete操作作用在容器节点上，其子节点不能有其它显示声明操作，包括merge、replace、no-delete、no-delete-nested；delete操作不能作用于list的key节点上。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" ns0:operation="delete">2</inform-timeout>
 ... ...
```

replace: 替换一个已存在的节点或创建一个不存在的节点；如果replace操作作用在容器节点上，其子节点不能有merge或replace操作；replace操作不能作用于list的key节点或leaflist节点上。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" s0:operation="replace">2</inform-timeout>
... ...
```

13.3.4 jinja 模板支持 order 调整顺序

用户在模板里面通过定制带order-by-user属性调整网元数据的顺序。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的order-by-user属性操作不同实例间的顺序调整。

接口功能

Jinja模板提供的网元数据实例间顺序调整能力。

接口约束

1. 前提条件：无。
2. 注意事项：order只能作用在声明order-by user的schema节点。
3. 使用限制：无。
4. 接口之间的关联关系：无。

 版权所有 © 华为技术有限公司

使用方法

对于打ordered-by属性标签的list或者leaflist节点（请参看RFC7950-YANG1.1-7.7.1），SSP支持insert操作来调整顺序。

示例：

```
... ...
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
<rule-list xmlns:ns0="urn:ietf:params:xml:ns:yang:1" ns0:insert="after" ns0:key="[name='name2']">
      <name>name1</name>
   </rule-list>
</nacm>
... ...
```

说明

假设已经存在name为name2的rule-list节点，表示merge一个name为name1的 rule-list节点，并且位置在name2后。

版权所有 © 华为技术有限公司