数通网络开放可编程
V100R020C10
开发指南

文档版本 01

发布日期 2021-02-05

华为技术有限公司

版权所有 © 华为技术有限公司 2021。 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默示的声明或保证。

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

前言

概述

本文档介绍了基于NCE 提供的开放可编程功能进行设备对接、快速构建业务等二次开发的操作指导。

读者对象

本文档主要适用于以下工程师：

- 系统管理员
- 维护工程师
- 技术支持工程师

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

版权所有 © 华为技术有限公司

图形界面元素引用约定

本文中可能出现下列图形界面元素，它们所代表的含义如下。

格式	意义
“ ”	带双引号“ ”的格式表示各类界面控件名称和数据表，如单击“确定”。
>	多级菜单用“>”隔开。如果选择“文件 > 新建 > 文件夹”，表示选择“文件”菜单下的“新建”子菜单下的“文件夹”菜单项。

命令行格式约定

本文可能出现下列命令行格式，它们所代表的含义如下。

格式	意义
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用**加粗**字体表示。
斜体	命令行参数（命令中必须由实际值进行替代的部分），采用*斜体*表示。
[ ]	表示用“[ ]”括起来的部分在命令配置时是可选的。
{x | y |...}	表示从两个或多个选项中选取一个。
[x | y | ...]	表示从两个或多个选项中选取一个或者不选。
{ x | y | ... } *	表示从两个或多个选项中选取多个，最少选取一个，最多选取所有选项。
[x | y | ...] *	表示从两个或多个选项中选取多个或者不选。

修改记录

文档版本	发布日期	修改说明
02	2021-07-30	第二次正式发布。
01	2021-02-28	第一次正式发布。

 版权所有 © 华为技术有限公司

目 录

前言 ii
1 基础知识 1
1.1 NETCONF 1
1.2 YANG 模型 4
1.3 Jinja2 模板 7
1.4 Python 编程基础 8
1.5 软件包结构 8
1.6 日志记录 9
1.6.1 查看日志 9
1.6.2 设置日志级别 9
2 开发流程 11
3 开发准备 12
4 开发环境部署 13
4.1 安装 Python 解析器 13
4.2 安装开放可编程 SDK 14
4.3 安装集成开发环境 14
4.4 安装 Gpg4Win 及生成签名用密钥 15
5 开发设备原子驱动包 17
5.1 创建原子驱动包模板 17
5.2 开发原子驱动包 18
5.2.1 编辑软件包配置文件 18
5.2.2 编写设备 YANG 模型 20
5.2.3 开发代码 24
5.2.3.1 引入必要的头文件 24
5.2.3.2 编写设备纳管定制 25
5.2.3.3 编写设备类型能力定制 26
5.2.3.3.1 配置设备驱动数据 26
5.2.3.3.2 设置业务自定义处理 30
5.2.3.3.3 设置数据前置处理 31
5.2.3.3.4 设置数据后置处理 31
5.2.3.3.5 设置数据联动 31

 版权所有 © 华为技术有限公司

5.2.3.3.6 获取 syncId 32
5.2.3.4 编写设备特性能力定制 32
5.2.3.5 管理资源 34
5.3 验证原子驱动包 35
5.3.1 验证 YANG 文件 35
5.4 编译原子驱动包 35
6 开发网络业务插件包 37
6.1 创建业务包模板 37
6.2 开发业务包 38
6.2.1 编辑软件包配置文件 38
6.2.2 开发业务 YANG 模型 41
6.2.3 开发业务包代码 45
6.2.4 开发 Jinja2 模板 46
6.3 验证业务包 49
6.3.1 验证 YANG 文件 49
6.3.2 执行单元测试 49
6.4 编译业务包 51
7 开发业务还原业务插件包 53
7.1 创建业务包模板 53
7.2 开发业务包 54
7.2.1 编辑软件包配置文件 54
7.2.2 开发业务 YANG 模型 57
7.2.3 开发业务包代码 62
7.2.4 开发 Jinja2 模板 64
7.3 验证业务包 65
7.3.1 验证 YANG 文件 65
7.3.2 执行单元测试 65
7.4 编译业务包 69
8 开发 RPC 业务插件包 71
8.1 创建业务包模板 71
8.2 开发业务包 72
8.2.1 编辑软件包配置文件 72
8.2.2 开发业务 YANG 模型 74
8.2.3 开发业务包代码 78
8.2.4 开发 Jinja2 模板 80
8.3 验证业务包 80
8.3.1 验证 YANG 文件 80
8.3.2 执行单元测试 80
8.4 编译业务包 81
9 导入并安装软件包 83
9.1 登录系统 83

 版权所有 © 华为技术有限公司

9.2 导入并激活软件包 85
10 软件包升级说明 88
11 业务安全 138

 版权所有 © 华为技术有限公司

1 基础知识

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 NETCONF

1.2 YANG模型

1.3 Jinja2模板

1.4 Python编程基础

1.5 软件包结构

1.6 日志记录

1.1 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。本小节主要基于RFC6241对NETCONF协议提供简要的引导。

基本概念

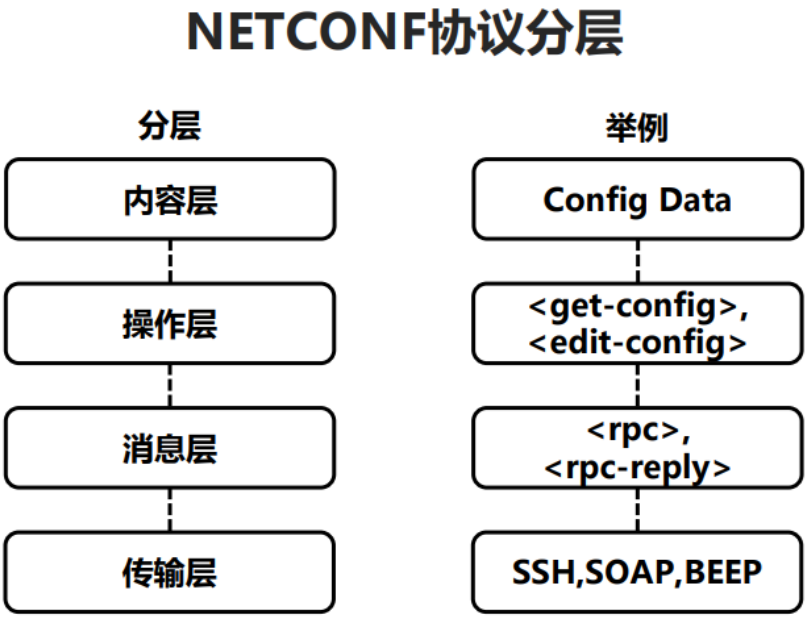
- 配置数据（Configuration data）：将系统从其初始默认状态转换为当前状态所需的一组可写数据。
- 状态数据（State data）：系统上不是配置数据的附加数据，如只读状态信息和收集的统计数据。
- 协议操作（Protocooperation）：NETCONF协议中使用的特定远程过程调用。
- 远程过程调用（RPC）：通过交换<rpc>和<rpc-reply>消息来实现。
- 服务器（Server）：执行由客户端调用的协议操作。另外，服务器可以发送通知给客户端。
- 客户端（Client）：调用服务器上的协议操作。另外，客户端可以订阅从服务器接收通知。

关于更多的NETCONF相关概念，可以参考RFC6241中的**术语**一节。

 版权所有 © 华为技术有限公司

协议分层

图 1-1 NETCONF 协议分层



- 内容层：内容层则是需要传递的配置数据和通知数据。
- 操作层：操作层定义了一个XML编码的操作集用于实现RPC操作。
- 消息层：消息层负责按照传输协议的要求实现简单的成帧机制，用于包裹RPC消息或通知消息。
- 传输层：传输层提供客户端与服务器间的通信路径。通过这些满足相关条件的通信路径实现NETCONF消息的分层机制。

关于数据交互及传输协议的要求的更多信息，可以参考RFC6241中的协议概述、能力、配置和状态数据分离及传输协议的基本要求等章节。

RPC 模型

NETCONF协议使用基于RPC的通信模型。NETCONF对等体使用<rpc>和<rpc-reply>元素来提供NETCONF请求和响应的与传输协议无关的数据帧。

- <rpc>

  用于封装从客户端发送到服务器的RPC请求。例如：

  ```
  <rpc message-id="101"
       xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <some-method>
      <!-- method parameters here... -->
    </some-method>
  </rpc>
  ```

  其中，message-id为强制属性。这是由RPC请求的发送者选择的一个字符串。RPC请求的接收者不解析该字串，仅在返回该请求的响应时，使用该字符串作为<rpcreply>消息的message-id属性。

- <rpc-reply>

  用于封装由服务器返回的RPC响应。例如：

版权所有 © 华为技术有限公司

```
<rpc-reply message-id="101"
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data>
    <!-- contents here... -->
  </data>
</rpc-reply>
```

其中，message-id为强制属性。该属性的值与对应RPC请求的同名属性的值保持一致。另外，RPC响应消息还必须原样返回对应请求消息中的其它属性。

- <rpc-error>

  如果在处理<rpc>请求期间发生错误，<rpc-error>元素将在<rpc-reply>消息中发送。例如：

  ```
  <rpc-reply
      xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <rpc-error>
      <error-type>rpc</error-type>
      <error-tag>missing-attribute</error-tag>
      <error-severity>error</error-severity>
      <error-info>
        <bad-attribute>message-id</bad-attribute>
        <bad-element>rpc</bad-element>
      </error-info>
    </rpc-error>
  </rpc-reply>
  ```

  在上述示例中，rpc-reply未携带message-id属性。这是因为本端在接收到RPC请求时未携带message-id属性，这会导致本端直接返回包含<rpc-reply>的RPC响应。只有在这种情况下，<rpc-reply>消息才会不携带message-id属性。

- <ok>

  如果在处理<rpc>请求期间没有发生错误或警告，<ok>元素将在<rpc-reply>消息中发送。 例如：

  ```
  <rpc-reply message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <ok/>
  </rpc-reply>
  ```

  关于RPC模型更多信息，可以参考RFC6241中RPC模型章节的内容。

RPC 操作

NETCONF协议提供一组底层操作来管理设备配置和检索设备状态信息。基本协议提供了检索，配置，复制和删除配置数据存储的操作。可以根据设备宣告的功能，提供了额外的操作。

基本协议包括以下协议操作：

- get：获取状态数据和running数据库的配置数据
- get-config：获取某数据库的配置数据
- edit-config：修改、添加、删除配置数据
- copy-config：替代某数据库中的所有数据
- delete-config：删除某数据库中的所有数据
- lock：对某数据库上锁
- unlock：对某数据库解锁
- close-session：结束自身的NETCONF会话
- kill-session：强行结束其他的NETCONF会话

下面是一个简单的例子：

 版权所有 © 华为技术有限公司

在该RPC请求中，使用<get-config>操作向服务器请求当前运行时配置中的<users>节点的内容。

```
<rpc message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
   <get-config>
      <source>
         <running/>
      </source>
      <filter type="subtree">
         <top xmlns="http://example.com/schema/1.2/config">
            <users/>
         </top>
      </filter>
   </get-config>
</rpc>
```

关于RPC操作的更多信息，可以参考RFC6241中的协议操作章节的内容。

1.2 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。本小节主要基于RFC7950对YANG模型提供简要的介绍。

基本概念

YANG将数据的分层组织模型化为一个树，其中每个节点都有一个名称，或者一个值或一组子节点。YANG提供了对节点的清晰简洁的描述，以及这些节点之间的交互。下面对YANG模型中常用的概念进行简要介绍：

- 模块和子模块

  YANG数据模型是通过模块来定义的，模块是相关定义的集合。一台支持YANG模型的网元上可以实现多个模块，并对同样的数据提供多种不同的视图；也可以只实现一个模块来包含所有数据。

  根据模块设计的不同，模块中的部分内容是可以独立出来成为子模块的。在模块中可以使用import子句声明对其它模块及其子模块的调用；也可以使用include子句来声明属于该模块的子模块。

  每个文件/模块要有一个特定的命名空间，有全球唯一的URI。标准YANG的命名空间： "urn:ietf:params:xml:ns:yang:模块名";

  在模块声明内部使用时，prefix声明定义了当这个模块被引入时将要被使用的前缀。在import声明内部使用时，prefix声明定义了当访问引入模块的定义时，将要使用的前缀。

  下面的例子中定义了一个名为HVPNService的模块：

  ```
  module HVPNService {
     namespace "http://example.com/HVPNService";
     prefix CreateInterfaceService;

     import huawei-ac-applications {
        prefix app;
     }

     import huawei-ac-nes-device {
        prefix device;
     }

     description
  ```

 版权所有 © 华为技术有限公司

```
    "The module for HVPNService example.";

  revision 2018-12-09 {
    description "Initial revision.";
  }
  augment "/app:applications" { … }
}
```

- 叶

  叶包含简单的数据（如整数和字符串）。它只有一个特定类型的值，没有子节点。

  叶用leaf来定义，示例如下：

```
leaf host-name {
  type string;
  description "Hostname for this system";
}
```

- 叶列表

  叶列表是只有一个叶节点的列表，没有key说明，只能创建和删除。作用相当于只有一个节点的表。

  叶列表用Leaf-list定义，示例如下：

```
leaf-list domain-search {
  type string;
  description "List of domain names to search";
}
```

- 容器

  容器用来为节点分组，它只有子节点，没有值。子节点可以是容器、叶、叶列表和列表。

  容器用container来定义，示例如下：

```
container system {
  container login {
    leaf message {
    type string;
    description "Message given at start of login session";
    }
  }
}
```

- 列表

  列表是一系列数据节点的集合，由key标识，并由unique来定义哪些数据节点需要保持唯一。子节点可以是容器、叶和叶列表。

  列表用list来定义，示例如下：

```
list user {
  key "name";
  config true;
  description "This is a list of users in the system.";
  leaf name {
    type string;
  }
  leaf type {
    type string;
  }
  leaf full-name {
    type string;
  }
}
```

- 配置数据和状态数据

  YANG模型中config语句可以用来区分配置数据和状态数据。当一个节点被标记为config false时，其子层被标记为状态数据；当一个节点被标记为config true时，其子层被标记为配置数据。

 版权所有 © 华为技术有限公司

- 原生类型和派生类型

  在定义叶和叶列表节点时，需指定值的类型。YANG原生支持的类型有binary、bits、boolean、decimal64、empty、enumeration、identityref、int8/16/32/64、leafref、string和unit8/16/32/64。除此之外，还可以使用typedef语句从原生类型和其它派生类型中派生新类型。

  例如：

```
typedef percent {
  type unit8 {
    range "0 .. 100";
  }
}
```

  之后，就可以在模型中使用这个名为percent的新类型了：

```
container group-statistics {
  list group {
    key "name";
    leaf name {
      type string;
    }
    leaf proportion {
      type percent;
    }
  }
}
```

- 模型增扩

  可以用augment在一个模块中定义对另一个模块某模型的增广模型，比如在一个contrainer或list中扩一些属性。有单独的报文格式。

```
augment "/if:interfaces/if:ifEntry" {
    when "if:ifType='ds0'";
    leaf ds0ChannelNumber {
        type ChannelNumber;
    }
}
```

- 分组和选择

  如果一组节点可以在不同的模块或节点中重复使用，可以将这一组节点定义为分组，然后在需要引用的模块或节点中使用uses语句引用。当需要定义一组不能一起出现的模式节点集合时，可以考虑使用choice和case语句。choice声明定义了一系列替代选择，同一时间只能有一项存在。一个choice节点包含许多分支，分支用case定义声明。有点像C语言中的case语句。

  在下面这个例子中，定义了一个名为ip-port的分组，内含两个叶节点：一个为ip地址，另一个为端口。

```
grouping ip-port {
  leaf ip {
    type inet:ip-address;
    description "ip address";
  }
  leaf port {
    type inet:port-number;
    description "port number";
  }
}
```

  之后，就可以在模型中使用这个分组了。在这个例子中，还使用了选择结构。

```
choice terminal {
  case source {
    container source {
      uses ip-port;
    }
  }
  case destination {
```

 版权所有 © 华为技术有限公司

```
    container destination {
      uses ip-port;
    }
  }
}
```

- 操作和动作

  在YANG模型中，操作是指在模块顶层使用RPC语句定义的模块级的功能定义；动作则是指在容器或列表中定义的针对所在容器和列表执行的具体动作。

```
module server-farm {
  namespace "urn:example:server-farm";
  prefix "sfarm";
  import ietf-yang-types {
    prefix yang;
  }
  rpc reset-specified-servers {
    input {
      leaf-list servers {
        type "inet:ip-address";
      }
    }
  }
  list servers {
    key "name";
    leaf name {
      type string;
    }
    leaf address {
      type inet;
    }
    leaf location {
      type string;
    }
  action reset {
    input {
      leaf reset-at {
        type yang:date-and-time;
      }
    }
    output {
      leaf complete-at {
        type yang:date-and-time;
      }
    }
  }
 }
}
```

  在这个例子中，定义一个名为server-farm的模块。在这个模块里定义了一个RPC操作。该操作的输入是一个ip地址集合，用来指定需要重启的服务器集合。在servers列表中，定义了reset动作。该动作的输入是启动reset的时间，输出是reset完成的时间。

1.3 Jinja2 模板

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。本小节将通过一个简单的例子结合Jinja2的官方文档提供简要的引导。简单地说，Jinja2模板是一个包含了标签、变量和表达式的文本文件。文件中的标签用来控制模板的内在逻辑，变量和表达式则会在模板被渲染时被替换成对应的值。

下面是一个简单的Jinja2模板示例：

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
```

 版权所有 © 华为技术有限公司

```
    <ne>
        <neid>{{neName.deviceName| to_ne_id}}</neid>
       <ifm xmlns="urn:huawei:yang:huawei-ac-ne-ce-ifm">
          <interfaces>
             {% for info in ifm.interfaces.interface %}
             <interface>
                <ifName>{{ info.ifName }}</ifName>
                <ifDescr>{{ info.ifDescription }}</ifDescr>
                <ifmAm4>
                   <am4CfgAddrs>
                      <am4CfgAddr>
                         <ifIpAddr>{{ info.ifIP.ipv4Addr }}</ifIpAddr>
                         <subnetMask>255.255.255.0</subnetMask>
                      </am4CfgAddr>
                   </am4CfgAddrs>
                </ifmAm4>
             </interface>
             {% endfor %}
          </interfaces>
       </ifm>
    </ne>
  </nes>
</inventory-cfg>
```

在这个模板中出现了如下两种结构，分别对应控制结构和变量：

- {% ... %}

  控制结构语句标签，用于包裹Python脚本，处理if/elif/else、for循环以及宏和块的控制结构等，在本例中使用的是for循环。关于语句的详细介绍，可以参考Jinja模板设计指南中的控制结构章节的内容。

- {{ ... | ... }}

  表达式标签，用于包裹Python变量及运算表达式，在本例的最里层for循环里用于包裹迭代对象info的相关属性。关于表达式的详细介绍，可以参考Jinja模板设计指南中的表达式章节的内容。

另外，在本例的<neid>标签里，还使用了过滤器to_ne_id。在Jinja2模板中，过滤器用来指定修改目标变量的方法。在使用了过滤器的表达式中会有一个“|”符号。该符号之前的部分为目标变量，该符号之后的部分为修改目标变量的方法。Jinja2提供了一些常用的过滤器，也可以根据需要自行编写过滤器。

关于Jinja2模板引擎的更多介绍，可以参考Jinja官方文档中的Jinja模板设计指南。

1.4 Python 编程基础

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。网上相关的教程很多，请用户自行学习，这里就不做介绍了。

1.5 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

 版权所有 © 华为技术有限公司

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

1.6 日志记录

1.6.1 查看日志

- 微服务日志路径：/opt/oss/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
- Python运行日志路径：/opt/oss/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

1.6.2 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

/opt/oss/envs/Product-ExtendedPkgRTService/{$version}/venv-{$group_name}/Python/{$package_name}/resources

版权所有 © 华为技术有限公司

说明

{$version}为当前环境运行的ExtendedPkgRTService微服务的版本号。
{$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-2.0.0-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。

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

4.1 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。

步骤1 在“此电脑”上单击右键菜单“属性”，打开“系统”界面。

 版权所有 © 华为技术有限公司

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**开放可编程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。您可以从配套软件资源包中获取。
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”。

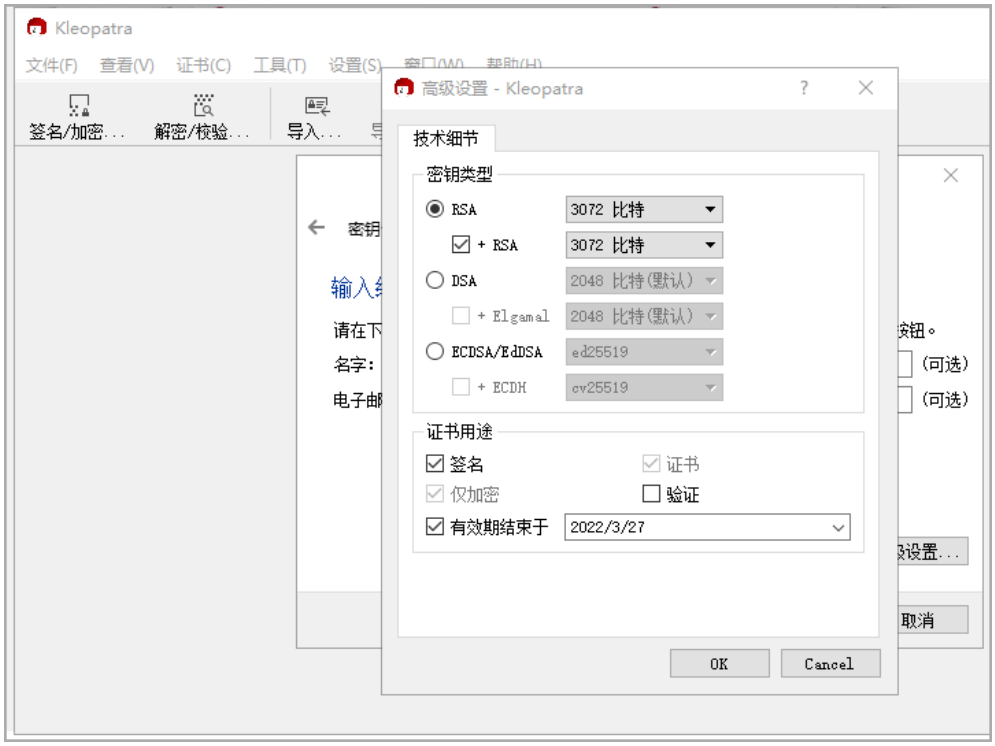
步骤4 单击“高级设置”，在“密钥类型区域”，选择“RSA”，并将密钥长度设置为“3072比特”。单击“OK”关闭当前窗口。然后单击“下一步”。

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

 版权所有 © 华为技术有限公司

5 开发设备原子驱动包

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。

本章节以开发华为路由器设备NE8000 M8的NETCONF网元驱动包为例，介绍详细的开发过程。

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。

开发环境准备

产品支持Python开发设备原子驱动包。Python开发环境，请按照开发环境部署章节的描述进行准备。

5.1 创建原子驱动包模板

5.2 开发原子驱动包

5.3 验证原子驱动包

5.4 编译原子驱动包

5.1 创建原子驱动包模板

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”。

 版权所有 © 华为技术有限公司

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5-1设置设备软件包的相关属性。

表 5-1 NE8000 M8 软件包属性

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	NetEngine 8000 M8
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“OK”，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

5.2 开发原子驱动包

本节详细介绍开发SND包全过程。

5.2.1 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。

版权所有 © 华为技术有限公司

表 5-2 pkg 文件参数表

参数名称	含义	必选/可选
name	软件包名称	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	对包的说明	可选
package-type	包类型	必选
group	包分组信息，主要用作沙箱进程隔离。	可选
producer	软件包的提供商。目前对于 snd 包和 gnd 包，只支持该字段为"huawei"的包。	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，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，则必须包含name（必选）， version（必选）字段。	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”。	可选
snd-id	"package-type"为 snd包才有值，id信息区分不同snd包，不同名的包不能使用相同的snd-id。	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vendor（必选），device-type（必选），device-version（必选）字段。	可选
service-name	ssp包特有值，服务名，不同名的包不能使用相同的service-name。	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。snd包时，snd-id组合要唯一，不同名的包不能使用相同的snd-id。如果配置，则需包含type（必选）， key（必选）， java-class-name（可选），Python-class-name（可选）， groovy-class-name（可选）， template（可选）字段。EasyMap，type设置为mapping；RPC，type设置为service-rpc；Discover，type设置为discover。多项功能组合时可以设置多个hooks。	必选

检查文件内容，补充属性设置，如“device-version”、“nce-min-versions”等，最终编辑文件示例如下：

 版权所有 © 华为技术有限公司

```
{
    "name": "NE8000M8_SND",
    "version": "1.0.0",
    "description": "NE8000M8_SND",
    "package-type": "snd",
    "producer": "HUAWEI",
    "nce-min-versions": [
        "1.0.0"
    ],
    "snd-id": "NE8000M8_SND",
    "devices": [
     {
       "vendor": "HUAWEI",
       "device-type": "NetEngine 8000 M8",
       "device-version": "V800R012C00SPC300"
     }
    ],
    "hooks": [
     {
       "type": "snd",
       "key": "ecs-driver",
       "python-class-name": "com.huawei.controller.devicetype.ne8000m8snd.NE8000M8NETCONF"
     }
    ]
}
```

说明

如需设置netconf-keepalive-info，需要在pkg.info中引入如下hook：

```
    {
      "type": "snd",
      "key": "ecs-connect-agent",
      "java-class-name": "com.huawei.enterprise.naas.neagent.service.NeAgentImpl"
    }
```

----结束

5.2.2 编写设备 YANG 模型

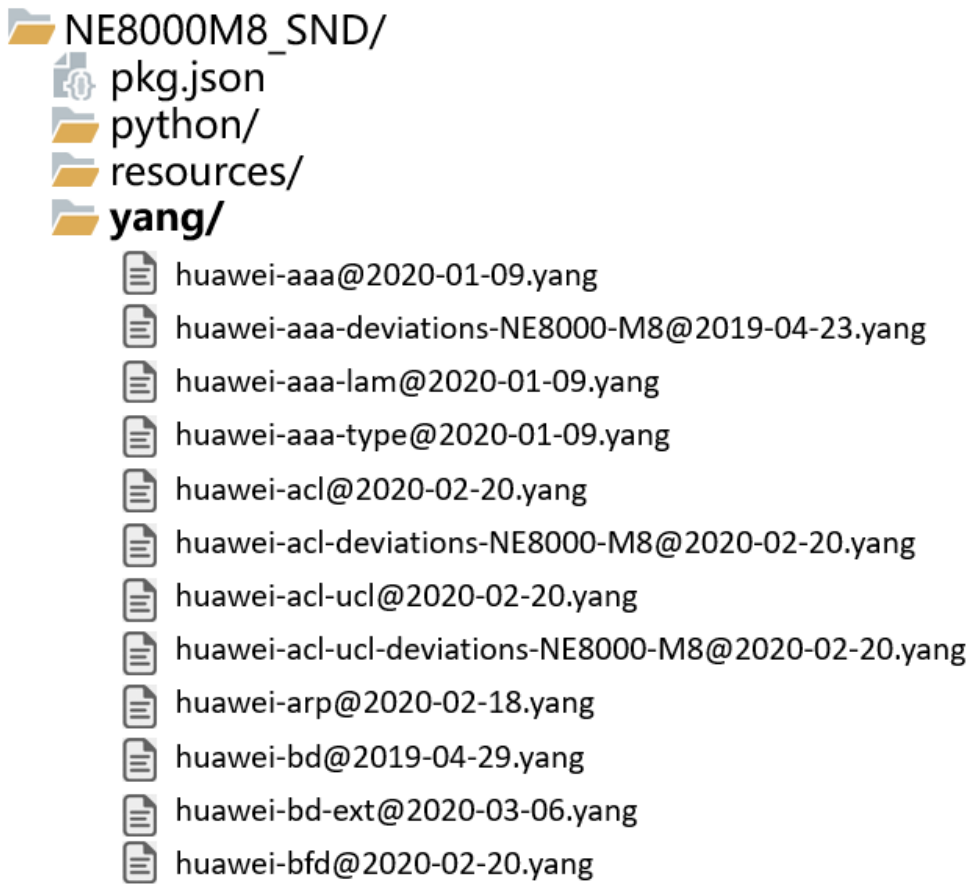
SND包中的YANG不参与编译，用于描述设备的配置能力、设备的北向接口以及生成南向报文。用户根据SND包类型，获取或编写对应的YANG模型，替换掉软件包中默认的YANG模型文件。

- NETCONF SND：将软件包中的默认YANG模型文件删除，从设备厂商的官方网站获取NETCONF协议的YANG模型。
- CLI SND：请获取《开放编程框架CLI Driver用户指导手册》，根据文档指导开发CLI协议的YANG模型。
- NETCONF&CLI SND：只需要从设备厂商的官方网站获取NETCONF协议的YANG模型。
- RESTCONF SND：从厂商的官方网站获取RESTCONF协议的YANG模型。

本示例NE8000 M8软件包为NETCONF SND包，请用户将SND模板中默认YANG文件删除，将从设备商获取的NE8000M8设备的YANG模型文件放置到yang文件夹下，如图5-1所示。

版权所有 © 华为技术有限公司

图 5-1 yang 文件路径



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。

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

版权所有 © 华为技术有限公司

节点名称	父节点	属性	说明
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：

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

 版权所有 © 华为技术有限公司

```
  <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",
         "method":"exec"
      }
      ]
   }
   ]
}
```


版权所有 © 华为技术有限公司

5.2.3 开发代码

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

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，软件包只需要编写设备纳管定制代码、设备驱动数据代码，同时设备特性能力无定制，默认全量差异发现。

5.2.3.1 引入必要的头文件

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。

在代码最前面，首先需要引入必要的头文件中的相关类。请用户根据SND包类型、业务逻辑，参考示例代码导入需要的类。

本示例NE8000 M8软件包需要引入以下类，具体类的介绍请参考下文描述。

```
from aoc.snd.netconfsnd import NetconfSND
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo
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, PROTECTED_MODE
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```

公共导入类

与SND包类型无关，公共导入的类参考如下所示代码。

```
# 可选，复写设备sysoid注册接口时需导入SysoidInfo类，配置原子驱动包对应的设备类型、厂商、款型等信息。
from aoc.snd.snd_model_pb2.sysoidinfo_pb2 import SysoidInfo

# 必选，导入ConnectInfos, ProtocolEntity, DEFAULT_CONNECT, PRIMARY_CONNECTION, HelloEntity类，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

# 可选，导入SINGLE_CHANNEL类，配置设备连接能力时使用
from aoc.snd.snd_model_pb2.channelInfo_pb2 import SINGLE_CHANNEL

# 可选，导入CommonDriverInfo类，编写设备类型时使用，基于通用能力定制，用于定制设备配置报文中的一些操作类型，如设备不支持create需要用merge；设备的同步方式是否是先删除后添加等
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import CommonDriverInfo

# 可选，导入FeatureCfgsMsg, Feature, Function类，编写设备特性能力时使用
from aoc.snd.snd_model_pb2.FeatureCfgs_pb2 import FeatureCfgsMsg, Feature, Function

# 可选，导入EcsDbConfigOut, EcsDbConfig类，配置数据联动时使用
from aoc.snd.snd_model_pb2.ecsDbProcess_pb2 import EcsDbConfigOut, EcsDbConfig
```

NETCONF SND 包需导入的类

开发NETCONF SND包时需要导入的类参考如下所示代码。

```
# 必选，开发NETCONF SND包时，需要引入NetconfSND类，原子驱动包父类，默认的配置已在父类中实现，针对定制化的配置需要在网元驱动包中继承“NetconfSND”，复写父类中的方法
from aoc.snd.netconfsnd import NetconfSND

# 可选，导入NetconfDriverInfo类，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，如是否支持两阶段，是否支持set模式等
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```


版权所有 © 华为技术有限公司

CLI SND 包需导入的类

开发CLI SND包时需要导入的类参考如下所示代码。

```
# 必选，开发CLI SND包时，需要引入CliSND类
from aoc.snd.clisnd import CliSND

# 可选，导入CliDriverInfo类，编写设备类型时使用，基于Cli协议类型能力定制
from aoc.snd.snd_model_pb2.cliDriver_pb2 import CliDriverInfo

# 必选，CliCustomTransformOutput自定义CLI和YANG转换输出，OperType自定义操作类型，
CliToYangCustom自定义CLI转换YANG逻辑
from aoc.snd.snd_model_pb2.cliTransform_pb2 import CliCustomTransformOutput
from aoc.snd.snd_model_pb2.cliTransform_pb2 import OperType
from .clitoyang import CliToYangCustom
```

NETCONF&CLI SND 包需导入的类

开发NETCONF&CLI 双协议SND包时需要导入的类参考如下所示代码。

```
# 必选，开发NETCONF&CLI SND包时，需要导入的类
from aoc.snd.netconfClisnd import netconfClisnd

# 可选，导入NetconfDriverInfo类
from aoc.snd.snd_model_pb2.ecsdriver_pb2 import NetconfDriverInfo
```

说明

在SND包中，用户自定义的包名+类名需保证全局唯一。即不同的SND包中，包名+类名也不能相同，否则会在包激活过程会失败。

5.2.3.2 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。这部分代码为公共代码，用户可参考代码中的注释进行定制开发。

注册设备 sysoid 信息

用户通过该接口设置设备的sysoid信息，用于纳管此款型设备。纳管时期望对设备的sysoid信息校验，则必须准确填写sysoid字段。若添加设备时不带sysoid字段，则通过pkg.json文件里的三元组来匹配。

本示例NE8000 M8软件包注册设备信息代码如下。

```
# 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 8000 M8"

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


版权所有 © 华为技术有限公司

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。

本示例NE8000 M8软件包只需要配置主协议设备连接能力信息，代码如下。

```
# 配置NETCONF协议类型及连接能力信息，这里为读写共享单通道场景
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.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

    # NETCONF&CLI双协议SND，还需要配置辅助协议能力信息，辅助协议配置请参考主协议代码注释
    """
      auxiliaryConnectInfo = connectInfos.connectInfo.add()
    ...
    ...
    """
    self.logger.info('getConnectInfo end.')
    return connectInfos
```

5.2.3.3 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。

5.2.3.3.1 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。NE8000 M8软件包通用能力交互策略代码如下。

  ```
  # 配置设备通用驱动：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"；syncToDel.value设置为true，设置数据对账时支持删除南向设备已配置数据
  ```


版权所有 © 华为技术有限公司

```
def getCommonDriverInfo(self, aoccontext, request=None):
    common_driver = CommonDriverInfo()

    # 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"
    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**

  NETCONF SND包和NETCONF&CLI SND包，支持NETCONF协议，需要定义NETCONF协议参数。

  NE8000 M8软件包协议类型定制代码如下。

```
# 配置设备NETCONF驱动：设置设备支持二阶段下发；设置设备原型，大部分华为设备是huawei-v5；设置下发设备的报文是无序下发
def getNetconfDriverInfo(self, aoccontext, request=None):
    netconf_driver = NetconfDriverInfo()

    # 设置neconf下发阶段，一阶段还是二阶段。
    netconf_driver.phase = "two"

    # 定义驱动设备原型类别
    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 = "same"
    """

    # 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包和NETCONF&CLI SND包，支持CLI协议，必须要定义CLI协议参数。

  NE8000 M8软件包不涉及，其他软件包请参考下文进行定制。

```
    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.xml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.xml")
            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
```

通过 xml 文件配置设备驱动信息

- NETCONF SND：用户通过“resources>netconf-driver.xml”文件，可以定制设备驱动信息。
- CLI SND：用户通过“resources>cli-driver.xml”文件，可以定制设备驱动信息。
- RESTCONF SND：用户通过“resources>restconf-driver.xml”文件，可以定制设备驱动信息。
- NETCONF&CLI SND：用户通过配置“resources>netconf-driver.xml”文件，可以定制设备驱动信息；还可以通过“resources>CliPassthroughCommands.xml”文件，配置CLI命令透传白名单，只有配置在白名单中CLI命令才可以通过北向接口透传命令行至设备，提高了业务安全性。

NE8000 M8软件包不提供通过xml文件配置设备驱动信息代码示例。

如果用户对通过xml文件配置设备驱动信息感兴趣，请参考下面“resources>netconf-driver.xml”文件通用配置示例。具体字段含义可以参考API参考文档。

```
<?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 -->
  <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>
```

 版权所有 © 华为技术有限公司

```
  <!-- 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 -->
      <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>
```

 版权所有 © 华为技术有限公司

```
            <!-- 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>
```

5.2.3.3.2 设置业务自定义处理

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，用户可以使用该接口自定义设备报文转换处理。

NE8000 M8软件包不涉及业务自定义处理，用户不关注时可直接跳过。

- NETCONF SND和NETCONF&CLI SND包，请参考如下代码示例设置业务自定义处理。

```
def netconfTransformer(self, aoccontext, input=None):
    self.logger.info('netconfTransformer start.')
    netconf_msg = NetconfMsg()
    netconf_msg.msg = input.data
    self.logger.info('netconfTransformer end.')
    return netconf_msg
```

- CLI SND包，自定义代码实现CLI转YANG和YANG转CLI的逻辑，请参考如下代码示例设置业务自定义处理。

```
    def yangToCli(self, aoccontext, request):
        self.logger.info('yangToCli start.')
        out = CliCustomTransformOutput()
        path1 = '/huawei-ifm:interfaces/huawei-ifm:interface/(.+)/huawei-ifm:isis/huawei-ifm:authentication-
mode/huawei-ifm:md5'
        raw_url = urllib.parse.unquote_plus(request.path)
        path1_res = re.search(path1, raw_url)
        if request.oper_type == OperType.Value('DELETE') and path1_res:
            list_key = path1_res.group(1)
            out.data = 'interface ' + list_key + '\nundo isis authentication-mode'
            return out

        self.logger.info('yangToCli end.')
        out.data = ''
        return out


    def cliToYang(self, aoccontext, request):
        self.logger.info('cliToYang start.')
        self.logger.info('yangToCli current path : %s' % request.path)
        if request.path == '/huawei-bfd:bfd-conf':
```


版权所有 © 华为技术有限公司

```
        return CliToYangCustom.clitoyang_bfd_conf(request, self.logger)
    if request.path == '/huawei-bgp:bgp-conf':
        return CliToYangCustom.clitoyang_bgp_conf(request, self.logger)
    self.logger.info('cliToYang end.')
```

5.2.3.3.3 设置数据前置处理

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。

NE8000 M8软件包不涉及数据前置处理，用户不关注时可直接跳过。

- NETCONF SND和NETCONF&CLI SND包，请参考如下代码示例设置设置数据前置处理。

```
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
```

5.2.3.3.4 设置数据后置处理

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。

NE8000 M8软件包不涉及数据后置处理，用户不关注时可直接跳过。

- NETCONF SND和NETCONF&CLI SND包，请参考如下代码示例设置设置数据后置处理。

```
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
```

5.2.3.3.5 设置数据联动

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。

NE8000 M8软件包不涉及数据联动处理，用户不关注时可直接跳过。

各种包类型设置数据联动方法是通用的，请参考如下代码示例。

```
def dbPostProcessBatch(self, aoccontext, request)
    self.logger.info('dbPostProcess start.')
    ecsDbConfigOut = EcsDbConfigOut()
    for ecsItem in request.data:
        ecsData = ecsItem.data
        ecsPath = ecsItem.path
        ecsopType = ecsItem.opType
        self.logger.info("ecsDbconfig  getData ddw:" + ecsData)
        self.logger.info("ecsDbconfig  getPath ddw:" + ecsPath)
```


版权所有 © 华为技术有限公司

```
            self.logger.info("ecsDbconfig  getOpType ddw:" + ecsopType)
            #  /huawei-ifm:interfaces/huawei-ifm:interface/asd/huawei-ifm:mpls/huawei-ifm:l2vc/primary

            # /huawei-ifm:interfaces/huawei-ifm:interface/Giga0000001/huawei-ifm:mpls/huawei-ifm:l2vc/
secondary
            if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
                self.logger.info("ecsDbconfig  getOpType primary ddw:" + ecsopType)
                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":
                self.logger.info("ecsDbconfig  getOpType secondary ddw:" + ecsopType)
                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"

        self.logger.info('dbPostProcess end.')
        return ecsDbConfigOut
```

5.2.3.3.6 获取 syncId

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。

NE8000 M8软件包不涉及获取syncId，用户不关注时可直接跳过。

请参考如下代码示例设置获取syncId。

```
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
```

5.2.3.4 编写设备特性能力定制

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。

本示例NE8000 M8软件包没有对设备特性能力进行定制，默认全量差异发现，用户不关注时可直接跳过。

用户需要进行设备特性能力定制时，各种包类型定制设备特性能力方法是通用的，请参考如下通用代码示例。

```
    def getFeatures(self, aoccontext, request=None):
        self.logger.info('getFeatures start.')
        feature_msg = FeatureCfgsMsg()
        feature_msg.replace = True
```

 版权所有 © 华为技术有限公司

```
        # 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”，则调用复写的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”，则调用复写的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”，则调用复写的postSyncFromNe过程，设置为“False”，则不定制
        function.postSyncFromNe = False
        feature.functions.extend([function])
        return [feature]


    # 复写对账前定制过程
    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
```

 版权所有 © 华为技术有限公司

```
        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。

5.2.3.5 管理资源

提供软件包的驱动配置文件（“cli-driver.properties”）及其它资源的管理。

针对CLI SND，需要根据目标命令行的特点对“resources/cli-driver.properties”文件进行定制，以便适配目标命令行。

```
#
# 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=Login:
passwordResponse=Enter password:
initCommand=display clock
keepAliveCommand=dis clock utc
cliTxCreateCommand=return\nsystem-view\n
# 针对支持二阶段的华为设备，可按如下样例设置“cliTxCommitCommand”、“cliTxCancelCommand”及“configModeCommand”三个参数。对于支持二阶段的三方设备，须按设备的实现完成设置。
cliTxCommitCommand=commit \n
cliTxCancelCommand=clear configuration candidate\n
configModeCommand=\n
cliTxSaveCommand=return \n
abortCommand=return
cliTxCancelRetryTimes=3
connectionTimeout=20000
keepAliveInterval=60000
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 ${context}
operModeCommand=display
partialReadCommand=return\nsystem-view\n${context}\ndisplay this
indentationSequence=\u0020
exitSubmodeCommand=quit
```


版权所有 © 华为技术有限公司

```
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| BGP process will be deleted. Continue|
BGP peer will be deleted. Continue| This operation will reset the peer session. Continue?| The destination
address and the mask do not match).*;^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*
cliInteractionGoOnCommond=Y;N
judgeContinueCondition=.*--- More ---*
partialReadSubmode=false
continueReadingCommand=\u0020
changePasswordPrompt=^The password needs to be changed\\. Change now\\? \\[Y\\/N\\].*
skipChangePasswordCmd=N
```

5.3 验证原子驱动包

5.3.1 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

5.4 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。

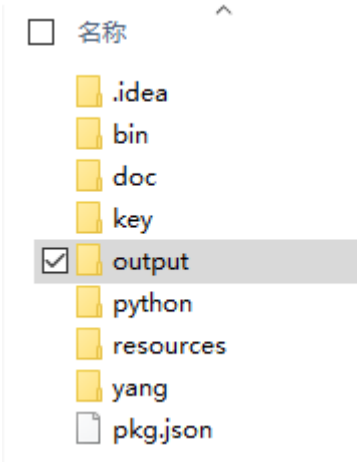
在命令行中运行如下命令：

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

 版权所有 © 华为技术有限公司

```
(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
[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）。

 版权所有 © 华为技术有限公司

6 开发网络业务插件包

业务插件包可以实现三种功能：EasyMap、RPC、Discover。EasyMap指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。

本章节以一个简单示例，介绍如何通过下载的默认SSP包，开发一个EasyMap业务插件包，实现网络层业务向网元层业务的正向分解。示例业务：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。

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。

6.1 创建业务包模板

6.2 开发业务包

6.3 验证业务包

6.4 编译业务包

6.1 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“增加”对话框中设置软件包的相关属性。

请根据**表6-1**填写软件包的相关属性值。

表 6-1 L3VPN 软件包属性

属性名称	属性值
名称	L3VPN
版本	1.0.0

 版权所有 © 华为技术有限公司

属性名称	属性值
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“OK”，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

6.2 开发业务包

6.2.1 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 6-2 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

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。	三种枚举值：Specific Service plugin、Generic NE driver、Specific NE driver。	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，目前对于 snd 包和 gnd 包，只支持该字段为"huawei"的包。	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，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
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”。	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，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。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
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
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。 key（必选），回调key值，不能超过128个字符。 java-class-name（可选），java开发语言特有值，回调映射类。不能超过 256个字符。 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。 groovy-class-name（可选），groovy开发语言特有值，回调映射类。不能超过 256个字符。 template（可选），模板名。不能超过256个字符。	可选

检查文件内容，EasyMap网络层业务分解为网元层业务时，type设置为mapping。

示例如下：

```
{
    "name": "L3VPN",
    "version": "1.0.0",
    "description": "l3vpn",
    "package-type": "ssp",
    "producer": "huawei",
    "service-name":"l3vpn",
    "nce-min-versions":[
        "1.0.0"
    ],

    "hooks": [
      {
        "type": "mapping",
        "key": "l3vpn",
        "python-class-name": "l3vpn.l3vpn.L3VPNService"
      }
    ]
}
```

----结束

 版权所有 © 华为技术有限公司

6.2.2 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。默认YANG文件名称对应pkg.json中"service-name"值，YANG模型中module名称对应pkg.json中"service-name"值。YANG模型中的“app:application-definition”的值对应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。

开发YANG模型时，分析梳理出配置下发涉及的参数，如表6-3所示，并在YANG模型中进行定义。

表 6-3 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
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	20.1.2.9
		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	2.2.2.2

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"{
```


版权所有 © 华为技术有限公司

```
    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。*/
      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.";
          }
        }
        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。

 版权所有 © 华为技术有限公司

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 6-4 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>
```

 版权所有 © 华为技术有限公司

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 "
      },
      {
```

 版权所有 © 华为技术有限公司

```
            "uri-pattern":"listA",
            "method":" create/delete/read/update "
        },
        {
            "uri-pattern":" listA/listC",
            "method":" create/delete/read/update"
        },
        {
            "uri-pattern":"resetStatictic",
            "method":"exec"
        }
        ]
    }
    ]
}
```

6.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码，实现EasyMap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。

实现EasyMap功能，需要继承NcsService并复写ncs_map方法，实现业务逻辑。

说明

在SSP包中，用户自定义的包名+类名需保证全局唯一。即不同的SSP包中，包名+类名也不能相同，否则会在包激活过程会失败。

L3VPN软件包代码示例如下所示。

```
# 必选，NcsService是系统提供的供SSP包继承的父类，需要复写以实现业务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
# datastore提供了读写数据库的接口
from aoc import datastore
# AOCException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 AOCException

# L3VPNService继承NcsService类并复写ncs_map方法
class L3VPNService(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):
        l3vpn_dict = self.xmltodict(request.xml)

        # 检查主接口是否存在
        self.checkInterfaceIsExist(request, aoccontext)

        # 更新VPN描述信息
        self.updateDes(l3vpn_dict)
        self.logger.info(str(l3vpn_dict))

        # 将业务YANG定义的业务数据填充到jinja2模板
       res = self.render('l3vpn/servicepoint.j2', l3vpn_dict)
        self.logger.info(res)
        return res


    # 更新vrf描述信息
    # 如果要获取YANG模型相关的节点数据，可以通过x.y.z或x['y']['z']这样的方式便捷操作模型数据。
    def updateDes(self, l3vpn_dict):
```


版权所有 © 华为技术有限公司

```
    if '3G' in l3vpn_dict['l3vpn']['vrfName'] :
        l3vpn_dict.update({"vrfDes": '3G'})
        l3vpn_dict['l3vpn'].update({"description": '3G'})


# 校验主接口是否存在
def checkInterfaceIsExist(self, request, aoccontext):
    device_id = devicemgr.query_neid(request.xmldictnode.l3vpn.deviceName)
    subIfName = request.xmldictnode.l3vpn.ifName
    pointIndex = subIfName.find('.')
    parentName = subIfName[0:pointIndex]
    self.logger.info('deviceId:%s, parentName:%s' % (device_id, parentName))
    if devicemgr.is_snd(device_id, 'NE8000M8SPC300_SND') or devicemgr.is_snd(device_id, 'NE8000_M8'):
        path = "huawei-ac-nes:inventory-cfg/nes/ne/" + device_id + "/huawei-ifm:ifm/interfaces/"

        # 查询配置数据库中数据
        output = datastore.read_datastore_rdb(aoccontext, path)
        asnindex = str(output).find(parentName)
        if asnindex == -1:
            raise AOCException(parentName + ' not exist in device!')


# 添加Jinja2模板过滤器，用于在Jinja模板中实现SSP业务映射到不同款型设备
def add_filters(self):
    result = {"is_snd": self.is_snd}
    return result


def is_snd(self, nename, sndid):
    # 根据设备名称查询设备ID
    neid = devicemgr.query_neid(nename)
    # 判断设备使用的哪个网元驱动包
    return devicemgr.is_snd(neid, sndid)
```

6.2.4 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。

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“设备配置 > 设备配置”。进入“设备配置”界面，选择对应的设备，单击“同步操作 > 同步”。

步骤2 选择对应的设备，单击“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”，在操作列单击“查看差异”，打开差异呈现页面，选择“导出模板”。

 版权所有 © 华为技术有限公司

步骤4 在“设备配置”界面，单击对应设备操作列中的“编辑”。

步骤5 在左边模型节点上方单击导出图标，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>
                        <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>
```

 版权所有 © 华为技术有限公司

```
      </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>
```

----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。

 版权所有 © 华为技术有限公司

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节点都只会被删除数据源，不会被删除数据。

  示例如下：

```
<system  xmlns:ns0="urn:huawei:ac" ns0:tag="no_delete_nested">
  <inform-timeout>2</inform-timeout>
</system>
<ntpAuthKeyCfg xmlns:x="urn:huawei:ac" x:tag="no_delete">
```

----结束

6.3 验证业务包

6.3.1 验证 YANG 文件

在完成软件包的开发后，可以使用YANG模型校验工具校验软件包中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-offline-util.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

6.3.2 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>
```

 版权所有 © 华为技术有限公司

```
    <description>wireless base</description>
    <ip>20.1.2.9</ip>
    <subnet>255.255.255.0</subnet>
    <peerAddress>2.2.2.2</peerAddress>
  </l3vpn>
```

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>20.1.2.9</ip>
    <subnet>255.255.255.0</subnet>
    <peerAddress>2.2.2.2</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。

 版权所有 © 华为技术有限公司

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

6.4 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

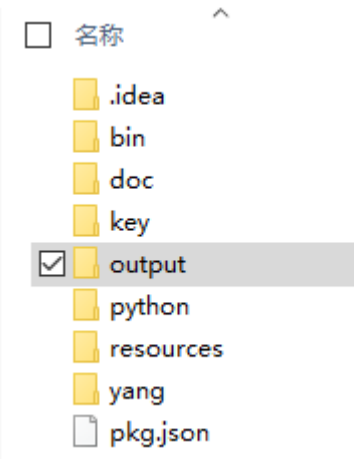
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

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。




```
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

版权所有 © 华为技术有限公司

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

7 开发业务还原业务插件包

业务插件包可以实现三种功能：EasyMap、RPC、Discover。Discover即业务还原，将网元层业务重新组织成网络层业务，即EasyMap的逆过程。如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。

本章节通过一个示例介绍如何通过下载的默认SSP包，开发一个业务还原业务插件包，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。

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 创建业务包模板

7.2 开发业务包

7.3 验证业务包

7.4 编译业务包

7.1 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“增加”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。

表 7-1 软件包属性

属性名称	属性值
名称	SSP-SRTE

 版权所有 © 华为技术有限公司

属性名称	属性值
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
业务定义点	neTunnels

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“OK”，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 开发业务包

7.2.1 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
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

版权所有 © 华为技术有限公司

参数名称	含义	取值	必选/可选
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。	三种枚举值：Specific Service plugin、Generic NE driver、Specific NE driver。	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，目前对于 snd 包和 gnd 包，只支持该字段为"huawei"的包。	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，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
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”。	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，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。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
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
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。 key（必选），回调key值，不能超过128个字符。 java-class-name（可选），java开发语言特有值，回调映射类。不能超过 256个字符。 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。 groovy-class-name（可选），groovy开发语言特有值，回调映射类。不能超过 256个字符。 template（可选），模板名。不能超过256个字符。	可选

示例如下：

```
{
  "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",
```

 版权所有 © 华为技术有限公司

```
    "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

7.2.2 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。默认YANG文件名称对应pkg.json中"service-name"值，YANG模型中module名称对应pkg.json中"service-name"值。YANG模型中的“app:application-definition”的值对应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。

正向的网络业务下发和逆向的业务还原使用的YANG模型是一样的，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7-3所示，并在YANG模型中进行定义。

表 7-3 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
		bandwidth	ct0 1
		setup-priority	0
		pce-delegate	active
		hotstandby	hotstandby

版权所有 © 华为技术有限公司

类别	描述	参数名称	参数值
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
igp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cmnetSrte软件包YANG模型示例如下，节选SRTE隧道配置部分。

```
module cmnetSrte {
    ......
   grouping ne-tuls {
      list neTunnel {
      app:application-definition "neTunnels";
         key "deviceId tunnelName";
         uses srte-tunnel;
      }
   }

   augment "/app:applications" {
      uses ne-tuls;
}
```

 版权所有 © 华为技术有限公司

```
    // 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;
      }
...
```

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 7-4 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

 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

7.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码，实现Discover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。

实现Discover功能，需要继承NcsService并复写Discover方法，实现业务的逻辑。

```
# 必选，导入NcsService类，NcsService是aoc_api提供的供SSP包继承的父类。
from ipaddress import IPv4Network
from aoc import NcsService, devicemgr
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))

        # 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()))

        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
```


版权所有 © 华为技术有限公司

```
            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


    # 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']]
```

 版权所有 © 华为技术有限公司

```
            # 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
```

7.2.4 开发 Jinja2 模板

开发Jinja2模板的思路：导出数据模板法：在业务管理界面导出需要的YANG模板，在YANG模板中将变量作为参数填充进去，变量可以参考正向配置下发的Jinja2模板。

步骤1 在主菜单中选择“业务管理”。进入“业务管理”界面。

步骤2 单击 ，勾选“NeTunnel”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3 打开导出的模板，可以看到文件中包含数据结构，增加逻辑处理，将需还原的配置数据作为变量填充进去。最后把编辑的内容放入到模板文件“template>cmnetSrte.j2”中。节选代码示例如下。

```
{# 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'] %}
   <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 %}
```


版权所有 © 华为技术有限公司

```
  {%- endif %}
  {%- endif %}
...
```

----结束

7.3 验证业务包

7.3.1 验证 YANG 文件

在完成软件包的开发后，可以使用YANG模型校验工具校验软件包中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-offline-util.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

7.3.2 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。需要注意的是，这里的YANG模型要使用设备的YANG模型，而不是业务YANG模型。仅使用涉及到设备YANG模型即可，ifm和network_instance。

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。

```
<interface>
    <mtu>500</mtu>
    <name>Tunnel1</name>
    <description>1234567890</description>
    <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
      <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
        <sr-te>
          <hotstandby>
            <wtr-time>10</wtr-time>
            <path-overlap>false</path-overlap>
            <revertive-mode>revertive</revertive-mode>
          </hotstandby>
          <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
            <detect-multiplier>3</detect-multiplier>
            <min-rx-interval>3</min-rx-interval>
            <ability>enable</ability>
            <min-tx-interval>3</min-tx-interval>
          </bfd-for-lsp>
          <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
```

 版权所有 © 华为技术有限公司

```
          <detect-multiplier>3</detect-multiplier>
          <min-rx-interval>3</min-rx-interval>
          <mode>seamless</mode>
          <ability>enable</ability>
          <min-tx-interval>3</min-tx-interval>
        </bfd-for-tunnel>
        <bandwidth>1</bandwidth>
        <hold-priority>0</hold-priority>
        <pce-delegate>active</pce-delegate>
        <lsp-paths>
          <lsp-path>
            <explicit-path-name>expath_0110</explicit-path-name>
            <type>primary</type>
            <include-any>0x0</include-any>
            <exclude-any>0x0</exclude-any>
            <hop-limit>20</hop-limit>
          </lsp-path>
          <lsp-path>
            <explicit-path-name>expath_0111</explicit-path-name>
            <type>hot-standby</type>
            <include-any>0x0</include-any>
            <exclude-any>0x0</exclude-any>
            <hop-limit>32</hop-limit>
          </lsp-path>
        </lsp-paths>
        <setup-priority>0</setup-priority>
      </sr-te>
      <igp-attr xmlns="urn:huawei:yang:huawei-igp">
        <hold-time>0</hold-time>
        <shortcut-type>ospf</shortcut-type>
        <absolute>1</absolute>
        <adv-enable>false</adv-enable>
      </igp-attr>
      <common-attributes>
        <lsp-tp-outbound>true</lsp-tp-outbound>
        <signal-protocol>segment-routing</signal-protocol>
        <reserved-for-binding>true</reserved-for-binding>
        <egress-lsr-id>10.10.110.114</egress-lsr-id>
        <tunnel-id>1</tunnel-id>
        <statistic-enable>true</statistic-enable>
      </common-attributes>
    </te-tunnel>
  </tunnel-protocol>
</interface>
<interface>
  <mtu>500</mtu>
  <name>Tunnel2</name>
  <description>1234567890</description>
  <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
    <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
      <sr-te>
        <hotstandby>
          <wtr-time>10</wtr-time>
          <path-overlap>false</path-overlap>
          <revertive-mode>revertive</revertive-mode>
        </hotstandby>
        <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
          <detect-multiplier>3</detect-multiplier>
          <min-rx-interval>3</min-rx-interval>
          <ability>enable</ability>
          <min-tx-interval>3</min-tx-interval>
        </bfd-for-lsp>
        <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
          <detect-multiplier>3</detect-multiplier>
          <min-rx-interval>3</min-rx-interval>
          <mode>seamless</mode>
          <ability>enable</ability>
          <min-tx-interval>3</min-tx-interval>
        </bfd-for-tunnel>
```

 版权所有 © 华为技术有限公司

```
        <bandwidth>1</bandwidth>
        <hold-priority>0</hold-priority>
        <pce-delegate>active</pce-delegate>
        <lsp-paths>
          <lsp-path>
            <explicit-path-name>expath_0110</explicit-path-name>
            <type>primary</type>
            <include-any>0x0</include-any>
            <exclude-any>0x0</exclude-any>
            <hop-limit>20</hop-limit>
          </lsp-path>
          <lsp-path>
            <explicit-path-name>expath_0111</explicit-path-name>
            <type>hot-standby</type>
            <include-any>0x0</include-any>
            <exclude-any>0x0</exclude-any>
            <hop-limit>32</hop-limit>
          </lsp-path>
        </lsp-paths>
        <setup-priority>0</setup-priority>
      </sr-te>
      <igp-attr xmlns="urn:huawei:yang:huawei-igp">
        <hold-time>0</hold-time>
        <shortcut-type>ospf</shortcut-type>
        <absolute>1</absolute>
        <adv-enable>false</adv-enable>
      </igp-attr>
      <common-attributes>
        <lsp-tp-outbound>true</lsp-tp-outbound>
        <signal-protocol>segment-routing</signal-protocol>
        <reserved-for-binding>true</reserved-for-binding>
        <egress-lsr-id>10.10.110.114</egress-lsr-id>
        <tunnel-id>1</tunnel-id>
        <statistic-enable>true</statistic-enable>
      </common-attributes>
    </te-tunnel>
  </tunnel-protocol>
</interface>'''
```

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，并补充模拟调用代码。示例如下。

```
import unittest
import sys
from aoc.base.aoccontext import AocContext
from aoc.sys import devicemgr, datastore
from mock import Mock
sys.path.insert(0, "../../python")
from com.huawei.cmnet.srte.srte_restore import SrteRestoreService
from aoc.sys.sys_model_pb2.querydevicesoutput_pb2 import QueryDevicesPagedResult
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverInput


deviceId = "1111222222333344"
class Test(unittest.TestCase):

  # 本地测试无法调用真实环境数据，通过模拟方法调用模拟数据
  def mock_sys_func(self):
      devicemgr.query_basic_data_from_db = self.mock_query_basic_data_from_db
      datastore.read_datastore_rdb = self.mock_read_datastore_rdb

  # 模拟查询到的设备信息
  def mock_query_basic_data_from_db(self,neid='', nename='', nemanageip=''):
      response_body = QueryDevicesPagedResult()
      add_entity = response_body.queryDevicesPagedEntity.add()
      add_entity.neId = "neId"
      add_entity.operateName = "operateName"
      add_entity.hardWare = "NE40E"
      add_entity.softWare = "V800R21C00"
      add_entity.manageIp = "10.10.10.10"
```

 版权所有 © 华为技术有限公司

```
    add_entity.type = "NE40E"
    add_entity.manufacturer = "HUAWEI"
    return response_body


def setUp(self):
    self.mock_sys_func()


# 模拟调用业务还原接口，触发业务还原
def test_restore_srte(self):
    discoverinput = DiscoverInput()
    deviceInfo = discoverinput.deviceInfo.add()
    deviceInfo.deviceId = deviceId
    aoc_context = AocContext()
    aoc_context.transactionId = '112123'
    result = SrteRestoreService(None, '../../').discover(discoverinput, aoc_context)
    print(result)
    for sercfg in result.serviceConfig:
        print(sercfg.serviceData)

# 模拟从设备上返回的报文信息，报文较长，这里部分示例。
def mock_read_datastore_rdb(self, aoccontext, path, query_para=None):
    ifm_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + \
               '/huawei-ifm:ifm/interfaces/interface'
    network_instance_path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + \
                            '/huawei-network-instance:network-instance/' \
                            'instances/instance/_public_'
    valid_path = '/huawei-ac-applications:applications/cmcc-ip-srte:neTunnel/'

  # 下面<interface> 等标签代码根据subtree.xml及实际情况替换
    ifm_result_context = '''
<interface>
  <mtu>500</mtu>
  <name>Tunnel1</name>
  <description>1234567890</description>
  <tunnel-protocol xmlns="urn:huawei:yang:huawei-tunnel-management">
    <te-tunnel xmlns="urn:huawei:yang:huawei-mpls-te">
      <sr-te>
        <hotstandby>
          <wtr-time>10</wtr-time>
          <path-overlap>false</path-overlap>
          <revertive-mode>revertive</revertive-mode>
        </hotstandby>
        <bfd-for-lsp xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
          <detect-multiplier>3</detect-multiplier>
          <min-rx-interval>3</min-rx-interval>
          <ability>enable</ability>
          <min-tx-interval>3</min-tx-interval>
        </bfd-for-lsp>
        <bfd-for-tunnel xmlns="urn:huawei:yang:huawei-mpls-te-bfd">
          <detect-multiplier>3</detect-multiplier>
          <min-rx-interval>3</min-rx-interval>
          <mode>seamless</mode>
          <ability>enable</ability>
          <min-tx-interval>3</min-tx-interval>
        </bfd-for-tunnel>
        <bandwidth>1</bandwidth>
        <hold-priority>0</hold-priority>
        <pce-delegate>active</pce-delegate>

...


    if path == ifm_path:
        if query_para:
            return ifm_result_with_para
        return ifm_result
    if path == network_instance_path:
```

 版权所有 © 华为技术有限公司

```
            return network_instance_result
        if path == valid_path:
            return valid_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

7.4 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

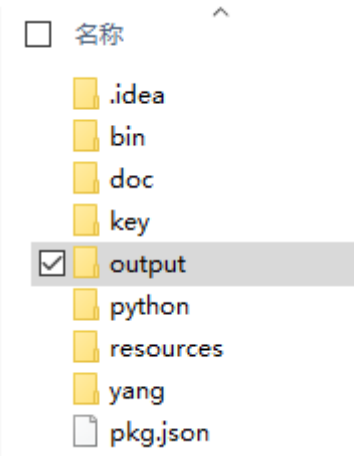
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

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。




```
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
```


版权所有 © 华为技术有限公司

```
[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）。

 版权所有 © 华为技术有限公司

8 开发 RPC 业务插件包

业务插件包可以实现三种功能：EasyMap、RPC、Discover。RPC指用户自定义功能，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。

本章节以一个简单示例，介绍如何通过下载的默认SSP包，开发一个RPC业务插件包。本示例实现自定义查询设备信息，获取设备报文。

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。

8.1 创建业务包模板

8.2 开发业务包

8.3 验证业务包

8.4 编译业务包

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“增加”对话框中设置软件包的相关属性。

请根据**表8-1**填写软件包的相关属性值。

表 8-1 RPC 软件包属性

属性名称	属性值
名称	RPC
版本	1.0.0

 版权所有 © 华为技术有限公司

属性名称	属性值
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“OK”，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。

表 8-2 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

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。	三种枚举值：Specific Service plugin、Generic NE driver、Specific NE driver。	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，目前对于 snd 包和 gnd 包，只支持该字段为"huawei"的包。	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，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
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”。	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，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。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
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
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。 key（必选），回调key值，不能超过128个字符。 java-class-name（可选），java开发语言特有值，回调映射类。不能超过 256个字符。 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。 groovy-class-name（可选），groovy开发语言特有值，回调映射类。不能超过 256个字符。 template（可选），模板名。不能超过256个字符。	可选

示例如下：

```
{
   "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"
     }
   ]
}
```

----结束

8.2.2 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。默认YANG文件名称对应pkg.json中"service-name"值，YANG模型中module名称对应

版权所有 © 华为技术有限公司

pkg.json中"service-name"值。YANG模型中的“app:application-definition”的值对应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;
            }
            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;
                    }
                }
```

 版权所有 © 华为技术有限公司

```
        }
      }
    }
}
```

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 8-3 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
-	-	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。

 版权所有 © 华为技术有限公司

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；

 版权所有 © 华为技术有限公司

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"
      }
      ]
   }
   ]
}
```

8.2.3 开发业务包代码

在集成开发环境左侧导航树中，双击打开Python目录下的.py文件，编写Python代码，实现RPC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并复写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。

```
# 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
```

 版权所有 © 华为技术有限公司

```
            </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))
    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
```

 版权所有 © 华为技术有限公司

```
    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。

8.2.4 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。

8.3 验证业务包

8.3.1 验证 YANG 文件

在完成软件包的开发后，可以使用YANG模型校验工具校验软件包中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-offline-util.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

8.3.2 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
```

 版权所有 © 华为技术有限公司

```
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

8.4 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

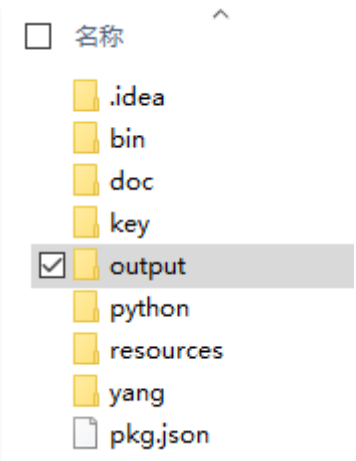
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

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。



 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）。

 版权所有 © 华为技术有限公司

9 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。

9.1 登录系统

9.2 导入并激活软件包

9.1 登录系统

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：

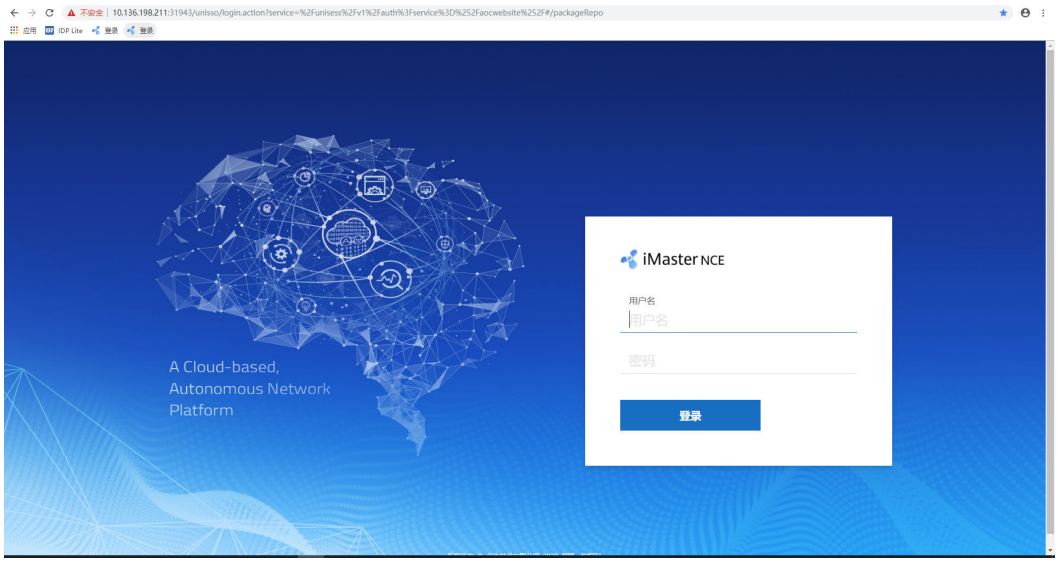
浏览器	浏览器版本
Microsoft Edge （Chromium Inside）	MicrosoftEdge_89.0.774.27_Beta
Chrome Stable Channel	Chrome 73.0.3683 开始
Firefox	• Firefox 65 开始 • Firefox 68 ESR 开始

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

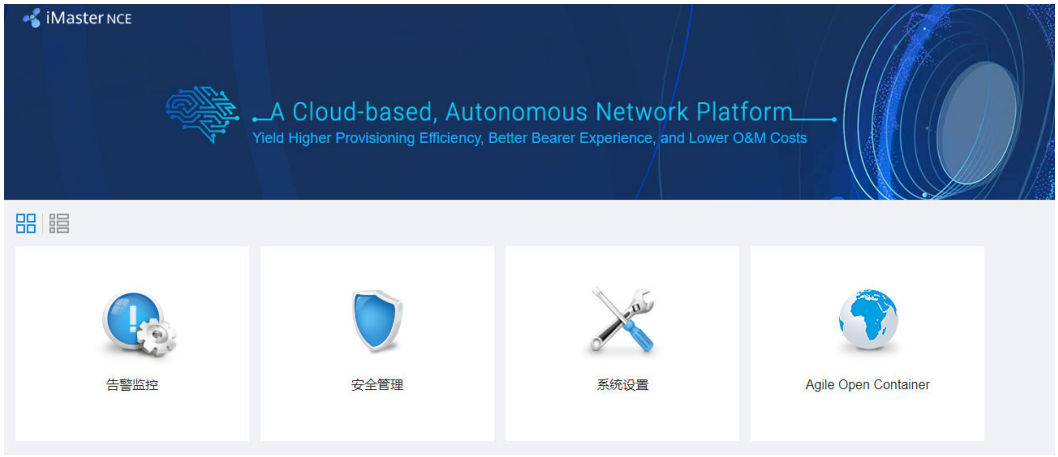
图 9-1 登录 NCE 界面



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

9.2 导入并激活软件包

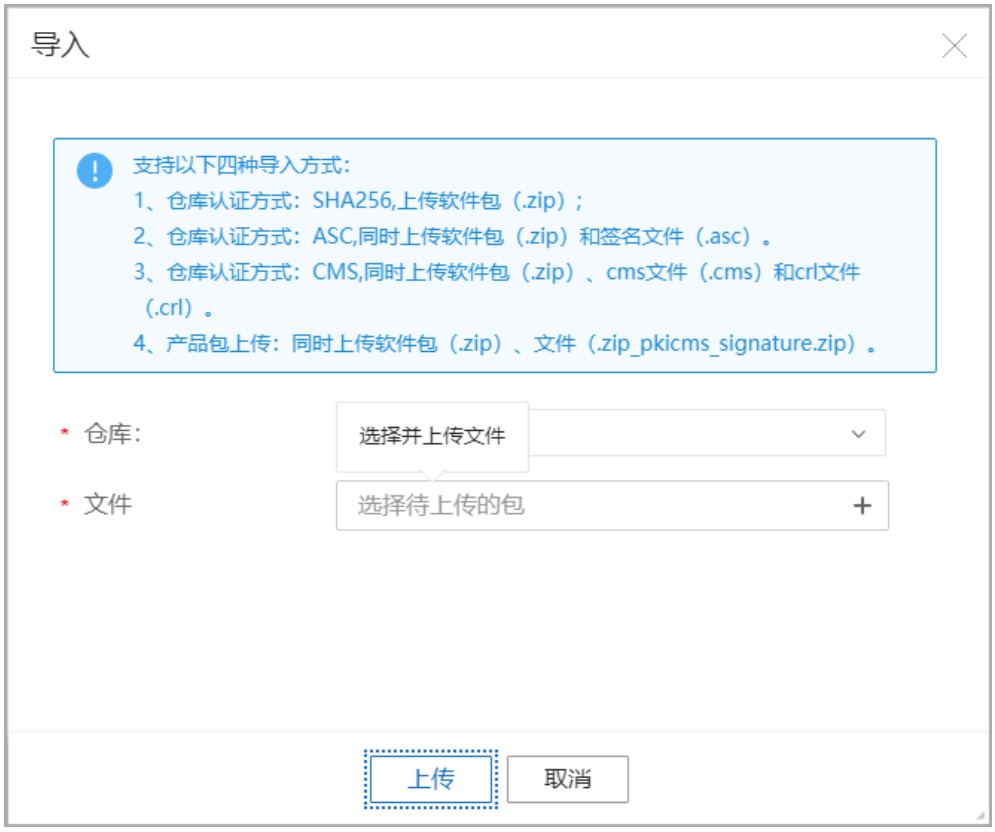
步骤1 在主菜单中选择“包仓库”，从左侧导航栏选择进入“包管理”界面，单击“导入”。

图 9-2 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

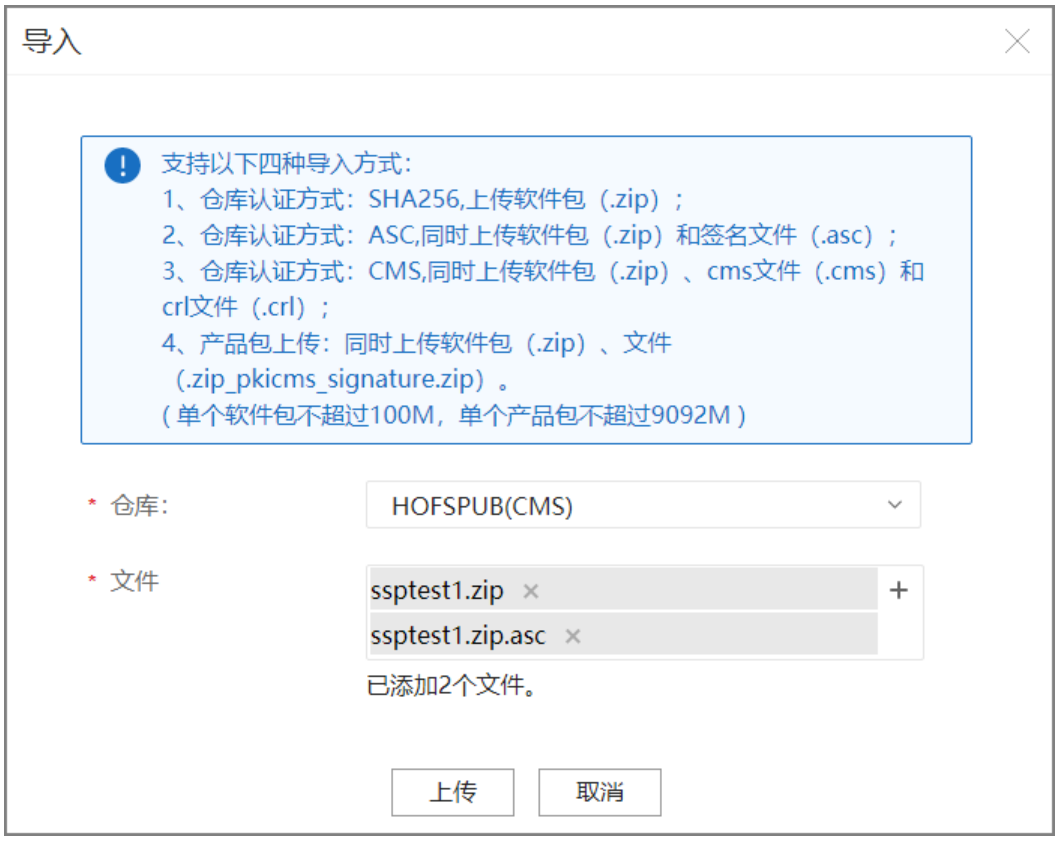
图 9-3 软件包导入选择对话框



步骤3 选择完成，单击“上传”。

 版权所有 © 华为技术有限公司

图 9-4 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。

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 9-5 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。

图 9-6 软件包安装成功

----结束

 版权所有 © 华为技术有限公司

10 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。

不兼容升级处理步骤

不兼容升级处理步骤：

步骤1 创建迁移包模板，并编辑包配置文件。

1. 迁移包可以通过创建默认模板的方法来生成，在“包管理”界面，单击“创建模板”。包类型选择“Snd Migrate Plugin”，提供商和设备类型请和要升级的SND保持一致。然后单击“导出”将创建的模板包下载到本地。迁移包的相关属性配置示例如下表所示。

   表 10-1 迁移包属性

属性名称	属性值
名称	NE8000M8_Migrate
版本	1.0.0
提供商	HUAWEI
包类型	Snd Migrate Plugin
映射类型	python
设备类型	NetEngine 8000 M8

2. 编辑pkg.json文件，"from-snd-id"设置为升级前SND包的snd-id，"to-snd-id"设置为升级后SND包的snd-id。

```
 "from-snd-id":"migrate_test_new_domain",
 "to-snd-id":"migrate_test_new_domain2",
```

步骤2 使用工具比较升级前后模型，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。

 版权所有 © 华为技术有限公司

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3 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4 继续识别所有不兼容点，根据**表10-2**判断属于哪种场景，并进行对应的处理。

- 如果是简单不兼容场景，可参见**表10-3**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。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-path>
        </migration-item>
    </migration-rule>
    <migration-rule>
        <rule>REMOVE_LEVEL</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances</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-rule>
    <migration-rule>
        <rule>SECURITY</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNode</source-path>
        </migration-item>
    </migration-rule>
    <migration-rule>
        <rule>KEY_CHANGE</rule>
        <migration-item>
            <source-path>/huawei-l3vpn:l3vpn/instances/instance</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。

 版权所有 © 华为技术有限公司

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>
```

表 10-2 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

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
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
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

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。

	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

表 10-3 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>
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>

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。
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>
COMPLEX	<migration-rule> <rule>COMPLEX</rule> <migration-item> <source-path>/huawei-l3vpn:l3vpn/instances/instance/instanceChildContainer/unSecurityNodeList1</source-path> </migration-item> </migration-rule>

步骤5 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，当前只支持json数据库。

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。

```
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.')
```

 版权所有 © 华为技术有限公司

```
        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
```

步骤6 编译迁移包，然后导入并激活迁移包。在设备升级时，自动根据迁移包中配置规则和回调脚本进行处理。

----结束

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。

 版权所有 © 华为技术有限公司

表 10-4 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; } }

	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; } }

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

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

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";
				}

当前支持的兼容/不兼容升级规格如下表所示。

 版权所有 © 华为技术有限公司

11 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