数通网络开放可编程
V100R020C00
插件包 API 参考

文档版本 02

发布日期 2020-12-30

华为技术有限公司

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NCE V100R020C00版本开放可编程系统插件包的相关API接口。

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

使用约束

做为NCE的子系统，NCE开放可编程具备基本的网络管理能力。对于通过NCE下发过业务配置（包括但不限于 MBGP L3VPN、VPLS、动/静态隧道等）的设备，不建议在NCE开放可编程中使用，否则可能会出现由于多头管理导致的业务数据不一致，进而产生数据冲突或业务安全风险。

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	用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害。
警告	用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害。
注意	用于警示潜在的危险情形，若不避免，可能会导致中度或轻微的人身伤害。

符号	说明
须知	用于传递设备或环境安全警示信息，若不避免，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。 “注意”不涉及人身伤害。
说明	用于突出重要/关键信息、最佳实践和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

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
01	2020-08-30	第一次正式发布。

目 录

1 编程接口概述

1.1 接口概览

本节介绍开放可编程框架提供的接口。

图 1-1 API 和 SPI

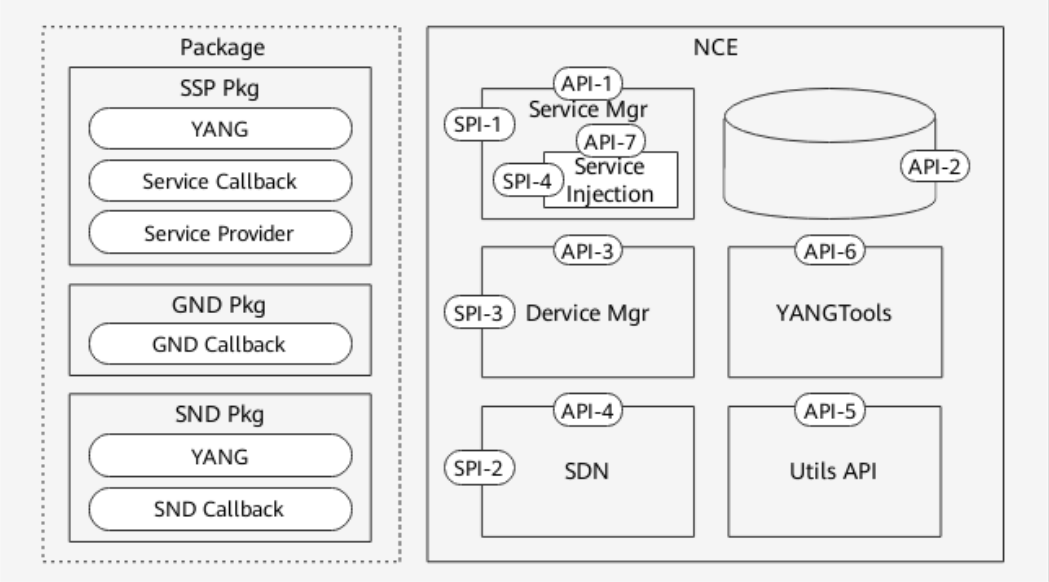


表 1-1

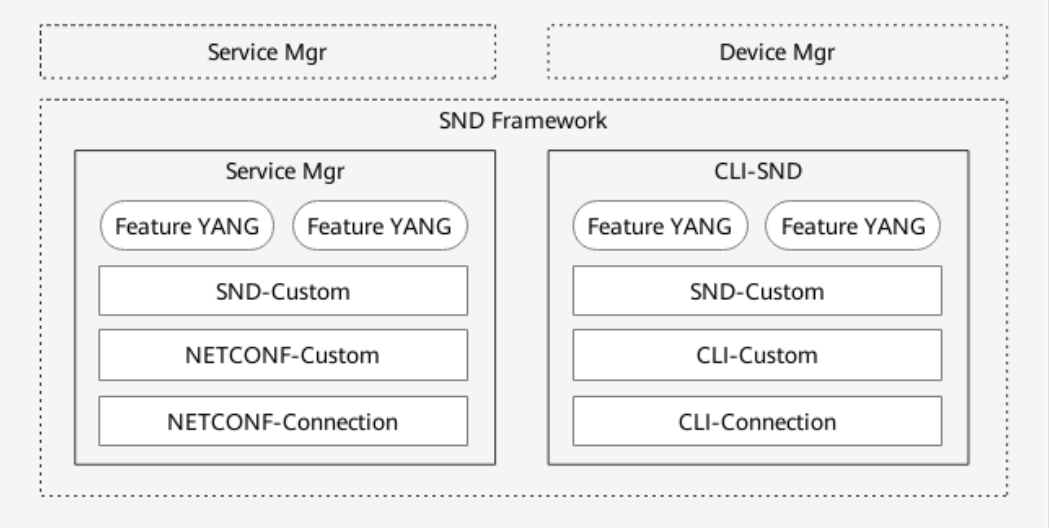
接口类型	接口子类	场景/能力
SPI	SPI-1：业务功能处理回调接口	提供了某类业务相关的功能实现，这些功能包括：业务映射、业务运维动作、业务还原等。

接口类型	接口子类	场景/能力
	SPI-2：SND功能处理回调接口	对厂商设备能力的抽象，提供了某类设备相关的功能实现，包括对设备功能的配置、运维动作、配置一致性、设备连接等。
	SPI-3：GND功能处理回调接口	对通用类设备能力抽象，提供了某类设备厂商模型数据到通用模型数据的映射，这些模型包括：接口、链路、告警、业务配置等。
	SPI-4：业务服务注入功能回调接口	为系统提供了特定工具功能，可以被系统其它SND、GND或业务功能调用。
API	API-1：配置事务接口	提供的配置事务接口，包括开启事务、编辑事务、回退事务、清除事务。
	API-2：数据查询接口	提供基于模型数据库的查询接口，包括CDB的查询接口、RDB的查询接口。
	API-3：设备基本信息查询接口	提供设备基本信息的查询接口，包括通过设备名查询设备ID、通过设备ID查询设备MAC地址等。
	API-4：设备数据查询接口和操作接口	提供基于设备YANG模型的查询接口、设备操作接口。
	API-5：模板类功能接口	提供模板的工具接口，比如IP地址的格式转换、mask地址的格式转换、模板的属性定制等。
	API-6：模型数据封装	提供操作模型的接口，封装成易用的数据获取接口。
	API-7：业务服务注入	提供注入服务接口。

1.2 接口调用原理

1.2.1 SND 接口调用原理

图 1-2 SND 接口调用原理



SND 框架调用SND包的代码，实现设备相关功能的定制，定制包括三种大类：

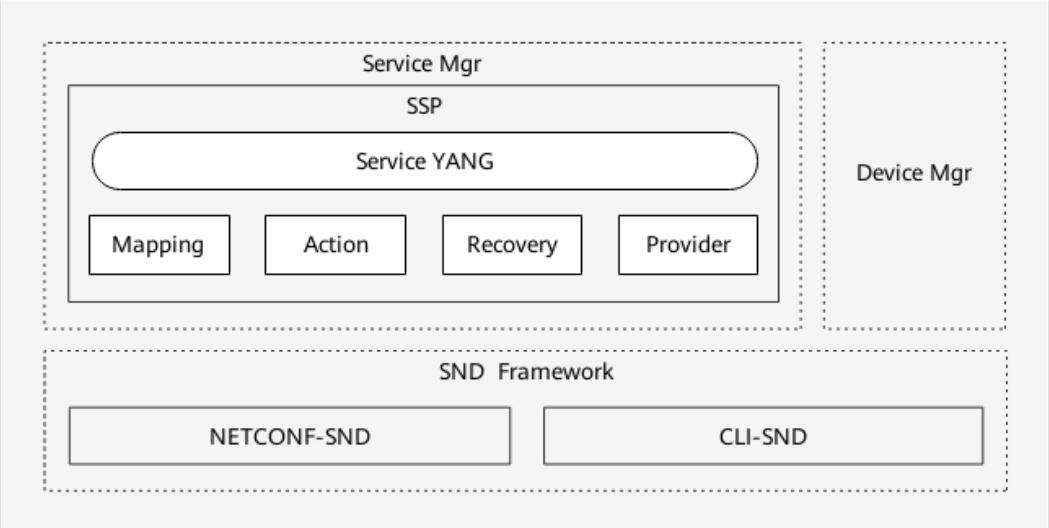
SND通用特殊处理定制：对于同步数据时特性采集范围定制，以及其他协议无关的定制。

协议特殊处理定制：如由于设备不符合协议标准进行的定制，或定制事务对接方式。

协议连接方式定制：如连接的通道数、保护方式。

1.2.2 SSP 接口调用原理

图 1-3 SSP 接口调用原理



业务管理调用SSP的代码，实现业务功能的定制，定制包括四种大类：

业务mapping定制：通过定制映射逻辑，实现了业务创建、删除、更新等业务处理能力。

业务action定制：通过定制行为逻辑，实现了业务运维能力。

业务recovery定制：通过定制还原逻辑，实现了业务从设备还原到业务的能力。

业务provider定制：通过定制服务提供逻辑，实现了对系统提供工具类的能力，并支持和系统事务的一致性。

2 API 参考

2.1 通用 API

本节介绍通用API，SSP包、GND包或者SND包在开发的过程中都可以调用。

2.1.1 设备操作 API

本节包含的接口都是实时与设备交互的接口。

2.1.1.1 查询设备配置

查询设备配置数据，接口入参不需要设置南向协议信息，返回值按设备YANG模型呈现设备配置数据。

类型

API-4，设备数据查询接口和操作接口。

典型场景

用户在编写SSP包、GND包或者SND包时，某些数据需要从设备上实时获取时调用。

接口功能

query_data_from_device接口实现了实时查询设备数据的功能，具体处理如下。

1. 根据入参neid指定设备的YANG模型以及入参path生成南向报文。
2. 下发南向报文并同步等待设备返回结果。

接口约束

1. 前提条件：使用该接口时必须满足设备在线。
2. 注意事项：设备超过30s未返回查询结果，接口查询失败。
3. 使用限制：构造path与设备yang模型保持一致
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def query_data_from_device(neid, storetype, path, properties={}, timeout=30000):
    """
    query_data_from_device is used to query device configuration.

    Input:
        string neid
        string storetype, value can be 'OPERATIONAL' or 'CONFIGURATION'.
        string path
        map<string, string> properties, default value is {}
        int timeout, default value is 30000 ms
    Output:
        string result
    """
```

请求参数描述

表 2-1 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neid	是	STRING	-	-	格式：uuid 设备的标识ID。
storetype	是	ENUM	-	-	OPERATIONAL：用于查询状态数据，对应NETCONF get报文； CONFIGURATION：用于查询配置数据，对应NETCONF get-config报文。
path	是	STRING	-	-	将要从设备上读取的数据对应的YANG子树路径。
properties	否	DICT	-	dict{}	附加字段，可支持子树过滤。
timeout	否	INT32	大于0	30000	超时时间，默认为30000ms。

请求和响应样例

Python调用样例：

```
# 引入aoc devicemgr
from aoc import devicemgr

# 入参 neid 及 path 都是Java侧传入的或者通过接口从Java侧查询到的
neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
path = '/huawei-ac-ne-snmp:snmp'

output = devicemgr.query_data_from_device(neid, 'CONFIGURATION', path)

# 得到的返回结果output类似于如下xml报文
<snmp xmlns=\"urn:huawei:yang:huawei-ac-ne-snmp\">
   <mibViews>
      <mibView>
         <viewName>118</viewName>
         <subtree>iso</subtree>
         <type>excluded</type>
      </mibView>
   </mibViews>
</snmp>
```

错误码

错误码	错误信息	处理措施
无	python invoke read fail!	三方包制作问题，排查pkg.json是否正确
无	input is null!	三方包制作问题，排查pkg.json是否正确

2.1.1.2 设备 RPC

设备RPC操作，这里RPC指的是设备上实现的基于YANG标准的RPC能力，NCE能够通过NETCONF协议下发RPC报文。

类型

API-4，设备数据查询接口和操作接口。

典型场景

对设备进行RPC操作。

接口功能

device_rpc接口实现了实时RPC的功能，具体处理如下。

1. 根据入参neid指定设备的YANG模型以及入参path生成南向报文。
2. 下发南向报文并阻塞等待返回结果。

接口约束

1. 前提条件：使用该接口时必须满足设备在线。
2. 注意事项：设备超过30s未返回查询结果，接口操作失败。
3. 使用限制：传入rpcData，rpcName与设备yang模型保持一致
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def device_rpc(neid, rpcdata, rpcName, timeout=30000):
    """
    device_rpc is used to config the device.
    Input:
        string neid
        string rpcdata
        string rpcName
        int timeout, default value is 30000 ms
    Output:
        string result
    """
```

请求参数描述

表 2-2 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neid	是	STRING	-	-	格式：uuid 设备的标识ID。
rpcdata	是	STRING	-	-	YANG模型定义的RPC对应xml格式数据。
rpcName	是	STRING	-	-	对应RPC的QName。
timeout	否	INT32	大于0	30000	超时时间，默认为30000ms。

请求和响应样例

Python调用样例：

```
"""
假设设备上定义了如下RPC：
YANG Example:
    rpc activate-software-image {
        input {
            leaf image-name {
                type string;
            }
```

```
        }
        output {
            leaf status {
                type string;
            }
        }
    }
"""

# 引入aoc devicemgr
neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
path = '(urn:huawei:yang:huawei-aoc-rpc-test?revision=2018-01-01)activate-software-image'
rpcName = '''
<activate-software-image xmlns="urn:huawei:yang:huawei-aoc-rpc-test">
    <image-name>acmefw-2.3</image-name>
</activate-software-image>
'''
result = devicemgr.device_rpc(neid, rpcdata, rpcName)

# 得到的返回结果output类似于如下xml报文
<activate-software-image xmlns="urn:huawei:yang:huawei-aoc-rpc-test">
   <status>The image acmefw-2.3 is being installed.</status>
</activate-software-image>
```

错误码

错误码	错误信息	处理措施
无	python invoke read fail!	三方包制作问题，排查pkg.json是否正确
无	input is null!	三方包制作问题，排查pkg.json是否正确

2.1.1.3 通过 CLI 查询设备配置或运行数据

业务在Python侧通过CLI查询设备配置或运行数据。

类型

API-4，设备数据查询接口和操作接口。

典型场景

通过CLI对设备进行查询操作。

接口功能

业务通过Python侧的sendCli接口实现了通过CLI从设备上获取设备配置数据或运行数据，具体处理如下：

1. 业务填写sendCli的入参有：neid、CLI命令和超时时间。
2. java侧调用CLI协议将CLI命令下发给设备。
3. 设备返回CLI命令的查询结果，通过Python代理返回给业务。

接口约束

1. 前提条件：使用该接口时，设备必须在线。
2. 注意事项：接口默认的超时时间为60s。
3. 使用限制：命令行必须为当前设备支持的命令行
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def sendCli(neid, cliCommond, timeOut=60):
    """
    sendCli is used to query devices’ config or operational information.
    :param neid:str, id of device in aoc, default value is ''
    :param cliCommond:str, cli commond ''
    :param timeOut:int timeout second"
    :return response_body:aoc.sys.sys_model_pb2.sendCli_pb2 .SendCliOutput, contains cli response

    """
```

请求参数描述

表 2-3 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neid	是	STRING	-	-	设备的标识ID。格式：uuid。
cliCommond	是	STRING	-	-	下发设备的CLI命令。 格式：STRING。
timeout	否	INT32	大于0	60000	超时时间，默认为60000ms。

请求和响应样例

Python调用样例：

```
# 引入包
from aoc.sys import cliproxy

# 调用接口得到响应结果
response_data = cliproxy.sendCli("8a394835-cb84-38f3-44d5-36a7f2074a77","display this",120)
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.2 NCE Datastore API

本章节与两阶段事务关系紧密，在两阶段事务中，编辑阶段的配置写入的是NCE的CDB（candidate database），而提交之后配置将会从CDB转移到RDB（running database）。

2.1.2.1 读取 CDB

通过事务标识和YANG子树路径读取Datastore中CDB内的数据。

类型

API-2，数据查询接口。

典型场景

还原业务配置时，查询本事务中已经编辑过的业务数据。

接口功能

携带全局事务ID，查询CDB数据。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def read_datastore_cdb(aoccontext, path, query_para=None):
    """
        read_datastore_cdb is a interface for reading cdb.
        :param aoccontext: AocContext, structure defined in aoc.base.aoccontext
        :param path: str, yang path, required field,
        :param query_para: dict type,example1: {"fields": fields=taskGroups1(nametaskGroups1), "depth": 1}
                           example2: {"where":
fields=room(name;taskGroups1(nametaskGroups1;nametaskGroups2);rack/server)}
                           example3: { "depth": 1}
        :return: str, configuration data string, defined in sys_model_pb2.datastore_pb2.ReadResult
    """
```

请求参数描述

表 2-4 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	AocContext	-	-	transactionId为必填字段。
path	是	STRING	-	-	YANG模型对应的子树路径。
query_para	否	DICT	-	-	过滤查询参数。

请求和响应样例

Python调用样例：

```
# 引入aoc datastore
from aoc import datastore

neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
path = 'huawei-ac-nes:inventory-cfg/nes/ne/868e778e-153c-3afe-a02c-89678e31e3e4/huawei-ac-ne-snmp:snmp'
aoccontext = {
    'transactionId' = '148e31d3-79a5-4da2-94fd-893adc66080e'
}
output = datastore.read_datastore_ne_cdb(aoccontext, path, neid, {'where':'neId=a1a4ce9c-a24f-11ea-ab05-caf3e5f90b35'})

# 得到的返回结果output类似于如下xml报文
<snmp xmlns=\"urn:huawei:yang:huawei-ac-ne-snmp\">
    <mibViews>
        <mibView>
            <viewName>118</viewName>
            <subtree>iso</subtree>
            <type>excluded</type>
        </mibView>
    </mibViews>
</snmp>
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.2.2 读取 RDB

通过YANG子树路径读取Datastore中RDB内的数据。

类型

API-2，数据查询接口。

典型场景

还原业务配置时，通过YANG子树查询已提交的配置数据。

接口功能

根据YANG子树路径，查询Datastore中RDB内的数据。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def read_datastore_rdb(aoccontext, path, query_para=None):
    """
    read_datastore_rdb is a interface for reading rdb.

    :param aoccontext:AocContext, structure defined in aoc.base.aoccontext
    :param query_para: dict type,example1: {"fields": fields=taskGroups1(nametaskGroups1), "depth": 1}
                       example2: {"where":
fields=room(name;taskGroups1(nametaskGroups1;nametaskGroups2);rack/server)}
                       example3: { "depth": 1}
    :return document:str, configuration data string, defined in sys_model_pb2.datastore_pb2.ReadResult
    """
```

请求参数描述

表 2-5 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	AocContext	-	-	-
path	是	STRING	-	-	YANG模型对应的子树路径。
query_para	否	DICT	-	-	过滤查询参数。

请求和响应样例

Python调用样例：

```
# 引入aoc datastore
from aoc import datastore

path = 'urn:huawei:yang:huawei-ac-applications/bng_binding'
output = datastore.read_datastore_rdb(aoccontext, path)

# 得到的返回结果output类似于如下xml报文
<bng_binding xmlns="https://example.com/example">
   <master_bng>master</master_bng>
   <slave_bng>slave</slave_bng>
   <master_info>
     <master_alias>master-alias</master_alias>
     <master_router_id>master-router-id</master_router_id>
   </master_info>
   <slave_info>
     <slave_alias>slave-alias</slave_alias>
     <slave_router_id>slave-router-id</slave_router_id>
   </slave_info>
 </bng_binding>
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.2.3 写 CDB

通过事务标识和YANG子树路径向Datastore的CDB内写入数据。

类型

API-2，数据查询接口。

典型场景

还原业务数据时，将编辑好的业务数据写到CDB。

接口功能

携带全局事务ID，向Datastore的CDB内写入数据。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def write_datastore(txid, content, path, source):
    """
    write_datastore is a interface for writing cdb in the datastore.
    Input:
```



```
    string txid, transactionId can not be None
    string content, required filed
    string path, required field
    string source, default value is None
Output:
    WriteResult object, contains the following attributes:
        string result
"""
```

请求参数描述

表 2-6 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
txid	是	STRING	-	-	transactionId为必填字段。
content	是	STRING	-	-	待写入Datastore的内容。
path	是	STRING	-	-	YANG模型对应的子树路径。
source	是	STRING	-	-	如果不考虑数据源，该值传入为空字符串。

请求和响应样例

Python调用样例：

```
# 引入aoc datastore
from aoc.sys import transaction
from aoc.sys import datastore

ne_id = '8d394835-cb84-38f3-a4d5-16a7f2074b40'
trans_id = transaction.create_transaction().transId
self.logger.info('create_transaction trans_id=%s' % trans_id)
path = '/huawei-ac-nes:inventory-cfg/nes/ne/%s/huawei-snmp:snmp' % ne_id
content = '<snmp xmlns="https://www.huawei.com/netconf/vrp/huawei-snmp">\n  <mibViews>\n
<mibView>\n       <viewName>testmibabc</viewName>\n       <subtree>iso</subtree>\n
<type>excluded</type>\n     </mibView>\n  </mibViews>\n</snmp>'

response = datastore.write_datastore(trans_id, content, path, '')
self.logger.info('write_datastore response=%s' % response)
transaction.commit_transaction(trans_id)
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.3 查询设备信息

本节介绍查询设备相关信息的接口。

2.1.3.1 查询设备基本信息

查询设备基本信息，设备的基本信息在设备导入及设备上线时已经采集到NCE。

类型

API-3，设备基本信息查询接口。

典型场景

查询设备基本信息。

接口功能

query_basic_data_from_db接口实现了查询设备的基本数据信息，具体处理如下：

1. 根据入参neid、nename、nemanageip查询其对应的设备的基本数据信息。
2. 如果入参存在neid或nename，则支持缓存加速，提高查询性能；如果入参只有nemanageip，则不支持缓存加速。

接口约束

1. 前提条件：N/A。
2. 注意事项：该接口三个入参都是可选，但至少存在一个，通过不同的组合能够过滤出满足条件的目前NCE纳管的设备。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def query_basic_data_from_db(neid='', nename='', nemanageip=''):
    """
    query_basic_data_from_db is used to query devices' detail information.
    Input:
        string neid, default value is ''
        string nename, default value is ''
        string nemanageip, default value is ''
    Output:
        QueryDevicesPagedResult object, contains the following object:
            queryDevicesPagedEntity object, contains the following attributes:
                neId, name, hardWare, softWare, manageIp, type, manufacturer,
                manageMac, deviceModel, esn, location, description, layer, get_channel_index,
                edit_channel_index
    """
```

请求参数描述

表 2-7 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neid	否	STRING	N/A	-	设备的标识ID。格式：uuid。
nename	否	STRING	N/A	-	设备的名字。
nemanageip	否	STRING	N/A	-	设备的管理口IP地址。

备注：neid、nename、nemanageip三个参数至少存在一个。

请求和响应样例

Python调用样例：

```
# 引入aoc devicemgr
from aoc import devicemgr

# 入参neid、nename、nemanageip支持组合查询
neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
nename = 'HUAWEI_ROUTER'
nemanageip = '{ip:port}'

# 得到的返回结果是QueryDevicesPagedResult类对象
result = devicemgr.query_basic_data_from_db(neid, nename, nemanageip)

 """
 QueryDevicesPagedEntity类有如下成员：
 # 网元ID
 neId: "868e778e-153c-3afe-a02c-89678e31e3e4",

 # 硬件版本号
 hardWare: "CX600-X1-M4",

 # 软件版本号
 softWare: "V800R012C10SPC680B680",

 # 管理IP
 manageIp: "{ip:port}",

 # 设备类型
 type: "ROUTER",

 # 设备厂商
 manufacturer: "HUAWEI",

 # 设备MAC地址
 manageMac: "38:BA:16:58:1E:03",

 # 设备款型
 deviceModel: "CX600-X1-M4",

 # 设备ESN号
```

```
esn: "391091484237311",

# 设备位置
location: "Beijing China",

# 描述信息
description: "Huawei Versatile Routing Platform Software \\r VRP (R) software, Version 8.120 (CX600
V800R012C10SPC680B680) \\r Copyright (C) 2012-2016 Huawei Technologies Co., Ltd. \\r",

# 设备层次
layer: "Physical",

# Netconf读通道号
get_channel_index: "b9b51c1d-38cd-35ed-9acd-2cb477762707",

# Netconf写通道号
edit_channel_index: "a377dae5-6f88-3b15-a04d-56b9f769fdbb"
"""
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.3.2 根据设备名查询设备 ID

查询设备标识ID。

类型

API-3，设备基本信息查询接口。

典型场景

用户在编写SSP包代码时，需要根据设备名查询设备ID。

接口功能

query_neid接口实现了根据设备名查询设备的标识ID，具体处理如下：

根据入参nename查询其对应的设备的标识ID。

query_neid接口支持缓存加速，提升查询性能。

接口约束

1. 前提条件：N/A。
2. 注意事项：入参nename对应Web界面上添加设备时所设置的Operation Name。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def query_neid(nename):
    """
    query_neid is used to query neid by nename.

    Input:
        string nename
    Output:
        string neid
    """
```

请求参数描述

表 2-8 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
nename	是	STRING	N/A	-	设备的名字。

请求和响应样例

Python调用样例：

```
# 引入aoc devicemgr
from aoc import devicemgr

# 入参设备名称
nename = 'HUAWEI_ROUTER'

neid = query_neid(nename)

# 得到的返回结果是一个字符串：
"868e778e-153c-3afe-a02c-89678e31e3e4"
```

错误码

错误码	错误信息	处理措施
无	无	无

2.1.3.3 查询设备状态信息

实时查询设备状态。

类型

API-3，设备基本信息查询接口。

典型场景

用户在编写SSP包代码时，需要根据设备ID、设备Name以及设备IP实时查询设备状态。

接口功能

query_status接口实现了实时查询设备的状态信息，具体处理如下：

根据入参neid、nename、nemanageip查询其对应的设备的实时状态信息。

接口约束

1. 前提条件：N/A。
2. 注意事项：该接口三个入参都是可选，但至少存在一个，查询出满足查询条件的NCE纳管的设备。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def query_status(neid='', nename='', nemanageip=''):
    """
    query_status is used to query device status by neid, nename or nemanageip

    Input:
        string neid, default value is ''
        string nename, default value is ''
        string nemanageip, default value is ''
    Output:
        string status
    """
```

请求参数描述

表 2-9 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neid	否	STRING	-	-	设备的标识ID。格式：uuid。
nename	否	STRING	N/A	-	设备的名字。
nemanageip	否	STRING	N/A	-	设备的管理口IP地址。

备注：neid、nename、nemanageip三个参数至少存在一个。

请求和响应样例

Python调用样例：

```
# 引入aoc devicemgr
from aoc import devicemgr
```

```
# 入参neid、nename、nemanageip支持组合查询
neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
nename = 'HUAWEI_ROUTER'
nemanageip = '{ip:port}'

ne_status = query_status(nename)

# 得到的返回结果是一个字符串：
"OperateDown"
```

错误码

错误码	错误信息	处理措施
无	无	无

2.2 SSP 包对应的 API

本章主要介绍在SSP包开发需要用到的接口。

2.2.1 jinja 模板

在SSP包开发网络层到网元层数据的映射功能中，根据参数生成网元配置，需要用到jinja模板渲染能力。

2.2.1.1 jinja 模板过滤器

Jinja开发手册请参考Jinja2官方文档，为了方便编码，aoc_api除了Jinja本身的过滤器支持之外，还新增了一些可能用到的方法及过滤器。

类型

API-5，模板类功能接口。

典型场景

用户使用jinja模板渲染引擎开发网络层到网元层的映射逻辑时，需要使用过滤器能力进行编码。

接口功能

Jinja模板提供系列的过滤器能力，包括设置全局变量、转换neId、转换IP地址。

接口约束

1. 前提条件：无。
2. 注意事项：to_ne_id不能本地测试，因为本接口需要到AOC库查询设备信息。
3. 使用限制：无。
4. 接口之间的关联关系：无。

GET_GLOBAL 与 SET_GLOBAL 使用方法

使用GET_GLOBAL与SET_GLOBAL方法解决在Jinja内部需要使用全局变量的问题，使用方法：

```
<example>
   #初始化temp全局变量
   {{SET_GLOBAL('temp', 0)}}
   {%- for name in class%}
      <name>{{name}}</name>
      {{GET_GLOBAL('temp') + loop.index0}}
      {%- if loop.lase %}
         {{SET_GLOBAL('temp',GET_GLOBAL('temp' + loop.index0)}}
      {%- endif%}
   {%- endfor%}
   {{GET_GLOBAL('temp')}}
</example>
```

说明

使用GET_GLOBAL与SET_GLOBAL方法访问for循环索引的示例。

表 2-10 SET_GLOBAL 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
key	是	STRING	N/A	N/A	全局变量的key。
value	是	任意类型	N/A	N/A	全局变量的内容。

表 2-11 GET_GLOBAL 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
key	是	STRING	N/A	N/A	全局变量的key。

表 2-12 GET_GLOBAL 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
value	是	任意类型	N/A	N/A	全局变量的内容。

to_ne_id 过滤器使用方法

查询设备标识ID的接口请参见：**2.1.3.2 根据设备名查询设备ID**。

```
输入:{{HUAWEI_ROUTER| to_ne_id}}
输出:868e778e-153c-3afe-a02c-89678e31e3e4
```

说明

设备名和设备ID以具体环境为准。

表 2-13 to_ne_id 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
ne_name	是	STRING	N/A	N/A	设备的名称。

表 2-14 to_ne_id 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
ne_id	是	STRING	UUID	N/A	设备的ID。

ipaddr 过滤器使用方法

ipaddr过滤器封装了 netaddr Python库的部分功能。

- 计算ip地址：

```
输入:{{ '192.168.32.0/24' | ipaddr('0')}}
输出: 192.168.32.0/24
输入:{{ '192.168.32.0/24' | ipaddr('1')}}
输出: 192.168.32.1/24
输入:{{ '192.168.32.0/24' | ipaddr('-1')}}
输出: 192.168.32.255/24
输入: {{ '192.168.32.0/24' | ipaddr('100')}}
输出: 192.168.32.100/24
```

- 计算掩码：

```
输入: {{ '192.168.32.0/24' | ipaddr('netmask')}}
输出: 255.255.255.0
输入: {{ '192.168.32.0/25' | ipaddr('netmask')}}
输出: 255.255.255.128
输入: {{ '192.168.32.0/32' | ipaddr('netmask')}}
输出: 255.255.255.255
```

- 计算网关地址：

```
输入: {{ '192.168.32.1/32' | ipaddr('network')}}
输出: 192.168.32.1
输入: {{ '192.168.32.1/255.255.255.255' | ipaddr('network')}}
输出: 192.168.32.1
输入: {{ '192.168.32.1/24' | ipaddr('network')}}
输出: 192.168.32.0
```

- 计算广播地址：

```
输入: {{ '192.168.32.1/24' | ipaddr('broadcast')}}
输出: 192.168.32.255
```

- 计算主机掩码：

```
输入: {{ '192.168.32.1/24' | ipaddr('hostmask')}}
输出: 0.0.0.255
计算广播地址：
输入: {{ '192.168.32.1/25' | ipaddr('hostmask')}}
输出: 0.0.0.127
```

2.2.1.2 jinja 模板支持 no_delete 标签定制删除

当对某个业务层配置做删除动作并提交成功，NCE默认的删除行为是同时删除配置数据与数据源，jinja模板支持no_delete和no_delete_nested两种标签，提供了一种只删除数据源不删除数据的方法。

no_delete：表示打上标记的本层节点只会被删除数据源，不会被删除数据，子节点是可以被删除数据的。

no_delete_nested：表示打上标记的本层节点以及子节点都只会被删除数据源，不会被删除数据。

说明

- 数据源：数据源表示一个真实数据的引用。例如网元数据interface1，它如果被业务1和业务2同时分解过，那么它会存在两个引用（数据源），当业务1删除的时候，因为网元数据interface1还被业务2引用，该接口还存在不会实际删除。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的no_delete标签控制业务更新时的删除配置逻辑。

接口功能

Jinja模板提供的定制业务更新删除逻辑能力。

接口约束

1. 前提条件：无。
2. 注意事项：参考下文使用方法及说明。
3. 使用限制：参考下文使用方法及说明。
4. 接口之间的关联关系：无。

使用方法

no_delete示例：

```
... ...
<ntpAuthKeyCfg xmlns:x="urn:huawei:ac" x:tag="no_delete">
... ...
```

no_delete_nested示例：

```
... ...
<system  xmlns:ns0="urn:huawei:ac" ns0:tag="no_delete_nested">
  <inform-timeout>2</inform-timeout>
  <inform-resend-times>2</inform-resend-times>
  <inform-pend-number>111</inform-pend-number>
</system>
... ...
```

说明

- 对于container节点，只支持no_delete_nested。
- no_delete下面可嵌套no_delete或者no_delete_nested，no_delete_nested下面不允许嵌套no_delete标签。
- neid不支持no_delete和no_delete_nested。

2.2.1.3 jinja 模板支持 merge/delete/replace 操作

用户在模板里面定制的网元数据的删改操作。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的操作码标签直接操作网元数据，不经过差异对比。

接口功能

Jinja模板提供定制网元的删改逻辑能力。

接口约束

1. 前提条件：无。
2. 注意事项：参考下文使用方法。
3. 使用限制：参考下文使用方法。
4. 接口之间的关联关系：无。

使用方法

merge：修改一个已存在的节点或者创建一个不存在的节点，merge是SSP默认操作。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" ns0:operation="merge">2</inform-timeout>
 ... ...
```

delete：删除一个已存在的节点，如果节点不存在不报错；如果delete操作作用在容器节点上，其子节点不能有其它显示声明操作，包括merge、replace、no-delete、no-delete-nested；delete操作不能作用于list的key节点上。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" ns0:operation="delete">2</inform-timeout>
 ... ...
```


replace: 替换一个已存在的节点或创建一个不存在的节点；如果replace操作作用在容器节点上，其子节点不能有merge或replace操作；replace操作不能作用于list的key节点或leaflist节点上。

示例：

```
... ...
<inform-timeout xmlns:ns0="urn:huawei:ac" s0:operation="replace">2</inform-timeout>
... ...
```

2.2.1.4 jinja 模板支持 order 调整顺序

用户在模板里面通过定制带order-by-user属性调整网元数据的顺序。

类型

API-5，模板类功能接口。

典型场景

用户做SSP的easymap功能时，开发网络层到网元层的映射逻辑，使用jinja模板里面的order-by-user属性操作不同实例间的顺序调整。

接口功能

Jinja模板提供的网元数据实例间顺序调整能力。

接口约束

1. 前提条件：无。
2. 注意事项：order只能作用在声明order-by user的schema节点。
3. 使用限制：无。
4. 接口之间的关联关系：无。

使用方法

对于打ordered-by属性标签的list或者leaflist节点（请参看RFC7950-YANG1.1-7.7.1），SSP支持insert操作来调整顺序。

示例：

```
... ...
<nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
<rule-list xmlns:ns0="urn:ietf:params:xml:ns:yang:1" ns0:insert="after" ns0:key="[name='name2']">
      <name>name1</name>
   </rule-list>
</nacm>
... ...
```

说明

假设已经存在name为name2的rule-list节点，表示merge一个name为name1的 rule-list节点，并且位置在name2后。

2.2.2 两阶段事务配置

Python支持两阶段事务配置接口，包括创建、编辑、提交、删除事务，进行两阶段操作。

类型

API-1，配置事务接口。

典型场景

Python侧两阶段事务配置，常用于python侧的service-rpc里面调用配置接口进行下发。

接口功能

两阶段事务操作。

接口约束

1. 前提条件：设备在线。
2. 注意事项：申请的事务如果提交失败需要显式调用事务释放接口。
3. 使用限制：无。
4. 接口之间的关联关系：无。

调用方法

Python调用接口方法：

```
# 创建
def create_transaction():
    """
    create_transaction is used to create transaction.

    Input:
        None
    Output:
        NcsServiceConfigCreateTransOutput object, contains the following attributes:
            string transId
    """


# 编辑
def edit_transaction(transid, data, path, action, neid=''):
    """
    edit_transaction is used to edit transaction.

    Input:
        string transid
        string data
        string path
        string neid
        string action, values can be create,delete,remove,merge,replace
    Output:
        NcsServiceConfigEditOutput object, contains the following attributes:
            boolean result
            string errorCode
            string errorMsg
    """
```

```
# 提交
def commit_transaction(transid):
    """
    commit_transaction is used to commit transaction.

    Input:
        string transid
    Output:
        NcsServiceConfigCommitTransOutput object, contains the following attributes:
            boolean result
            string errorCode
            string errorMsg
    """


# 删除
def delete_transaction(transid):
    """
    delete_transaction is used to reset transaction.

    Input:
        string transid
    Output:
        NcsServiceConfigResetTransOutput object, contains the following attributes:
            boolean result
            string errorCode
            string errorMsg
    """
# 回滚
def rollback_transaction(transid):
    """
    rollback_transaction is used to reset transaction.

    Input:
        string transid
    Output:
        NcsServiceConfigResetTransOutput object, contains the following attributes:
            boolean result
            string errorCode
            string errorMsg
    """
```

请求参数描述

表 2-15 两阶段事务创建接口返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transId	是	STRING	N/A	N/A	创建事务的ID 格式：uuid。

表 2-16 两阶段事务编辑接口入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transid	否	STRING	N/A	N/A	create接口返回的事务ID。 如果不填写表示一阶段下发。 格式：uuid。
data	是	STRING	N/A	N/A	对应配置的xml报文。
path	是	STRING	N/A	N/A	对应配置的YANG的子树路径。
action	是	STRING	create，merge，delete，remove，replace	N/A	对应配置的操作类型。
neid	否	STRING	N/A	N/A	如果是单站配置需要指定设备的设备ID，如果是网络配置则不用。 格式：uuid。

表 2-17 两阶段事务编辑接口返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
result	是	BOOL	true，false	false	编辑结果。
errorCode	否	STRING	N/A	N/A	编辑报错错误码。
errorMsg	否	STRING	N/A	N/A	编辑报错错误信息。

表 2-18 两阶段事务提交接口入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transId	是	STRING	N/A	N/A	事务ID。 格式：uuid。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
onlyService	否	bool	true，false	false	是否仅生效网络层。

表 2-19 两阶段事务提交接口返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
result	是	BOOL	true，false	false	提交结果。
errorCode	否	STRING	N/A	N/A	提交报错错误码。
errorMsg	否	STRING	N/A	N/A	提交报错错误信息。

表 2-20 两阶段事务回滚接口入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transId	是	STRING	N/A	N/A	事务ID。 格式：uuid。

表 2-21 两阶段事务回滚接口返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
result	是	BOOL	true，false	false	回滚结果。
errorCode	否	STRING	N/A	N/A	回滚报错错误码。
errorMsg	否	STRING	N/A	N/A	回滚报错错误信息。

请求和响应样例

Python调用样例：

```
transid= create_transaction()
# 返回事务ID
```

```
neid = '868e778e-153c-3afe-a02c-89678e31e3e4'
path = 'huawei-ac-nes:inventory-cfg/nes/ne/868e778e-153c-3afe-a02c-89678e31e3e4/huawei-ac-ne-
snmp:snmp'
date ='''
<snmp xmlns=\"urn:huawei:yang:huawei-ac-ne-snmp\">
   <mibViews>
      <mibView>
         <viewName>118</viewName>
         <subtree>iso</subtree>
         <type>excluded</type>
      </mibView>
   </mibViews>
</snmp>
'''

# 调用编辑接口
edit_transaction(transid, data, path, 'create',neid)

# 提交事务
commit_transaction(transid)
```

错误码

错误码	错误信息	处理措施
publicinfocode.fp.config.error.0x00c80022	设备 {} 下发报错，报错信息{}	设备报错，根据设备的报错信息，填写正确的参数，重新下发。

2.2.3 业务映射

服务包开发Python代码时，需要继承ncsservice.NcsService，并复写ncs_map方法。

类型

SPI-1，业务映射接口。

典型场景

开发业务包Python脚本，脚本里面完成网络业务数据到网元业务数据的映射。

接口功能

完成业务层到网元层的业务编程。

接口约束

1. 前提条件：网元存在并在线。
2. 注意事项：涉及到"<", ">", "&" 特殊字符需要用户自行编码。
3. 使用限制：无。
4. 接口之间的关联关系：无。

调用方法

- Python调用接口方法：

继承ncsservice.NcsService，并实现ncs_map方法，在业务分解的过程中，框架会调用ncs_map方法，并且把业务层数据以MapRequest入参传入ncs_map方法。

```
class MapRequest(object):
    def __init__(self, xml, context):
        self.xml = xml
        self.context = context
        self.__xmldict = None
        self.__xmldictnode = None
```

说明

MapRequest包含4个成员变量：

xml：str类型的service层配置xml报文。

xmldict：xml通过xmltodict转出的结果。

xmldictnode：xmldict 转换成dictnode的结果，更多时候我们使用的是该类型。

context：Python语言中的字典，包含一些映射逻辑需要用到的变量。

请求参数描述

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
xml	是	STRING	N/A	N/A	str类型的service层配置xml报文。
xmldict	是	字典	N/A	N/A	xml通过xmltodict转出的结果。
xmldictnode	是	DictNode	N/A	N/A	xmldict 转换成dictnode的结果，更多时候我们使用的是该类型。
context	是	字典	N/A	N/A	Python语言中的字典，包含一些映射逻辑需要用到的变量。

请求和响应样例

Python调用样例：

```
from aoc import NcsService

class AocNcsexample(NcsService):
    def ncs_map(self, request):
        return self.render('example/template_bng.j2', request.xmldictnode)
```

错误码

错误码	错误信息	处理措施
无	无	无

补充：复杂模型操作

业务包需要继承NcsService类复写ncs_map方法，其主要逻辑是处理service模型，NcsService引人了Jinja2、protobuf、xmltodict等三方依赖库，并且ncsservice还提供DictNode、NoLoopStr等辅助类，使开发服务包代码更加简洁、方便，本小节介绍如何使用这些工具类做模型操作。

初始化dictnode，模拟ncs_map输入。

```
xml = '''
   <test>
      <keyName>test</keyName>
      <value>hello world</value>
   </test>
   '''

dictnode = DictNode(xmltodict.parse(self.xml, encoding='utf-8'))
```

- 使用符号“DictNode.member”来访问DictNode中的value。
  ```
  print(dictnode.test.value) # 使用'.'来访问模型中的值
  # hello world
  ```
- 使用for遍历dictnode中的某个叶子节点并不会按照字母去遍历。
  ```
  i = 0
  for val in dictnode.test.keyName:
      i +=1;
  self.assertEqual(i, 1)
  ```
- 使用for遍历dictnode中的某个map点并不会按照去遍历而是只会遍历一次。
  ```
  for node in dictnode.test:
      print(node)
  # DictNode({u'keyName': u'test', u'value': u'hello world'})
  ```
- 使用for遍历dictnode中的list会按照list里的每个元素去遍历，这一点和list一致。
- 使用for遍历某个不存在的节点，不会报错，一次也不会循环。
  ```
  for node in dictnode.noexist:
      print(node)
  # dictnode下没有noexist，print不会执行
  ```
- 使用if 判断某个不存在的节点，返回False。
  ```
  if dictnode.noexist:
       print(dictnode.noexist)
  # dictnode下没有noexist，print不会执行
  ```
- 直接引用某个不存在的子节点，不会报错。
  ```
  print(dictnode.noexist)
  #dictnode.noexit将会返回空字符串。
  ```
- YANG模型里某个schema节点定义成list，而实际配置时list里只配置了一个leaf节点或者没有配置节点需要用到以上几点技巧。
- 使用key()方法来获取YANG模型中对应list中的item。
  ```
  nes = [
        {
            'pairName': 'pairName1',
            'hostname': 'hostname1'
        },

        {
            'pairName': 'pairName2',
            'hostname': 'hostname2'
        },

        {
            'pairName': 'pairName2',
            'hostname': 'hostname3'
        },

        {
  ```

```
        'pairName': 'pairName1',
        'hostname': 'hostname4'
    }
]

ne_map = DictNode()
ne_result = ListNode()
for ne in nes:
    ne_dictnode = DictNode(ne)
    ne_map.put(ne['hostname'], ne_dictnode)
    ne_result.append(ne_dictnode)

ne_pare = DictNode()
for hostname, ne in ne_map.items():
    if ne_pare.has_key(ne.pairName):
        pare = ne_pare.get(ne.pairName)
        ne.put('ne_peer', pare)
        pare.put('ne_peer', ne)
    else:
        ne_pare.put(ne.pairName, ne)

    # 使用key方法过滤出hostname = 'hostname1'的元素
self.assertEqual(ne_result.key(hostname = 'hostname1').ne_peer.hostname, 'hostname4')
```

2.2.4 Service RPC

服务包开发Python代码时，需要继承ncsservice.NcsService，并复写ncs_rpc方法。

类型

SPI-1，业务RPC映射接口。

典型场景

某些业务是非配置类的，用户通过开发业务RPC完成某项功能。

接口功能

继承ncsservice.NcsService，并复写ncs_rpc方法，完成业务层到网元层的业务编程。

约束

1. 前提条件：网元存在并在线。
2. 注意事项：涉及到"<", ">", "&" 特殊字符需要用户自行编码。
3. 使用限制：无。
4. 接口之间的关联关系：无。

调用方法

Python调用接口方法：

继承ncsservice.NcsService，并实现ncs_rpc方法，在service业务分解的过程中，框架会调用ncs_rpc方法，并且把业务层数据以RpcRequest入参传入ncs_rpc方法。

```
class RpcRequest(object):
    def __init__(self, xml, context):
        self.xml = xml
        self.context = context
        self.__xmldict = None
        self.__xmldictnode = None
```


说明

RpcRequest包含4个成员变量：

xml：str类型的service层配置xml报文。

xmldict：xml通过xmltodict转出的结果。

xmldictnode：xmldict 转换成dictnode的结果，更多时候我们使用的是该类型。

context：Python语言中的字典，包含一些映射逻辑需要用到的变量。

请求参数描述

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
xml	是	STRING	N/A	N/A	str类型的service层配置xml报文。
xmldict	是	字典	N/A	N/A	xml通过xmltodict转出的结果。
xmldictnode	是	DictNode	N/A	N/A	xmldict 转换成dictnode的结果，更多时候我们使用的是该类型。
context	是	字典	N/A	N/A	Python语言中的字典，包含一些映射逻辑需要用到的变量。

请求和响应样例

Python调用样例：

ncs_rpc样例：

```
from aoc import NcsService

class AocNcsDemoRpcService(NcsService):
    def ncs_rpc(self, request, arg1, arg2):
        return self.render('example/template_bng.j2', request.xmldictnode)
```

错误码

错误码	错误信息	处理措施
无	无	无

2.2.5 SSP 支持 service-inject

ssp包编写过程中有时候需要调用外部的接口，service-inject提供了一种调用外部接口的方式。

类型

SPI-4，业务服务注入接口。

典型场景

用户在做SSP的映射逻辑过程中，可能用到一些接口或者能力是非系统内置的，例如外部资源管理，需要将该外部接口或者能力通过服务注入的方式放置到系统内，供SSP的脚本调用。

接口功能

提供一整套服务提供方注入和服务使用方调用的能力。

接口约束

1. 前提条件：无。
2. 注意事项：参考下文说明。
3. 使用限制：参考下文说明。
4. 接口之间的关联关系：无。

服务提供方需继承 ServiceProvider 接口，并按要求实现 run/revert/action 方法

例子：

```
import netaddr
import random
from aoc.sys.inject.serviceprovider import ServiceProvider

class IpPool(ServiceProvider):
    ip_pool = set()
    def run(self, input):
        self.log.info('run %s' % input)
        output = {}
        if input.get("operation") == "apply":
            begin = netaddr.IPNetwork('{ip:port}').value
            end = netaddr.IPNetwork('{ip:port}').value
            value = random.randint(begin, end)
            output.setdefault("result", value)
            self.ip_pool.add(str(value))
            self.log.info('apply %s' % value)
        self.log.info('ip_pool %s' % self.ip_pool)
        return output


    def revert(self, input):
        self.log.info('revert %s' % input)
        value = input.get("result")
        self.ip_pool.discard(value)
        output = {}
        output.setdefault("result", "success")
        self.log.info('ip_pool %s' % self.ip_pool)
        return output


    def action(self, input):
        self.log.info('action %s' % input)
        if input.get("operation")  == "create":
            request = input.get("request")
            self.log.info('%s' % request.get("url"))
            self.log.info('%s' % request.get("begin"))
            self.log.info('%s' % request.get("end"))
```



```
output = {}
output.setdefault("result", "success")
self.log.info('output %s' % output)
return output
```

说明

- run/revert/action方法的输入与输出都是dict类型，不能为其它类型。
- run是有状态的，当precommit失败，开放可编程框架会调用revert方法。
- run的输入构造需要提供给使用者并且run的输出能够作为revert的输入。
- action要求能够重入，同样的输入重复调用action时不能报错。
- pkg.json里的hook-type必须为serviceprovider，hook-key也需要提供给使用者。

表 2-22 RUN 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
input	是	STRING	N/A	N/A	json报文。

表 2-23 RUN 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
output	是	STRING	N/A	N/A	json报文。

表 2-24 REVERT 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
input	是	STRING	N/A	N/A	json报文。

表 2-25 REVERT 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
output	是	STRING	N/A	N/A	json报文。

表 2-26 ACTION 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
input	是	STRING	N/A	N/A	json报文。

表 2-27 ACTION 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
output	是	STRING	N/A	N/A	json报文。

服务调用方编码示例

```
import netaddr
from aoc.ncs.ncsservice import NcsService
from aoc.sys import datastore
import traceback

class ServiceInjectConsumer(NcsService):
    service_name = 'ippool'
    def gen_request(self):
        begin = netaddr.IPNetwork('{ip:port}').value
        end = netaddr.IPNetwork('{ip:port}').value
        url = "https://aoc.serviceinject.test"
        request = {}
        request.setdefault("url", url)
        request.setdefault("begin", str(begin))
        request.setdefault("end", str(end))

        return request


    def apply_pool(self):
        input = {}
        input.setdefault("operation", "create")
        input.setdefault("request", self.gen_request())
        self.serviceinject.action(self.service_name, input)


    def apply_ip(self, key, aoccontext):
        input = {}
        input.setdefault("operation", "apply")
        input.setdefault("request", self.gen_request())
        result = self.serviceinject.run(self.service_name, key, input, aoccontext)
        ip_addr = result.get("result")
        return ip_addr


    def ncs_map(self, request, aoccontext=None, template=None):
        self.logger.info('%s' % request)
        try:
            ncs_context = dict(request.context)
            self.logger.info('%s' % ncs_context)

            if "ip_pool" not in ncs_context.keys():
                self.apply_pool();
```

```
            ncs_context.setdefault('ip_pool', 'created')

        if "vlan1" not in ncs_context.keys():
            result = self.apply_ip("vlan1", aoccontext);
            ncs_context.setdefault('vlan1', result)

    except Exception as e:
        self.logger.info("%s" % traceback.format_exc())

    return "<inventory-cfg xmlns=\"urn:huawei:yang:huawei-ac-nes\"><nes></nes></inventory-cfg>",
ncs_context
```

说明

- 通过self.serviceinject.action(self.service_name, input)调用服务提供方的action方法，service_name为服务提供方hook-key，input需按照服务提供方构造。
- 通过self.serviceinject.run(self.service_name, key, input, aoccontext)调用服务提供方run方法，key为使用方自定义用来区分其他调用的关键字。
- 通过ncs_context.setdefault('vlan1', result)设置上下文，下一次执行时开放可编程框架会将上次提交的上下文通过request.context带入。

表 2-28 RUN 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
input	是	STRING	N/A	N/A	json报文

表 2-29 RUN 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
output	是	STRING	N/A	N/A	json报文

表 2-30 ACTION 入参

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
input	是	STRING	N/A	N/A	json报文

表 2-31 ACTION 返回值

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
output	是	STRING	N/A	N/A	json报文

错误码

错误码	错误信息	处理措施
无	无	无

2.3 SND 包对应的 API

SND包中主要包含了设备管理、配置管理、数据一致性、SND CLI框架和SND RESTCONF框架相关的API。

2.3.1 设备管理定制

本节介绍设备管理的相关接口。

2.3.1.1 设备 sysoid 注册

设置设备的sysoid信息，用于纳管此款型设备。

类型

API-4，设备数据查询接口和操作接口。

典型场景

sysoid信息标识了设备的类型、款型和厂商信息。设备纳管前，需要向NCE注册设备sysoid信息。

接口功能

用户通过该接口设置设备的sysoid信息，用于纳管此款型设备。

该接口不是必须要复写，若添加设备时不带sysoid字段，那就通过pkg.json文件里的三元组来匹配；若添加设备时带了sysoid字段，需要复写此接口用于精确匹配设备，如果没复写则添加失败。

1.纳管一款网元时，需要先注册该网元的设备类型、厂商、SysOid，然后才能纳管

2.必须要注册的：设备类型、厂商

3.可选注册的：SysOid。如果纳管是期望对 SysOid 校验，注册时必须有 SysOid

4.注册的方法是重写此接口

5.附加说明：设备纳管时，会比较用户输入的类型、厂商、SysOid（如果有注册），这些参数完全吻合时，才能纳管，否则会报***错误

接口约束

1. 前提条件：N/A。
2. 注意事项：添加设备时若带了sysoid字段，就要编写此接口。
3. 使用限制：添加设备时不带sysoid字段，通过pkg.json里的三元组来匹配，若添加设备时带了sysoid字段，就要复写此接口。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def getSysoidInfo(self, aoccontext, request=None):
    """
    注册设备sysoid信息，用于纳管此款型设备。
    不是必须要复写此接口。若添加设备时不带sysoid字段，那就通过pkg.json里的三元组来匹配。
    若添加设备时带了sysoid字段，就要复写此接口用于精确匹配设备，如果没复写则添加失败。
    :param aoccontext: 上下文环境
    :param request: 方法携带的参数
    :return: 设备信息
    """
    pass
```

请求参数描述

表 2-32 输入参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。

表 2-33 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-34 SysoidInfoProtos

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
SysoidEntity	否	ARRAY_REFERENCE	请参考表SysoidEntity的详细内容	N/A	设备sysoid信息。

表 2-35 SysoidEntity

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
sysoid	否	STRING	N/A	N/A	设备sysoid值。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceModel	是	STRING	N/A	N/A	设备型号。
deviceVendor	是	STRING	N/A	N/A	设备厂商。

请求和响应样例

Python调用样例：

```
def getSysoidInfo(self, aoccontext, request=None):
    sysoidInfo = SysoidInfo()
    sysoidEntity = sysoidInfo.sysoidEntity.add()

    # 设备sysoid值
    sysoidEntity.sysoid = "1.3.6.1.4.1.2011.2.62.2.18"

    # 设备类型
    sysoidEntity.deviceType = "ROUTER"
```

```
# 设备型号
sysoidEntity.deviceModel = "NE40E-X8A"

# 设备厂商
sysoidEntity.deviceVendor = "HUAWEI"
return sysoidInfo
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.1.2 设备连接定制

定制设备的连接方式和连接能力。

类型

API-4，设备数据查询接口和操作接口。

典型场景

用于定制设备的连接能力。

接口功能

用户通过该接口定制设备的连接方式和连接能力。

接口约束

1. 前提条件：N/A。
2. 注意事项：该接口不是必须要复写，父类已提供默认实现。若对设备的连接方式和连接能力有定制化需求，可参考父类注释复写此接口。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def getConnectInfo(self, aoccontext, request=None):
    """
    定制设备连接方式和连接能力。
    不是必须要复写此接口，父类已提供默认实现。若对设备的连接方式和连接能力有定制化需求，可参考父
    类注释复写此方法。
    :param aoccontext: 上下文环境
    :param request: 方法携带的参数
    :return: 建联实例
    """
```

请求参数描述

表 2-36 输入参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。

表 2-37 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-38 ConnectInfos

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
connectInfo	否	ARRAY_REFERENCE	请参考表 ConnectInfo的详细内容	N/A	设备连接配置信息。

表 2-39 ConnectInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
protocolEntity	否	ARRAY_REFERENCE	请参考表ProtocolEntity的详细内容	N/A	设备连接协议。
connectEntity	是	ARRAY_REFERENCE	请参考表ConnectEntity的详细内容	N/A	设备类型。

表 2-40 ProtocolEntity

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
protocolType	否	ENUM	NETCONF,CLI	N/A	NETCONF协议、 CLI协议
helloEntity	是	REFERENCE	请参考表HelloEntity的详细内容	N/A	hello报文信息。

表 2-41 HelloEntity

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
helloType	否	ENUM	stardardType, extendType, defaultType	defaultType	stardardType：标准类型报文，yang对接。 extendType：扩展类型报文，yang对接，携带扩展能力集。 defaultType：缺省类型报文，schema对接。
extendEntity	是	ARRAY_STRING	N/A	N/A	若报文类型为extendType类型时，需要携带扩展能力集。

表 2-42 ConnectEntity

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
methodType	是	STRING	Java	N/A	连接接口编码语言。
agentImpl	是	STRING	N/A	N/A	连接接口实现类。
priority	是	STRING	N/A	N/A	连接调用优先级。

请求和响应样例

Python调用样例：

```
def getConnectInfo(self, aoccontext, request=None):
    connectInfos = ConnectInfos()
    connectInfo = connectInfos.connectInfo.add()

    # 设置协议类型为NETCONF
```

```
    connectInfo.protocolEntity.protocolType = ProtocolEntity.netconf

    """
    设置连接策略connectPolicy为DEFAULT_CONNECT，表示用系统默认提供的agent。
    NETCONF协议系统默认提供的是NeAgent，CLI协议系统默认提供的是CliAgent。
    若设置连接策略connectPolicy为COMPATIBLE_CONNECT，表示采用pkg.json中hooks里配置的"type"为
"snd"，
    "key"为"ecs-connect-agent"的hook里配置的agent(可配置系统自带agent，也可以写自己的agent)，
    若无hook，则要自己增加。
    """
    connectInfo.connectPolicy = DEFAULT_CONNECT

    # readChannel通道配置为SINGLE_CHANNEL，表示读通道为单通道。
    connectInfo.channelInfo.writeChannel = PROTECTED_MODE

    # writeChannel通道配置为PROTECTED_MODE，表示写通道为主备保护。
    connectInfo.channelInfo.readChannel = SINGLE_CHANNEL

    """
    is_read_share_write通道配置为false，表示读写通道不共享；若配置为true，表示读写通道共享，
    则readChannel不需要配置。
    """
    connectInfo.channelInfo.is_read_share_write = False

    """
    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。
    """
    connectInfo.protocolEntity.helloEntity.helloType = HelloEntity.defaultType

    # 若通道为HelloType.extendType类型时，需要携带扩展能力集。
    # helloEntityNewBuilder.addExtendEntity("http://www.huawei.com/netconf/capability/execute-cli/1.0");
    # helloEntityNewBuilder.addExtendEntity("http://www.huawei.com/netconf/capability/discard-commit/
1.0");

    """
    设置此协议为主协议配置，若双协议场景则需要再增加一个辅协议配置，参考双协议父类netconfclisnd
    的getConnectInfo方法。
    """
    connectInfo.connectionPriority = PRIMARY_CONNECTION
    return connectInfos
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.2 配置管理定制

用户可以自定义设备的配置信息。

2.3.2.1 设置设备驱动

类型

API-4，设备数据查询接口和操作接口。

典型场景

设备与NCE能互相通信则需要配置相关驱动信息。

接口功能

该接口实现了配置设备驱动数据。

接口约束

1. 前提条件：调用该接口时，必须满足设备在线。
2. 注意事项：设备必须在线。
3. 使用限制：驱动参数设置与实际设备保持一致
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
# 定制设备通用驱动
def getCommonDriverInfo(self, aoccontext, request=None):
    pass

# 定制设备NETCONF驱动
def getNetconfDriverInfo(self, aoccontext, request=None):
    pass
```

请求参数描述

表 2-43 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。

表 2-44 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
deviceVersio	是	STRING	N/A	N/A	设备软件版本号。

表 2-45 CommonDriverInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rollbackLastErrPackage	否	BOOL	true/false	false	下发配置过程中设备返回报错，回滚时是否下发配置错误报文的逆向报文。如：下发报文1/2/3/4/5，下发1/2成功，下发报文3失败，若该字段字段为false，则回滚阶段下发报文1/2的逆向报文，若该字段为true，则回滚阶段下发报文1/2/3的逆向报文。
pathNeedPreProcess	否	STRING	N/A	N/A	该字段填写需要进行报文前置处理的特性QName。若该字段填写了对应QName，对于设备返回的查询结果报文，会在处理之前先调用netconfPreprocess进行加工，加工之后再进行报文转换。
pathNeedPostProcess	否	STRING	N/A	N/A	该字段填写需要进行报文后置处理的特性QName。若该字段填写了对应QName，对于即将下发设备的报文，会在下发之前先调用netconfPostprocess进行加工，加工之后再下发设备。
pathNeedEcsMapping	否	STRING	N/A	N/A	该字段填写需要进行业务报文定制处理的特性QName。若该字段填写了对应QName，对于下发设备的报文和设备返回的都统一调用toDocument、toDataObject进行转换。
pathNeedEcsRpcMapping	否	STRING	N/A	N/A	该字段填写需要进行rpc报文定制处理的rpc QName。若该字段填写了对应QName，对于下发设备的rpc报文和设备返回的都统一调用toRpcDocument、toRpcDataObject进行转换。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
para	否	EcsDeviceDriverPara	N/A	N/A	该字段为扩展字段，为key、value的键值对，目前数据一致性会使用到。
isNeedGlobalPush	否	STRING	"true"/"false"	"false"	是否进行分包下发的全局配置。当该值为"false"时，同一个事务中同一设备下发的报文会进行打包，之后统一下发。当该值为"true"时，同一个事务中同一设备下发的报文不会进行打包。
isPathNeedPush	否	STRING	"true"/"false"	"false"	对于该特性的报文是否进行分包下发。当该值为"false"时，同一个事务中同一设备下发的报文会进行打包，之后统一下发。当该值为"true"时，同一个事务中同一设备下发的报文，当前特性报文不会与后续报文进行打包。
pathNeedEcsDBHook	否	STRING	N/A	N/A	下发配置到设备后，某些配置会联动生成其他配置，而NCE只会存储联动前的配置，会导致NCE和转发器配置不一致。因此，NCE端也需要在配置下发前进行联动处理。
checkSync	否	BOOL	true/false	-	NCE下发配置是否校验NCE和设备的一致性： 是，不一致报错； 否，不校验。
unsupportedOperations	否	STRING	"create"/"delete"/"create,delete"	N/A	设备不支持的操作码，配置之后会对操作进行自动转换。当配置了create，下发create操作报文会自动转换成merge操作；当配置了delete，下发delete操作报文会自动转换成remove操作。
pathUnsupportedOperations	否	MAP<STRING,STRING>	N/A	N/A	设备对于某些特性不支持的操作码，配置之后会对操作进行自动转换。当配置了create，对于该特性下发create操作报文会自动转换成merge操作；当配置了delete，对于该特性下发delete操作报文会自动转换成remove操作。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
deleteStrategy	否	DeleteStrategy	PATH_ADD/ CONTAINER_WITH_PRESENCE	N/A	对于delete、remove操作的报文，是否删除报文中的具体信息。当为CONTAINER_WITH_PRESENCE时，下发报文不做内容删除。
uiSupportAbility	否	UiSupportAbility	N/A	N/A	基于设备版本定制界面操作能力和连接协议。
configDeliveryStrategy	否	ConfigDeliveryStrategy	SYNC_DB_AND_SYNC_DEVICE/ SYNC_DB_AND_ASYNC_DEVICE	N/A	配置下发策略。 SYNC_DB_AND_SYNC_DEVICE：存库并立刻下发设备。 SYNC_DB_AND_ASYNC_DEVICE：存库之后配置并不立刻下发设备，会在设备空闲并上线的情况下再下发。
outOfSyncDeviceStrategyInHASwitching	否	OutOfSyncDeviceStrategyInHASwitching	N/A	N/A	NCE主备倒换后基于设备版本定制的处理策略： ALARM：告警。 SYNV_TO_DEVICE：通知数据一致性对账。 CUSTOM_HOOK：务挂接处理钩子。
SwitchHACustomHook	否	STRING	N/A	N/A	NCE主备倒换后基于设备版本定制的处理策略之业务挂接钩子。
ecsCheckHook	否	EcsCheckHook	N/A	N/A	根据path定制的资源监测处理。
ecsBehaviourHook	否	EcsBehaviourHook	N/A	N/A	行为驱动。

表 2-46 EcsCheckHook

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
moduleName	是	STRING	N/A	N/A	业务特性名称。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
appClass	是	STRING	N/A	N/A	实现类所在包路径。

表 2-47 EcsBehaviourHook

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
apiClass	是	STRING	N/A	N/A	接口类。
implClass	是	STRING	N/A	N/A	实现类。

表 2-48 NetconfDriverInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
classification	否	STRING	"huawei-v5"/"huawei-v8"/"cisio"	N/A	设备原型。
phase	否	STRING	"one"/"two"	N/A	设备支持的几阶段下发。
maxPkgLength	否	INT32	>0	N/A	设备支持的最大单个报文的大小。
netconfLock	否	BOOL	true/false	false	下发设备时是否需要先获取设备锁。若设备为juniper的设备，该字段须为true。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
errorOption	否	STRING	"rollback-on-error"/"stop-on-error"/"continue-on-error"	"rollback-on-error"	设备配置数据出错之后的处理策略。
testOption	否	STRING	"set"	N/A	下发设备的报文是否需要加上<test-option>标签，当该字段为“set”时，下发报文会加上<test-option>set</test-option>。
netconfHelloEntity	否	NetconfHelloEntity	N/A	N/A	设备管理hello报文定制： standardType：标准报文 extendType：自定义报文 defaultType：默认报文（YANG对接）
modelDiff	否	STRING	"same"/"match"	N/A	netconf报文转换类型。当为same时，下发的报文的namespace会统一转换为"https://www.huawei.com/netconf/vrp"。
rpcPrefix	否	RpcPrefix	N/A	N/A	下发的rpc报文是否需要添加前缀节点。

表 2-49 RpcPrefix

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcQname	是	STRING	N/A	N/A	特性QName。
prefix	是	STRING	N/A	N/A	rpc报文的前缀节点名。

请求和响应样例

Python调用样例：

```
# 定制设备通用驱动
def getCommonDriverInfo(self, aoccontext, request=None):
    self.logger.info('getCommonDriverInfo start.')
    common_driver = CommonDriverInfo()

    """
    对于delete、remove操作的报文，是否删除报文中的具体信息。
    当为CONTAINER_WITH_PRESENCE(数值为1)时，下发报文不做内容删除。
    """
    common_driver.deleteStrategy = 1
    syncToDel = common_driver.para.add()

    """
    可选配置，设置数据对账是否支持删除南向设备配置实例。
    取值：false（默认值，不支持）、true（支持）。
    注意：设置数据对账支持删除南向设备的配置实例，如果误操作对账，可能会导致南向设备网络中断。
    """
    syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"

    self.logger.info('getCommonDriverInfo end.')
    return common_driver


# 定制设备NETCONF驱动
def getNetconfDriverInfo(self, aoccontext, request=None):
    self.logger.info('getNetconfDriverInfo start.')
    netconf_driver = NetconfDriverInfo()
    netconf_driver.phase = "two"
    netconf_driver.classification = "huawei-v5"
    self.logger.info('getNetconfDriverInfo end.')
    return netconf_driver
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.2.2 数据联动

类型

API-4，设备数据查询接口和操作接口。

典型场景

下发配置到设备后，某些配置会联动生成其他配置，而NCE只会存储联动前的配置，会导致NCE和转发器配置不一致。因此，NCE端也需要在配置下发前进行联动处理。

接口功能

生成配置下发到设备的联动数据。

接口约束

1. 前提条件：调用该接口时，设备必须在线。
2. 注意事项：产生的联动数据与设备行为保持一致
3. 使用限制：只能在配置下发产生联动数据时使用
4. 接口之间的关联关系：依赖于**2.3.2.1 设置设备驱动**的pathNeedEcsDBHook字段

调用方法

Python调用接口方法：

```
# 数据联动处理
# @param aoccontext 上下文环境
# @param request 需要联动的数据
def dbPostProcess(self, aoccontext, request):
    pass
```

请求参数描述

表 2-50 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
request	是	REFERENCE	请参考表 EcsDbConfig的详细内容	N/A	数据联动入参。

表 2-51 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersio	是	STRING	N/A	N/A	设备软件版本号。

表 2-52 EcsDbConfig

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	路径。
data	是	STRING	N/A	N/A	数据。
opType	是	STRING	N/A	N/A	操作类型。

请求和响应样例

Python调用样例：

```
# 数据联动处理
# @param aoccontext 上下文环境
# @param request 需要联动的数据
def dbPostProcess(self, aoccontext, request):
    self.logger.info('dbPostProcess start.')
    ecsData = request.data
    ecsPath = request.path
    ecsopType = request.opType

    #  /huawei-ifm:interfaces/huawei-ifm:interface/asd/huawei-ifm:mpls/huawei-ifm:l2vc/primary
    ecsDbConfigOut = EcsDbConfigOut()

    # /huawei-ifm:interfaces/huawei-ifm:interface/Giga0000001/huawei-ifm:mpls/huawei-ifm:l2vc/secondary
    if "huawei-ifm:l2vc/primary" in ecsPath and ecsopType == "DELETE":
        ifmKey = self.get_key(ecsPath)
        ecsDbconfig = ecsDbConfigOut.ecsDbConfig.add()
        ecsDbconfig.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + \
                           "/huawei-ifm:mpls/huawei-ifm:l2vpn/huawei-ifm:service-name"
    elif "huawei-ifm:l2vc/secondary" in ecsPath and ecsopType == "DELETE":
        ifmKey = self.get_key(ecsPath)
        ecsDbconfig1 = ecsDbConfigOut.ecsDbConfig.add()
        ecsDbconfig2 = ecsDbConfigOut.ecsDbConfig.add()
        ecsDbconfig1.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + \
                            "/huawei-ifm:mpls/huawei-ifm:l2vpn/reroute"
```

```
    ecsDbconfig2.path = "/huawei-ifm:interfaces/huawei-ifm:interface/" + ifmKey + \
                        "/huawei-ifm:mpls/huawei-ifm:l2vpn/redundancy"
  self.logger.info('dbPostProcess end.')
  return ecsDbConfigOut
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.2.3 业务自定义

类型

API-4，设备数据查询接口和操作接口。

典型场景

设备不支持通用的数据，需要业务自定义处理。

接口功能

提供匹配设备的数据。

接口约束

1. 前提条件：调用该接口时，必须满足设备在线。
2. 注意事项：自定义转换规则与设备保持一致
3. 使用限制：由业务自定实现，保证规则转换正确性
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
# 用户自定义处理
# @param aoccontext 上下文环境
# @param request 需要处理的数据
def netconfTransformer(self, aoccontext, request):
    pass
```

请求参数描述

表 2-53 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
request	是	REFERENCE	请参考表 AtomicConfig的详细内容	N/A	自定义业务入参。

表 2-54 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersio	是	STRING	N/A	N/A	设备软件版本号。

表 2-55 AtomicConfig

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	yang模型路径。
data	是	STRING	N/A	N/A	yang模型数据。
optype	是	STRING	N/A	N/A	操作类型。

表 2-56 NetconfMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
msg	是	STRING	N/A	N/A	报文。
isRpc	是	BOOL	N/A	N/A	是否是RPC配置报文。

请求和响应样例

Python调用样例：

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

错误码

错误码	错误信息	处理措施
无	无	无

2.3.2.4 前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

从设备同步到NCE的数据，可能出现需要对这些数据进行修改才能够被NCE识别。

接口功能

对设备同步到NCE的数据进行微调，使NCE能够识别。

接口约束

1. 前提条件：调用该接口时，必须满足设备在线。
2. 注意事项：保证报文前置处理正确性

3. 使用限制：只能通过Netconf协议下发时使用
4. 接口之间的关联关系：依赖于2.3.2.1 设置设备驱动的pathNeedPreProcess字段

调用方法

Python调用接口方法：

```
# 前置处理
# @param aoccontext 上下文环境
# @param request 需要处理的数据
# @return 处理后的数据
def netconfPreprocess(self, aoccontext, request):
    pass
```

请求参数描述

表 2-57 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
request	是	REFERENCE	请参考表 NetconfAtomicConfig的详细内容	N/A	前置处理入参。

表 2-58 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersio	是	STRING	N/A	N/A	设备软件版本号。

表 2-59 NetconfAtomicConfig

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	Netconf协议的xml报文

请求和响应样例

Python调用样例：

```
def netconfPreprocess(self, aoccontext, request=None):
    self.logger.info('netconfPreprocess start.')
    doc = request.data
    new_doc = doc.replace("<viewName>", "<viewName>pre")
    netconfAtomicConfig = NetconfAtomicConfig()
    netconfAtomicConfig.data = new_doc
    self.logger.info('netconfPreprocess end.')
    return netconfAtomicConfig
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.2.5 后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

从NCE同步到设备的数据，需要数据的某些字段进行转换，保证设备可以正常识别。

接口功能

对NCE同步到设备的数据进行微调，使设备能够识别。

接口约束

1. 前提条件：调用该接口时，必须满足设备在线。
2. 注意事项：保证报文后置处理的正确性
3. 使用限制：只能通过Netconf协议下发时使用
4. 接口之间的关联关系：依赖于**2.3.2.1 设置设备驱动**的pathNeedPostProcess字段

调用方法

Python调用接口方法：

```
# 后置处理
# @param context 上下文环境
# @param input 需要处理的数据
# @return 处理后的数据
def netconfPostprocess(self, aoccontext, request):
    pass
```

请求参数描述

表 2-60 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
request	是	REFERENCE	请参考表 NetconfAtomicConfig的详细内容	N/A	后置处理的入参。

表 2-61 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersio	是	STRING	N/A	N/A	设备软件版本号。

表 2-62 NetconfAtomicConfig

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	Netconf协议的xml报文

请求和响应样例

Python调用样例：

```
def netconfPostprocess(self, aoccontext, request=None):
    self.logger.info('netconfPostprocess start.')
    doc = request.data
    new_doc = doc.replace("<viewName>", "<viewName>post")
    netconfAtomicConfig = NetconfAtomicConfig()
    netconfAtomicConfig.data = new_doc
    self.logger.info('netconfPostprocess end.')
    return netconfAtomicConfig
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.3 数据一致性定制

2.3.3.1 特性定制

类型

API-4，设备数据查询接口和操作接口。

典型场景

用户由于业务需要，自定义差异发现、对账和同步使用的特性范围。

数据一致性支持分析SND包中的YANG模型，根据每个模块的顶层容器和顶层List配置节点，生成差异发现、对账和同步使用的数据路径。

接口功能

定制数据一致性差异发现、对账和同步使用的数据路径。

接口约束

1. 前提条件：设备必须在线。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：依赖*2.3.1.2 设备连接定制*和*2.3.2.1 设置设备驱动*。

调用方法

Python调用接口方法：

```
def getFeatures(self, aoccontext, request=None):
    """
    用户定制数据一致性差异发现、对账和同步使用的数据路径。
    :param aoccontext:上下文环境
    :param request:方法携带的参数
    :return:用户定制的特性信息
    """
```

请求参数描述

表 2-63 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。

表 2-64 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-65 FeatureCfgsMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
replace	否	BOOLEAN	N/A	false	是否定制特性；默认为否；如果值为true，特性定制方法返回值生效。
features	是	List<Feature>	N/A	N/A	特性列表。

表 2-66 Feature

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
name	是	STRING	N/A	N/A	特性名称（模块名称：例如huawei-ifm）。
operType	否	Oper	MERGE，REPLACE，DELETE	MERGE	特性与YANG文件合并规则。MERGE（默认值）：定制字段合并到YANG默认生成的特性中；REPLACE：该特性以定制为准；DELETE：删除该特性。
depends	否	List<STRING>	N/A	N/A	依赖其他特性（特性的Name字段）的列表。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
functions	否	List<Function>	N/A	N/A	特性对应的数据路径列表。

表 2-67 Function

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
value	是	STRING	N/A	N/A	差异比较用数据路径。例如：(https://www.huawei.com/netconf/vrp/huawei-ifm?revision=2018-06-11)ifm
collectPath	否	STRING	N/A	N/A	从南向设备采集配置用数据路径，默认同value字段。例如：(https://www.huawei.com/netconf/vrp/huawei-ifm?revision=2018-06-11)ifm
collectFilter	否	STRING	N/A	N/A	从南向设备采集配置的过滤器，默认采集数据路径对应的所有配置。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
supportDel	否	Switch	ENABLE,DISABLE	DISABLE	是否支持删除南向设备配置实例。如果未指定，以全局设置为准。全局配置参考：设备驱动中的getCommonDriverInfo()方法； 注意：设置数据对账支持删除南向设备的配置实例，如果误操作对账，可能会导致南向设备网络中断。
preSyncToNe	否	BOOLEAN	false,true	false	是否对账前定制。设置成false，不支持对账前定制。设置成true，参考**2.3.3.2 对账数据定制**。
preSyncFromNe	否	BOOLEAN	false,true	false	是否同步前定制。设置成false，不支持同步前定制。设置成true，参考**2.3.3.3 同步数据定制**。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
postSyncFromNe	否	BOOLEAN	false,true	false	是否同步后定制。设置成false，不支持同步后定制。设置为true，参考**2.3.3.4 同步后处理**。

请求和响应样例

Python调用样例：

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

        # 新增一个特性定制
        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
        function.preSyncToNe = False

        # 设置同步前定制
        function.preSyncFromNe = False

        # 设置同步后定制
        function.postSyncFromNe = False
        feature.functions.extend([function])
        return [feature]
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.3.2 对账数据定制

类型

API-4，设备数据查询接口和操作接口。

典型场景

用户需要修改数据一致性对账时，下发给南向设备的数据。例如，南向设备的配置A不参与对账时，可以通过对账数据定制，将配置A从对账数据中移除。

接口功能

定制数据一致性对账使用的数据。

接口约束

1. 前提条件：设备必须在线,。
2. 注意事项：N/A。
3. 使用限制：特性定制的preSyncToNe属性设置为true，方法生效。
4. 接口之间的关联关系：依赖*2.3.1.2 设备连接定制*、*2.3.2.1 设置设备驱动*和2.3.3.1 特性定制。

调用方法

Python调用接口方法：

```
def preSyncToNe(self, request, aoccontext):
    """
    用户定制数据一致性对账使用的数据。
    :param request:方法携带的参数
    :param aoccontext:上下文环境
    :return: 用户定制的特性信息
    """
```

请求参数描述

表 2-68 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
diffDataInMsg	是	REFERENCE	请参考表 DiffDataInMsg的详细内容。	N/A	差异数据信息。
neId	是	STRING	N/A	N/A	设备ID。
featureId	是	STRING	N/A	N/A	特性ID。
ChangeData	是	REFERENCE	N/A	N/A	转发器数据和控制器数据，可以合并出差异。

表 2-69 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-70 DiffDataInMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
feature	是	STRING	N/A	N/A	特性名称。
regPath	否	STRING	N/A	N/A	业务注册的数据路径。
transId	是	STRING	N/A	N/A	事务ID。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
diffDatas	是	List<STRING>	N/A	N/A	差异数据列表。差异数据为XML格式，使用<diff>标签标示存在差异的数据，<left>为NCE侧数据，<right>为南向设备侧数据。数据一致性对账以NCE侧数据为准，修改南向设备数据。

表 2-71 DiffDataOutMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
diffDatas	是	LIST<REFERENCE>	请参考表DiffData的详细内容。	N/A	业务处理后返回的数据。

表 2-72 DiffData

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
singleMsg	否	BOOLEAN	true/false	false	该差异数据是否需要单独提交到南向设备。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
diffData	是	STRING	N/A	N/A	差异数据，XML格式。使用<diff>标签标示存在差异的数据，<left>为NCE侧数据，<right>为南向设备侧数据。数据一致性对账以NCE侧数据为准，修改南向设备数据。

请求和响应样例

Python调用样例：

```
def preSyncToNe(self, request, aoccontext=None):
    self.logger.info('preSyncToNe start.')
    dataIn = request

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

    # 获得特性名称
    featureName = dataIn.feature

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

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

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

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

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

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

        # TODO 处理diffXml
        diffDataOut = diffData()

        # 该差异数据是否需要单独提交到南向设备。false表示不需要单独提交到南向设备
```

```
        diffDataOut.singleMsg = False
        diffDataOut.diffData = diffXml
        dataOut.diffDatas.extend([diffDataOut])
    self.logger.info('preSyncToNe end.')
    return dataOut
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.3.3 同步数据定制

类型

API-4，设备数据查询接口和操作接口。

典型场景

用户需要修改数据一致性同步时，保存到NCE的数据。例如，对从南向设备采集回来的密码字段，加密后保存到NCE。

接口功能

定制数据一致性同步使用的数据。

接口约束

1. 前提条件：设备必须在线；特性定制的preSyncFromNe属性设置为true，方法生效。
2. 注意事项：有两种实现方式，已有接口AbstractSND和模型规则新增接口YangSND。
3. 使用限制：N/A。
4. 接口之间的关联关系：依赖*2.3.1.2 设备连接定制*、*2.3.2.1 设置设备驱动*和2.3.3.1 特性定制。

调用方法

Python调用接口方法：

```
def preSyncFromNe(self, request, aoccontext):
    """
    用户定制数据一致性同步使用的数据。
    :param request:方法携带的参数
    :param aoccontext:上下文环境
    :return: 用户定制的特性信息
    """
```

请求参数描述

表 2-73 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
diffDataInMsg	是	REFERENCE	请参考表 DiffDataInMsg的详细内容。	N/A	差异数据信息。
neId	是	STRING	N/A	N/A	设备ID。
featureId	是	STRING	N/A	N/A	特性ID。
ChangeData	是	REFERENCE	N/A	N/A	转发器数据和控制器数据，可以合并出差异。

表 2-74 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-75 DiffDataInMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
feature	是	STRING	N/A	N/A	特性名称。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
regPath	否	STRING	N/A	N/A	业务注册的数据路径。
transId	是	STRING	N/A	N/A	事务ID。
diffDatas	是	List<STRING>	N/A	N/A	差异数据列表。差异数据为XML格式。使用<diff>标签标示存在差异的数据，<left>为NCE侧数据，<right>为南向设备侧数据。数据一致性同步以南向设备侧数据为准，修改NCE侧数据。

表 2-76 DiffDataOutMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
diffDatas	是	LIST<REFERENCE>	请参考表DiffData的详细内容。	N/A	业务处理后返回的数据。

表 2-77 DiffData

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
singleMsg	否	BOOLEAN	true/false	false	该差异数据是否需要单独提交到南向设备。该字段只在数据对账时有效。
diffData	是	STRING	N/A	N/A	差异数据为XML格式。使用<diff>标签标示存在差异的数据，<left>为NCE侧数据，<right>为南向设备侧数据。数据一致性同步以南向设备侧数据为准，修改NCE侧数据。

请求和响应样例

Python调用样例：

```
  def preSyncFromNe(self, request, aoccontext):
      self.logger.info('preSyncFromNe 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("preSyncFromNe 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

    # 定制数据一致性同步使用的数据
    dataOut = DiffDataOutMsg()

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

        # TODO 处理diffXml
        diffDataOut = diffData()

        # 该差异数据是否需要单独提交到南向设备。该字段只在数据对账时有效。
        diffDataOut.singleMsg = False
        diffDataOut.diffData = diffXml
        dataOut.diffDatas.extend([diffDataOut])
    self.logger.info('preSyncFromNe end.')
    return dataOut
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.3.4 同步后处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

用户需要在数据一致性同步后，占用业务资源。例如，从南向设备同步回L3VPN实例后，需要占用标签资源。

接口功能

定制数据一致性同步后处理。

接口约束

1. 前提条件：设备必须在线；特性定制的postSyncFromNe属性设置为true，方法生效。
2. 注意事项：有两种实现方式，已有接口AbstractSND和模型规则新增接口YangSND。
3. 使用限制：N/A。
4. 接口之间的关联关系：依赖*2.3.1.2 设备连接定制*、*2.3.2.1 设置设备驱动*和2.3.3.1 特性定制。

调用方法

Python调用接口方法：

```
def postSyncFromNe(self, request, aoccontext):
    """
    用户定制数据一致性同步后处理。
    :param request:方法携带的参数
    :param aoccontext:上下文环境
    :return: 用户定制的特性信息
    """
```

请求参数描述

表 2-78 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表 AocContext的详细内容。	N/A	上下文信息。
diffDataInMsg	是	REFERENCE	请参考表 DiffDataInMsg的详细内容。	N/A	差异数据信息。
neId	是	STRING	N/A	N/A	设备ID。
featureId	是	STRING	N/A	N/A	特性ID。
ChangeData	是	REFERENCE	N/A	N/A	转发器数据和控制器数据，可以合并出差异。

表 2-79 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-80 DiffDataInMsg

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
feature	是	STRING	N/A	N/A	特性名称。
regPath	否	STRING	N/A	N/A	业务注册的数据路径。
transId	是	STRING	N/A	N/A	事务ID。
diffDatas	是	List<STRING>	N/A	N/A	差异数据列表。差异数据为XML格式。使用<diff>标签标示存在差异的数据，<left>为NCE侧数据，<right>为南向设备侧数据。数据一致性同步以南向设备侧数据为准，修改NCE侧数据。

请求和响应样例

Python调用样例：

```
    def postSyncFromNe(self, request, aoccontext):
        self.logger.info('postSyncFromNe 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("postSyncFromNe 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

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

        # TODO
    self.logger.info('postSyncFromNe end.')
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.3.5 差异比较方式定制

类型

API-4，设备数据查询接口和操作接口。

典型场景

数据一致性差异比较方式根据数据源的不同分为E2E方式和非E2E方式(WHOLE_DIFF)。E2E差异是以控制器数据为基准比较的，非E2E以转发器数据为基准比较的。

接口功能

定制设备类型和差异比较方式。

接口约束

1. 前提条件：设备必须在线。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：依赖*2.3.1.2 设备连接定制*和*2.3.2.1 设置设备驱动*。

调用方法

Python调用接口方法:

```
def getEcsConfigParams(self, aoccontext, request=None):
    """
    定制差异比较方式
    :param aoccontext:上下文
    :param request:None
    :return:EcsConfigOut
    """
```

请求参数描述

表 2-81 参数列表

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
aoccontext	是	REFERENCE	请参考表AocContext的详细内容。	N/A	上下文信息。
EcsConfigOut	是	REFERENCE	N/A	N/A	一致性的配置数据。

表 2-82 AocContext

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
transactionId	否	STRING	N/A	N/A	事务ID，保证数据一致性，启用事物机制。
deviceVendor	是	STRING	N/A	N/A	设备厂商。
deviceType	是	STRING	N/A	N/A	设备类型。
deviceVersion	是	STRING	N/A	N/A	设备软件版本号。

表 2-83 EcsConfigOutPara

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
EcsConfigOutPara	否	REFERENCE	N/A	N/A	该字段为扩展字段，为key、value的键值对。

请求和响应样

Python调用样例:

```
def getEcsConfigParams(self, aoccontext, request=None):
    """
    定制差异比较方式
    :param aoccontext:上下文
    :param request:None
    :return:EcsConfigOut
    """
    return None
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4 SND CLI 框架定制

2.3.4.1 透传白名单获取

类型

API-4，设备数据查询接口和操作接口。

典型场景

在CLI命令透传功能中，客户可以定制透传命令的白名单。

接口功能

客户可以指定符合什么条件的命令可以透传下发到设备。这样就保证了业务安全。

接口约束

1. 前提条件：SND包的resources目录下有CliPassthroughCommands.xml配置文件。
2. 注意事项：XML配置文件中，配置的命令是正则表达式。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def getPassthroughCommands(self, aoccontext):
    """
    从SND包resources目录的XML配置文件中获取命令行白名单
    :param aoccontext: 上下文环境
    :return: 命令行白名单
    """
    pass
```


请求参数描述

表 2-84 CliPassthroughOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
commands	否	CliPassthroughCommand	参考表CliPassthroughCommand	N/A	命令白名单列表。

表 2-85 CliPassthroughCommand

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
command	否	STRING	N/A	N/A	命令正则表达式。

请求和响应样例

Python调用样例：

```
def getPassthroughCommands(self, aoccontext):
    self.logger.info("getPassthroughCommands begin")
    output = CliPassthroughOutput()

    # 新建File对象引用XML配置文件
    file_path = os.path.join(self.resourceDir, "resources/CliPassthroughCommands.xml")

    # 解析XML配置文件
    tree = ET.parse(file_path)
    root = tree.getroot()

    # 通过配置文件内容构建返参
    for child in root:
        value = codecs.getdecoder("unicode_escape")(child.text)[0]
        commandEntity = output.add()
        commandEntity.command = value
    self.logger.info('getPassthroughCommands end.')
    return output
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4.2 Yang 转 CLI

类型

API-4，设备数据查询接口和操作接口。

典型场景

在CLIDriver不能支持的特殊情况下，使用自定义yang转cli方法来解决转换问题。

接口功能

客户可以自定义yang转cli的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def yangToCli(self, aoccontext, request):
    """
    自定义Yang转CLI逻辑
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```

请求参数描述

表 2-86 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表CommandLine	N/A	请求的命令行。

表 2-87 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-88 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-89 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

```
def yangToCli(self, aoccontext, request):
    res_data = None

    # 获取请求path，根据不同path做不同业务处理
    if request.path.startswith("/path1"):
        # 根据不同请求操作类型，做不同业务处理
        if request.oper_type == OperType.Value('CREATE'):
            res_data = 'c1 leaf1 value1'
        elif request.oper_type == OperType.Value('MERGE'):
            res_data = 'c1 leaf1 value2'
        elif request.oper_type == OperType.Value('DELETE'):
            res_data = 'undo c1 leaf1'

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

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4.3 CLI 转 Yang

类型

API-4，设备数据查询接口和操作接口。

典型场景

在CLIDriver不能支持的特殊情况下，使用自定义cli转yang方法来解决转换问题。

接口功能

客户可以自定义cli转yang的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def cliToYang(self, aoccontext, request):
    """
    自定义CLI转YANG逻辑
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```

请求参数描述

表 2-90 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表 CommandLine	N/A	请求的命令行。

表 2-91 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-92 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-93 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
```

```
# 获取请求Path，根据不同的Path做不同的业务处理
if request.path.startswith("/cli-custom:c1"):
    out.data = "<c1 xmlns=\"http://huawei.com/cli-custom\"><f1>hello_world</f1></c1>"
    out.endOffset = str(request.data).__len__()
return out
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4.4 前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

设备返回配置报文后，再报文传给CLIDriver之前，对配置报文进行一次处理。

接口功能

CLIDriver会根据YANG模型进行CLI-YANG的默认转换。但在某些特殊情况下，设备返回的CLI命令行不能直接用作CLIDriver的转换入参，需要再进行一次自定义的修改，才可以达到作为CLI-YANG转换入参的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：扩展项annotation必须加在顶层YANG节点。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def readCliPreProcessor(self, aoccontext, request):
    """
    自定义前置处理方法
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```


请求参数描述

表 2-94 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表 CommandLine	N/A	请求的命令行。

表 2-95 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-96 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-97 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表 CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

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

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

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

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4.5 后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

在CLIDriver生成下发设备的命令后，再对生成的命令进行一次处理。

接口功能

CLIDriver会根据YANG模型进行YANG-CLI的默认转换。但在某些特殊情况下，CLIDriver生成的CLI命令不能满足设备的要求，需要再进行一次自定义的修改，才能达到下发设备的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：扩展项annotation必须加在顶层YANG节点。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCliPostProcessor(self, aoccontext, request):
    """
    自定义后置处理
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```

请求参数描述

表 2-98 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表CommandLine	N/A	请求的命令行。

表 2-99 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-100 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-101 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表 CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

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

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

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

错误码

错误码	错误信息	处理措施
无	无	无

2.3.4.6 后置按行处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

在CLIDriver生成下发设备的命令后，再对生成的命令进行一次处理。处理方式是按行处理，如果设备不支持两阶段事务，CLIDriver可以回滚。

接口功能

在后置处理的基础上，使不支持事务的设备可以回滚。

接口约束

1. 前提条件：N/A。
2. 注意事项：此方法可以使不支持事务的设备进行自动回滚，但不要变量命令行数量；也不要改变命令行顺序。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCliByLinePostProcessor(self, aoccontext, request):
    """
    自定义按行后置处理
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```

请求参数描述

表 2-102 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表 CommandLine	N/A	请求的命令行。

表 2-103 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-104 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-105 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

```
def configCliByLinePostProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
```

```
    # 获取请求Path，根据不同的path做不同业务处理
    req_path = request.path
    if req_path == '/path1':
        # 对每个请求命令行对象，新建一个返回命令行对象
        for item in request.commandLines:
            command = out.commandLines.add()

            # 设置命令行对象的索引值
            command.index = item.index

            # 对命令行做自定义处理
            if 'a' in item.commandLine:
                command.commandLine = item.commandLine.replace('a', 'b')
    return out
```

2.3.4.7 回滚后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

一阶段事务的设备，在CLIDriver生成下发设备的命令后，如果某个命令行下发失败，会自动触发CLIDriver的回滚机制，生成对应回滚命令行，当自动生成的回滚命令不满足设备回滚要求时，用户可以对生成的回滚命令进行一次后置处理。

接口功能

对设备自动回滚命令进行自定义。

接口约束

1. 前提条件：一阶段事务设备发生配置下发失败，触发自动回滚。
2. 注意事项：N/A。
3. 使用限制：对应的配置下发命令行满足CLIDriver自动回滚条件。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configRollbackCliPostProcessor(self, aoccontext, request):
    """
    自定义按行后置处理
    :param aoccontext: 上下文环境
    :param request: CLI自定义入参proto对象
    :return: CLI自定义出参proto对象
    """
    pass
```


请求参数描述

表 2-106 CliCustomTransformInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	请求的路径。
data	是	STRING	N/A	N/A	请求的数据。
oper_type	是	OperType	参考表 OperType	N/A	请求的操作类型。
startOffset	是	Integer	>=0	N/A	请求的偏移量。
neId	是	STRING	N/A	N/A	请求的网元ID。
commandLines	否	CommandLine	参考表 CommandLine	N/A	请求的命令行。

表 2-107 OperType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	READ_CFG READ_OPR CREATE MERGE DELETE RPC	N/A	操作类型。

表 2-108 CommandLine

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
index	否	Integer	>=0	N/A	命令行的序号。
commandLine	否	STRING	N/A	N/A	命令行的内容。

表 2-109 CliCustomTransformOutput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	修改后数据。
endOffset	是	Integer	>=-1	N/A	结束偏移量。
commandLines	是	Integer	参考表 CommandLine	N/A	修改后按行数据。

请求和响应样例

Python调用样例：

```
def configRollbackCliPostProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
    # 获取请求Path，根据不同的path做不同业务处理
    req_path = request.path
    if req_path == '/path1':
        out.data = 'no interface aaa'
    return out
```

2.3.5 SND RESTCONF 框架定制

2.3.5.1 RPC Yang 转 Restconf

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义yang转restconf方法来解决rpc操作转换问题。

接口功能

客户可以自定义rpc操作的yang转restconf的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def rpcCustomYangToRestconf(self, aoccontext, request):
    """
    自定义rpc操作的yang数据转restconf的转换逻辑
    :param aoccontext: 上下文环境
    :param request: RESTCONF自定义入参proto对象
    :return: RESTCONF自定义出参proto对象
    """
    pass
```

请求参数描述

表 2-110 RpcRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
rpcQName	是	STRING	N/A	N/A	rpc请求QName。
input	是	STRING	N/A	N/A	rpc请求入参。

表 2-111 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-112 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def rpcCustomYangToRestconf(self, aoccontext, request):
    url = None
    httpMethod = None

    # 获取请求path，根据不同path做不同业务处理
    if request.rpcQName == "/path1":
        url = '/sample/url_1'
        httpMethod = POST

    # 构建返参对象
    requestInfo = RestconfRequestInfo()
    requestInfo.url = url
    requestInfo.httpMethod = httpMethod
    return requestInfo
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.5.2 RPC Restconf 转 Yang

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义restconf转yang方法来解决rpc操作转换问题。

接口功能

客户可以自定义rpc操作的restconf转yang的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def rpcCustomRestconfToYang(self, aoccontext, input):
    """
    自定义rpc操作的Restconf转Yang转换逻辑 restconf:custom-restconf-to-yang 'rpc'
    :param aoccontext: 上下文环境
    :param input: rpc用户自定义RestconfToYang入参
    :return: output 对应的Yang数据
    """
    pass
```

请求参数描述

表 2-113 RpcCustomRestconfToYangInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcRequestInfo	是	RpcRequestInfo	查看表RpcRequestInfo	N/A	rpc请求信息。
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-114 RpcRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcQName	是	STRING	N/A	N/A	rpc请求QName。
input	是	STRING	N/A	N/A	rpc请求入参。

表 2-115 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。

表 2-116 RpcReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
ouput	是	STRING	N/A	N/A	rpc output xml格式报文。

请求和响应样例

Python调用样例：

```
def rpcCustomRestconfToYang(self, aoccontext, input):
    rpcRequestInfo = input.rpcRequestInfo
    output = None
    if rpcRequestInfo.rpcQname == 'testqname':
        output = "<xml></xml>"
    response = RpcResponseInfo()
    response.output = output
    return response
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.5.3 RPC 前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

RPC操作返回响应报文后，在报文传给RestconfDriver之前，对报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行RESTCONF-YANG的默认转换。但在某些特殊情况下，设备返回的RESTCONF报文不能直接用作RestconfDriver的转换入参，需要再进行一次自定义的修改，才可以达到作为RESTCONF-YANG转换入参的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def rpcCustomPreProcess(self, aoccontext, input):
    """
    rpc前置处理 restconf:custom-pre-process 'rpc'
    :param aoccontext: 上下文环境
    :param input: rpc请求相关的信息以及Restconf协议返回的相关信息
    :return: 经过SND包加工后的Restconf返回信息
    """
    pass
```

请求参数描述

表 2-117 RpcCustomPreProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcRequestInfo	是	RpcRequestInfo	查看表RpcRequestInfo	N/A	rpc请求信息。
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-118 RpcRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
rpcQName	是	STRING	N/A	N/A	rpc请求QName。
input	是	STRING	N/A	N/A	rpc请求入参。

表 2-119 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。
status	是	INT	N/A	N/A	响应状态

请求和响应样例

Python调用样例：

```
def rpcCustomPreProcess(self, aoccontext, input):
    requestInfo = input.rpcReqeustInfo
    respopnseInfo = input.restResponseInfo
    responseBody = None
    if requestInfo.rpcQname == "testqname":
        # 自定义处理逻辑
        responseInfo.responseBody = responseBody
    return responseInfo
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.5.4 RPC 后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver生成下发设备的报文后，再对生成的请求报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行YANG-RESTCONF的默认转换。但在某些特殊情况下，RestconfDriver生成的Restconf报文不能满足设备的要求，需要再进行一次自定义的修改，才能达到下发设备的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用样例：

```
def rpcCustomPostProcess(self, aoccontext, input):
    """
    rpc后置处理 restconf:custom-post-process 'rpc'
    :param aoccontext: 上下文环境
    :param input: rpc请求相关的信息以及经过SND框架机制默认转换算法生成的Restconf请求信息
    :return: 经过SND包加工后的Restconf请求信息
    """
    pass
```

请求参数描述

表 2-120 RpcCustomPostProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcRequestInfo	是	RpcRequestInfo	查看表RpcRequestInfo	N/A	rpc请求信息。
restconfRequestInfo	是	RestconfRequestInfo	查看表RestconfRequestInfo	N/A	restconf请求信息。

表 2-121 RpcRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
rpcQName	是	STRING	N/A	N/A	rpc请求QName。
input	是	STRING	N/A	N/A	rpc请求入参。

表 2-122 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-123 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def rpcCustomPostProcess(self, aoccontext, input):
    requestInfo = input.rpcRequestInfo
    restconfRequestInfo = input.restconfRequestInfo
    requestBody = None
    httpMethod = None
    if requstInfo.rpcQname == "testqname":
        # 自定义处理逻辑
        restconfRequestInfo.requestBody = requestBody
```

```
    restconfRequestInfo.httpMethod = httpMethod
return restconfRequest
```

错误码

错误码	错误信息	处理措施
无	无	无

2.3.5.5 READ Yang 转 Restconf

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义yang转restconf方法来解决查询操作转换问题。

接口功能

客户可以自定义查询操作的yang转restconf的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
def readCustomYangToRestconf(self, aoccontext, request):
    """
    自定义read的yang2Restconf转换逻辑 restconf:custom-yang-to-restconf 'read'
    :param aoccontext: 上下文环境
    :param request: read操作请求入参
    :return: 经SND包处理过的read操作YangToRestconf
    """
    pass
```

请求参数描述

表 2-124 ReadRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
path	是	STRING	N/A	N/A	查询路径。
properties	否	Map<STRING,STRING>	N/A	N/A	查询参数。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-125 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-126 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-127 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def readCustomYangToRestconf(self, aoccontext, request):
    restconfRequestInfo = RestconfRequest()
    url = None
    if request.path == "testreadpath":
        # 自定义转换逻辑
        restconfRequestInfo.url = url
    return restconfRequestInfo
```

2.3.5.6 READ Restconf 转 Yang

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义restconf转yang方法来解决查询操作转换问题。

接口功能

客户可以自定义查询操作的restconf转yang的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def readCustomRestconfToYang(self, aoccontext, input):
    """
    自定义read的Restconf2Yang转换逻辑 restconf:custom-restconf-to-yang 'read'
```

```
:param aoccontext: 上下文环境
:param input: read 用户自定义RestconfToYang入参
:return: 经snd处理过后的用户自定义RestconfToYang
"""
pass
```

请求参数描述

表 2-128 ReadCustomRestconfToYangInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
readRequestInfo	是	ReadRequestInfo	查看表ReadRequestInfo	N/A	read请求信息。
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-129 ReadRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	查询路径。
properties	否	Map<STRING,STRING>	N/A	N/A	查询参数。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-130 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-131 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。

表 2-132 ReadReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	是	STRING	N/A	N/A	查询结果xml报文。

请求和响应样例

Python调用样例：

```
def readCustomRestconfToYang(self, aoccontext, input):
    rpcRequestInfo = input.readRequestInfo
    data = None
    if rpcRequestInfo.path == 'testreadpath':
        data = "<xml></xml>"
    response = ReadReponseInfo()
    response.data= data
    return response
```

2.3.5.7 READ 前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

Read操作返回响应报文后，在报文传给RestconfDriver之前，对报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行RESTCONF-YANG的默认转换。但在某些特殊情况下，设备返回的RESTCONF报文不能直接用作RestconfDriver的转换入参，需要再进行一次自定义的修改，才可以达到作为RESTCONF-YANG转换入参的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def readCustomPreProcess(self, aoccontext, input):
    """
    查询前置处理 restconf:custom-pre-process 'read'
    :param aoccontext: 上下文环境
    :param input: read用户自定义前置处理入参
    :return: 经过SND包加工后的Restconf返回信息
    """
    pass
```

请求参数描述

表 2-133 ReadCustomPreProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
readRequestInfo	是	ReadRequestInfo	查看表ReadRequestInfo	N/A	read请求信息。
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-134 ReadRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	查询路径。
properties	否	Map<STRING,STRING>	N/A	N/A	查询参数。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-135 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-136 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。
status	是	INT	N/A	N/A	响应状态

请求和响应样例

Python调用样例：

```
def readCustomPreProcess(self, aoccontext, input):
    readRequestInfo = input.readRequestInfo
    restconfResponse = input.restconfResponse
    responseBody = restconfResponse.responseBody
    if readRequestInfo .path == 'testpath':
        # 自定义处理逻辑
        restconfResponse.responseBody = responseBody
    return restconfResponse
```

2.3.5.8 READ 后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver生成下发设备的报文后，再对生成的请求报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行YANG-RESTCONF的默认转换。但在某些特殊情况下，RestconfDriver生成的Restconf报文不能满足设备的要求，需要再进行一次自定义的修改，才能达到下发设备的要求。

接口约束

1. 前提条件：N/A。

2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def readCustomPostProcess(self, aoccontext, input):
    """
    查询后置处理 restconf:custom-post-process 'read'
    :param aoccontext: 上下文环境
    :param input: 后置处理入参
    :return: 经过SND包加工后的Restconf请求信息
    """
    pass
```

请求参数描述

表 2-137 ReadCustomPostProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
readRequestInfo	是	ReadRequestInfo	查看表ReadRequestInfo	N/A	read请求信息。
restconfRequestInfo	是	RestconfRequestInfo	查看表LogicalDatastoreType	N/A	restconf请求信息。

表 2-138 ReadRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	查询路径。
properties	否	Map<STRING,STRING>	N/A	N/A	查询参数。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-139 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-140 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-141 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def readCustomPostProcess(self, aoccontext, input):
    readReqeustInfo = input.readRequestInfo
    restconfRequestInfo = input.restconfRequestInfo
    url = restconfRequestInfo.url
    httpMethod = restconfRequestInfo.httpMethod
    if readRequestInfo.path == 'testpath':
        # 自定义转换逻辑
        restconfRequestInfo.url = url
        restconfRequestInfo.httpMethod = httpMethod
    return restconfRequestInfo
```

2.3.5.9 CONFIG Yang 转 Restconf

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义yang转restconf方法来解决配置操作转换问题。

接口功能

客户可以自定义配置操作的yang转restconf的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCustomYangToRestconf(self, aoccontext, request):
```

```
"""
自定义dryRun的yang2Restconf转换逻辑 restconf:custom-yang-to-restconf 'merge,replace,create,delete'
:param aoccontext: 上下文环境
:param request: config操作请求入参
:return: 经过SND包加工后的Restconf请求信息
"""
pass
```

请求参数描述

表 2-142 ConfigRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	配置路径。
data	否	STRING	N/A	N/A	配置xml报文。
requestType	是	RequestType	查看表RequestType	N/A	操作类型。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-143 RequestType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	CREATE MERGE REPLACE DELETE	N/A	操作类型。

表 2-144 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-145 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-146 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def configCustomYangToRestconf(self, aoccontext, request):
    url = None
    httpMethod = None
    requestBody = None
    restconfRequestInfo = RestconfRequestInfo()
    if request.path == 'testconfigpath':
        # 自定义转换逻辑
```

```
        restconfRequestInfo.url = url
        restconfRequestInfo.httpMethod = httpMethod
        restconfRequestInfo.requestBody = requestBody
    else:
        # 其他处理逻辑
    return restconfRequestInfo
```

2.3.5.10 CONFIG Restconf 转 Yang

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver不能支持的特殊情况下，使用自定义restconf转yang方法来解决配置操作转换问题。

接口功能

客户可以自定义配置操作的restconf转yang的部分逻辑。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCustomRestconfToYang(self, aoccontext, input):
    """
    自定义dryRun的Restconf2Yang转换逻辑 restconf:custom-restconf-to-yang 'merge,replace,create,delete'
    :param aoccontext: 上下文环境
    :param input: config操作请求入参
    :return: 经过SND包加工后的Restconf返回信息
    """
    pass
```

请求参数描述

表 2-147 ConfigCustomRestconfToYangInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
configRequestInfo	是	ConfigRequestInfo	查看表ConfigRequestInfo	N/A	config请求信息。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-148 ConfigRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	配置路径。
data	否	STRING	N/A	N/A	配置xml报文。
requestType	是	RequestType	查看表RequestType	N/A	操作类型。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-149 RequestType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	CREATE MERGE REPLACE DELETE	N/A	操作类型。

表 2-150 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-151 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。

表 2-152 ConfigResponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	N/A	N/A	N/A	N/A	N/A

请求和响应样例

Python调用样例：

```
def configCustomRestconfToYang(self, aoccontext, input):
    configRequestInfo = input.configRequestInfo
    restconfResponseIno = input.restconfReponseInfo
    responseBody = restconfResponseInfo.responseBody
    configReponseInfo = ConfigReponseInfo()
    if 'testconfigpath' == configRequestInfo.path:
        # 自定义转换逻辑
        configReponseInfo.responseBody = responseBody
    return configReponseInfo
```

2.3.5.11 CONFIG 前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

Config操作返回响应报文后，在报文传给RestconfDriver之前，对报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行RESTCONF-YANG的默认转换。但在某些特殊情况下，设备返回的RESTCONF报文不能直接用作RestconfDriver的转换入参，需要再进行一次自定义的修改，才可以达到作为RESTCONF-YANG转换入参的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCustomPreProcess(self, aoccontext, input):
    """
    dryRun前置处理 restconf:custom-pre-process 'merge,replace,create,delete'
    :param aoccontext: 上下文环境
    :param input: config操作PreProcess入参
    :return: 经过SND包加工后的Restconf返回信息
    """
    pass
```

请求参数描述

表 2-153 ConfigCustomPreProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
configRequestInfo	是	ConfigRequestInfo	查看表ConfigRequestInfo	N/A	read请求信息。
restconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-154 ConfigRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	配置路径。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
data	否	STRING	N/A	N/A	配置xml报文。
requestType	是	RequestType	查看表RequestType	N/A	操作类型。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-155 RequestType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	CREATE MERGE REPLACE DELETE	N/A	操作类型。

表 2-156 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-157 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
status	是	INT	N/A	N/A	响应状态

请求和响应样例

Python调用样例：

```
def configCustomPreProcess(self, aoccontext, input):
    configRequestInfo = input.configRequestInfo
    restconfResponseInfo = input.restconfResponseInfo
    responseBody = restconfResponseInfo.responseBody
    if configRequestInfo.path == 'testconfigpath':
        # 自定义处理逻辑
        restconfResponseInfo.responseBody = responseBody
    return restconfResponseInfo
```

2.3.5.12 CONFIG 后置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

在RestconfDriver生成下发设备的报文后，再对生成的请求报文进行一次处理。

接口功能

RestconfDriver会根据YANG模型进行YANG-RESTCONF的默认转换。但在某些特殊情况下，RestconfDriver生成的Restconf报文不能满足设备的要求，需要再进行一次自定义的修改，才能达到下发设备的要求。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def configCustomPostProcess(self, aoccontext, input):
    """
    dryRun后置处理 restconf:custom-post-process 'merge,replace,create,delete'
    :param aoccontext: 上下文环境
    :param input: config操作PostProcess入参
    :return: 经过SND包加工后的Restconf请求信息
    """
    pass
```

请求参数描述

表 2-158 ConfigCustomPostProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
configRequestInfo	是	ConfigRequestInfo	查看表ConfigRequestInfo	N/A	read请求信息。
restconfRequestInfo	是	RestconfRequestInfo	查看表RestconfRequestInfo	N/A	restconf请求信息。

表 2-159 ConfigRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
neId	是	STRING	N/A	N/A	设备ID。
path	是	STRING	N/A	N/A	配置路径。
data	否	STRING	N/A	N/A	配置xml报文。
requestType	是	RequestType	查看表RequestType	N/A	操作类型。
logicalDatastoreType	是	LogicalDatastoreType	查看表LogicalDatastoreType	N/A	逻辑数据类型。

表 2-160 RequestType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	CREATE MERGE REPLACE DELETE	N/A	操作类型。

表 2-161 LogicalDatastoreType

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	OPERATIONAL CONFIGURATION	N/A	逻辑数据类型。

表 2-162 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-163 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

请求和响应样例

Python调用样例：

```
def configCustomPostProcess(self, aoccontext, input):
    configRequestInfo = input.configReqeustInfo
    restconfRequestInfo = input.restconfRequestInfo
    url = restconfRequestInfo.url
    httpMethod = restconfRequestInfo.httpMethod
    requestBody = restconfRequestInfo.requestBody
    if configRequestInfo.path == 'testconfigpath':
        # 自定义处理逻辑
        restconfRequestInfo.url = url
        restconfRequestInfo.httpMethod = httpMethod
        restconfRequestInfo.requestBody = requestBody
    return restconfRequestInfo
```

2.3.5.13 异常报文前置处理

类型

API-4，设备数据查询接口和操作接口。

典型场景

当设备侧返回异常信息报文时，对返回的异常报文进行前置处理。

接口功能

通常Restconf在获取到报错响应以后，会从响应报文中解析到对应的报错信息，当设备侧返回的报文无法通过默认方法解析到对应的报错信息时，通过对返回报文的前置处理，处理成RestconfDriver可以解析的报文数据结构，以达到在页面展示对应报错信息的目的。

接口约束

1. 前提条件：N/A。
2. 注意事项：N/A。
3. 使用限制：N/A。
4. 接口之间的关联关系：N/A。

调用方法

Python调用接口方法：

```
@abstractmethod
def errorCustomPreProcess(self, aoccontext, input):
    """
    报错报文前置处理 restconf:custom-pre-process 'error'
    :param aoccontext: 上下文环境
    :param input: 报错报文操作PreProcess入参
    :return: 经过SND包加工后的Restconf返回信息
    """
    pass
```

请求参数描述

表 2-164 ErrorCustomPreProcessInput

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
restconfRequestInfo	是	RestconfRequestInfo	查看表RestconfRequestInfo	N/A	restconf请求信息。
RestconfReponseInfo	是	RestconfReponseInfo	查看表RestconfReponseInfo	N/A	restconf响应信息。

表 2-165 RestconfRequestInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
url	是	STRING	N/A	N/A	请求url。
requestBody	是	STRING	N/A	N/A	请求报文。
HttpMethod	是	HttpMethod	查看表HttpMethod	N/A	请求方法。
parameter	否	Map<string,string>	N/A	N/A	GET请求参数。
restfulClient	否	Boolean	N/A	N/A	是否调用restful接口。

表 2-166 HttpMethod

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
N/A	是	枚举	GET POST PUT DELETE PATCH	N/A	请求方法。

表 2-167 RestconfReponseInfo

参数名称（Python）	是否必选	参数类型	参数值域	默认值	参数说明
responseBody	是	STRING	N/A	N/A	响应报文。
status	是	INT	N/A	N/A	响应状态

请求和响应样例

Python调用样例：

```
def errorCustomPreProcess(self, aoccontext, input):
    rpcRequestInfo = input.restconfRequestInfo
    errorMsg= None
    # 自定义处理
    response = RestconfReponseInfo()
    response.responseBody= errorMsg
    return response
```

3 CLI 自定义扩展

3.1 背景

为了使CliDriver最大限度地成为一个通用引擎，而不是一个业务引擎，为了减少不必要的特例代码、兼容性代码，所以开发以下自定义扩展项，可以将YANG与CLI之间的正逆向转换逻辑转移到Specific Ne Driver（以下简称SND）包中实现。用户可以根据不同设备的不同CLI特性，通过编码的方式，定制出其中的转换逻辑，实现CliDriver在解析Cli与YANG功能上的灵活性。

3.2 扩展

3.2.1 cli-custom-transform

范围

container，list，leaf。

功能

由SND包定制被标记cli-custom-transform的container结点、list结点以及leaf结点的正向YangToCli的转换逻辑。此扩展可能有3个参数，可以用逗号进行组合，分别是create, update和delete，参数指定了使此扩展生效的操作类型。如果当前节点有扩展项：

1. 对于create操作
   a. path指向当前节点，那么当前结点的转换逻辑由SND包给出。
   b. path指向父节点，那么父节点其他路径的节点不受影响，当前路径的转换逻辑由SND包给出。
   c. path指向子节点，那么子节点的转换逻辑由SND包给出。
   参考下面**例子**，给出2个例子如下：

- 举例1：现有YANG文件结构c0/c1/c2/l2, c0/c1/l1, c0/c4/l4，标红的c2有create扩展项，path指向c1，路径c0/c4/l3与c1无关，就不会render。如果l1和l2都有值，那么l1的路径因为没有transform注解，会由CLIDriver正常render；而l2会由SND包render。
- 举例2：现有YANG文件结构c0/c1/c2/c3/l3，标红的c2和c3都有create扩展项，path指向l3，那么CLIDriver将从C0开始从顶至下检查，发现C2有transform注解，那么SND中c2路径对应的自定义代码将会执行，c2以下的都不再解析。因为c0和c1是container类型父结点，所以它们不会由CLI生成前缀，而是由c2的自定义逻辑给出。

2. 对于delete操作
   a. path指向父节点，那么没有任何影响。
   b. path指向当前节点或子节点，那么转换逻辑由SND包给出。
3. 对于update操作
   a. 如果当前节点有扩展项，path指向当前节点，那么当前节点的转换逻辑由SND包给出。
   b. 如果当前节点有扩展项，path指向父节点，那么父节点其他路径的节点不受影响，当前路径的转换逻辑由SND包给出。
   c. 如果当前节点有扩展项，path指向子节点，那么子节点的转换逻辑由SND包给出。

例子

```
container c0 {
  container c4 {
    leaf l4 {
      type string;
    }
  }
  container c1 {
    leaf l1 {
      type string;
    }
    container c2 {
    cli:cli-custom-transform 'create,update,delete'
      leaf l2 {
        type string;
      }
      container c3 {
      cli:cli-custom-transform 'create,update,delete'
        leaf l3 {
          type string;
        }
      }
    }
  }
}
```

在上方的YANG文件中，c2和c3 都标记了cli-custom-tranform扩展，c3是c2的子节点，c2是c1的子节点，c0是最顶层节点，c1和c4是兄弟节点。根据扩展参数，操作类型为create、update和delete，并且path末结点为c2/c3的父结点、c2/c3本身以及c2/c3的子节点的各种情况，其正向YangToCli转换逻辑都由SND包负责给出。

假设SND:

```
if path=/c0/c1/c2       return c1 c2custom  # c1前缀由SND负责给出
if path=/c0/c1/c2/c3    return c1 c2 c3custom  # 需给出c1和c2前置container
```

假设path:

```
path=/c0/c1
```

分析：path的末节点为c1，所以c4和l4不受影响。 c1及其所有子结点会进行正向YangToCli转换。l1的路径上没有cli-custom-transform注解，因此l1的转换由CLIDriver给出；c2有cli-custom-transform的注解，那么SND包会执行对应c2的自定义转换逻辑，即return c1 c2custom。c2的逻辑给出后，SND包不再执行c2子节点的转换逻辑，即C3不会再被SND包处理。

正常出参：

```
c1 l1 l1test
c1 c2 l2 l2test
c1 c2 c3 l3 l3test
```

自定义出参：

```
c1 l1 l1test
c1 c2custom
```

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:c1/cli-custom:c2" and request.oper_type == OperType.Value('CREATE'):
        out.data = "c1 c2custom"
    return out
```

上面是SND包的正向自定义转换代码，代码在yangToCli方法里，通过匹配request.path和操作类型request.oper_type拦截正确的请求。out.data存放的是生成的CLI命令。

3.2.2 cli-custom-read

范围

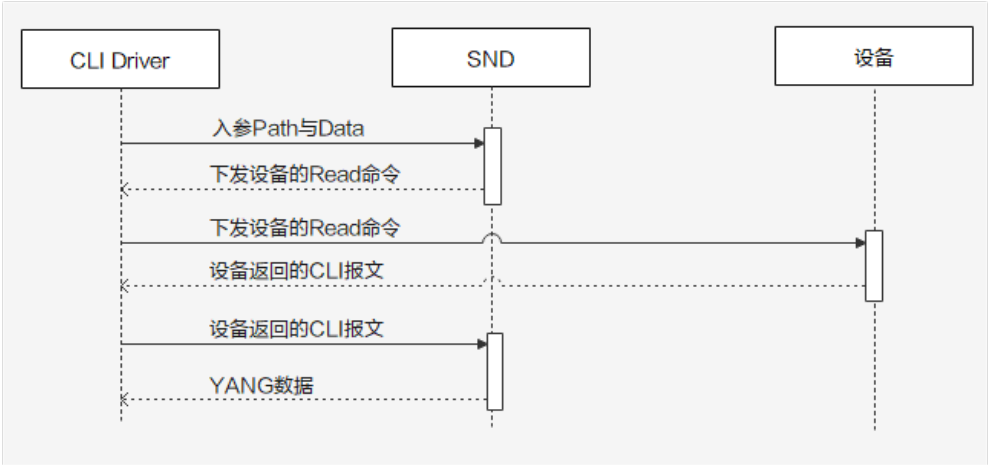
container, list, leaf。

功能

由SND包定制被标记cli-custom-read的节点的逆向CliToYang的转换逻辑。此扩展有三种参数，分别是two-step，one-step和one-step-no-parse，三个参数只能同时出现一个，不能叠加。它们区别如下：

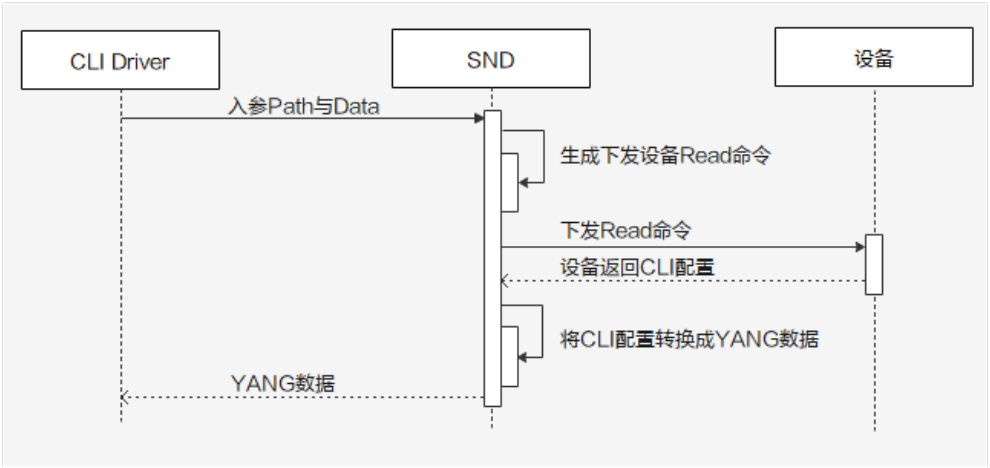
1. **two-step**

   read操作的正向生成命令行与逆向解析设备配置的两个转换逻辑都由SND负责给出，CLIDriver与SND包发生两次交互：CLIDriver调用SND包的yangToCli方法生成下发设备的CLI read命令，由CLIDriver将此命令下发设备，CLIDriver获取设备返回的配置报文后，再调用SND包的cliToYang方法，将配置报文转换成Yang数据，SND包最终将YANG数据返回给CLIDriver。



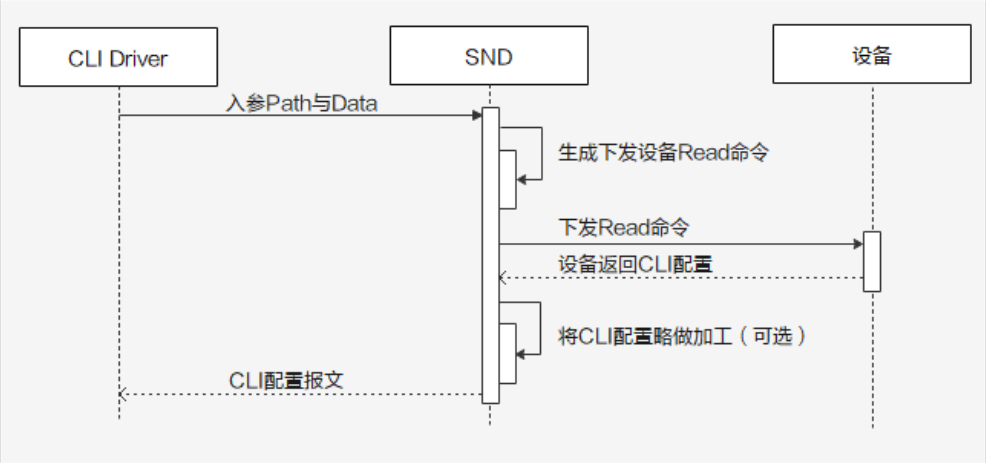
2. **one-step**

   CLIDriver直接向SND包发送read命令的path参数，调用SND包的cliToYang方法，由SND包负责生成下发设备的read命令，SND将命令下发到设备，获取到设备返回的配置报文后，将报文转换成Yang数据，最终返回给CLIDriver。在one-step流程中，CLIDriver与SND包只发生一次交互，只涉及cliToYang一个方法，并且与设备交互的是SND包而不是CLIDriver。



3. **one-step-no-parse**

   与one-step类似，CLIDriver向SND包发送read命令的path参数，调用SND包的cliToYang方法 。但是，当SND包下发命令到设备上，获取设备返回的配置报文后，可以对配置报文略做处理，随后将配置报文直接返回给CLIDriver，由CLIDriver负责CLI到YANG进行转换。SND包只负责下发命令的生成、调用设备、获取并加工返回的配置报文即可。



说明

- cli-custom-read 'one-step'和'one-step-no-parse'只能加在顶层节点上，非顶层节点不生效。
- cliToYang方法在返回out对象时，必须填写endOffset字段，表明入参字符串有多少被消费了，消费之后的offset值是多少。对于one-step与one-step-no-parse，endOffset返回-1；其他情况下，如果全部消费完，可以返回str(request.data).__len__()；如果没有全部消费完，endOffset根据实际消费的字符数加上入参request对象中的startOffset值得出。

例子

1. **two-step**

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

在上方的YANG文件中，router-id leaf节点标记了cli-custom-read‘two-step’自定义扩展，那么对router-id的read操作的正向下发命令的生成操作由SND负责完成，对设备返回的CLI配置报文转换成YANG数据的操作也由SND包负责完成。

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "tellabs:router-id":
        out.data = "show running config | block router-id"
    return out
```

上面是在two-step中，SND包生成自定义read命令的代码。SND包通过request.path匹配到所需路径，生成自定义read的下发设备的命令。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "tellabs:router-id":
        lines = request.data.splitlines()
        if len(lines) > 1:
            router_id_content = str(lines[1]).split()[1]
            out.data = "<router-id xmlns=\"https://huawei.com/tellabs\">" + router_id_content + "</
router-id>"
    out.endOffset = str(request.data).__len__()
    return out
```

上面是在two-step中，SND包cliToYang方法将设备返回配置报文转换成YANG数据的操作。自定义代码通过request.path匹配路径，然后通过request.data获取设备配置报文，处理报文获取到所需router-id的数据，存放在router_id_content变量中，最后将YANG数据组装好，返回给CLIDriver。

2. **one-step**

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

在上方的YANG文件中，router-id leaf节点标记有cli-custom-read 'one-step'，那么对router-id的read操作的所有CLI与YANG的转换逻辑由SND包的cliToYang方法给出。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    proxy = CommandProxy(request.neId, logger)
    proxy.send_command(["enable", "end", "configure terminal"])
    response_data = proxy.send_command(["show running-config | block router-id"], 120)
    response_body = response_data.response
    lines = response_body.splitlines()
    if len(lines) > 1:
        router_id_content = str(lines[1]).split()[1]
        out.data = "<router-id xmlns=\"https://huawei.com/tellabs\">" + router_id_content + "</router-
id>"
    out.endOffset = -1
    return out
```

在SND包中，我们从入参request的neId属性中获取网元ID，然后调用CommandProxy代理类将自定义的read命令下发至设备，获取设备返回的配置报文，并将配置报文转换成YANG数据。CLIDriver调用一次SND包，在cliToYang方法内完成read操作的CLI与YANG的全部自定义转换。

3. **one-step-no-parse**

```
leaf router-id {
    cli:cli-custom-read 'one-step-no-parse';
    type string;
}
```

在上方的YANG文件中，router-id标记有cli-custom-read 'one-step-no-parse'，那么相比one-step，SND包的cliToYang方法只要返回CLI配置报文即可，无需做CLI到YANG的转换操作。

```
def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    proxy = CommandProxy(request.neId, logger)
    proxy.send_command(["enable", "end", "configure terminal"])
    response_data = proxy.send_command(["show running-config | block router-id"], 120)
    response_body = response_data.response
    lines = response_body.splitlines()
    if len(lines) > 1:
        router_id_content = str(lines[1]).split()[1]
        out.data = "router-id" + router_id_content
    out.endOffset = -1
    return out
```

与one-step类似，one-step-no-parse在SND包cliToYang方法中编写自定义代码，通过CommandProxy代理类给设备下发自定义read命令，获取设备返回配置报文后，做适当数据处理，最后返回类似"router-id 1.2.3.4"这样的CLI配置报文给CLIDriver，CLIDriver负责根据YANG模型定义，将此CLI配置报文转换成YANG数据。one-step-no-parse让SND包自定义代码无需做CLI到YANG的转换操作。

3.2.3 cli-custom-processor

范围

顶层container, 顶层list。

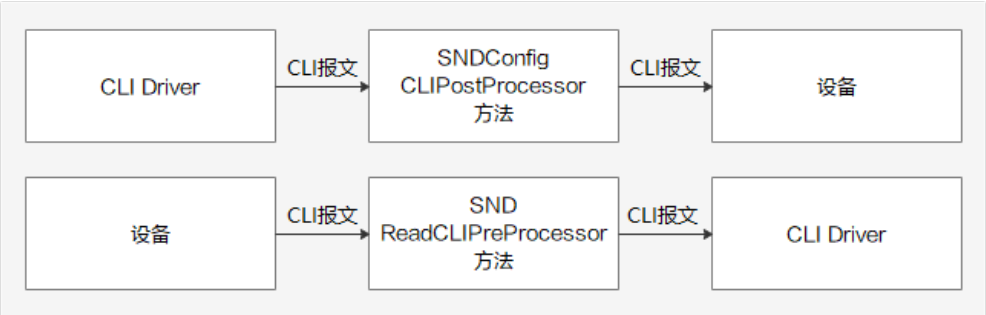
功能

这个扩展是前置后置处理，它有4个参数，分别是pre-read、post-config和post-config-by-line、post-config-rollback，可以用逗号分隔同时使用。

- pre-read就是将设备返回的CLI Config在返回给CLiDriver之前做一次前置自定义处理。一般使用在读的场景。
- post-config就是在CliDriver生成CLI命令后、下发设备前，对CLI做一个后置处理。一般使用在CUD下发场景，偶尔也用在读命令下发场景。
- post-config-by-line是将设备命令行按行后置处理。如果采用这样方式，用户自定代码不允许改变命令的行数和顺序。
- post-config-rollback当CLIDriver默认回滚机制无法满足一阶段设备的回滚特性时，用户可根据实际情况自定义回滚命令。

  流程如下：

  “CLIDriver正向 > CLI > (SND)后置处理 > 设备 > (SND)前置处理 > Config > CLIDriver逆向”。



此注解虽然配在顶层节点，由于父子继承关系，入参path如果指向任何子节点，都可以进入SND包前置后置处理方法中。用户可以针对不同的path及入参数据，采取不同的处理措施。

例子

YANG文件

```
container interfaces {
    cli:cli-custom-processor 'post-config,pre-read';
    list interface {
        container mpls {
            list l2vc {
                key role;
                leaf role {
                    typoe enumeration {
                        enum "primary";
                        enum "secondary";
                    }
                }
            }
        }
    }
}
```

在上方的YANG文件中，顶层节点interfaces由cli:cli-custom-processor 'post-config,pre-read'标记，所有的请求，如果入参path指向interfaces节点或它的子节点，其转换操作都会由SND包进行自定义前置处理或后置处理。当前的自定义需求是：

- 在下发命令时，如果CLI报文中l2vc节点的值包含primary，将primary删除。

- 在设备读取配置时，如果读取的l2vc节点的值为空，需要向CLI报文合适位置添加primary。

1. **pre-read**

```
def readCliPreProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
    cli_str = str(request.data)
    re_mpls = r'.*(undo\s+)?mpls l2vc.*'
    if re.search(re_mpls, cli_str):
        newlines = []
        lines = cli_str.splitlines()
        for line in lines:
            if re.search(re_mpls, line) and 'secondary' not in line:
                line += ' primary'
            newlines.append(line)
        cli_str = '\n'.join(newlines)
    out.data = cli_str
    return out
```

pre-read的自定义代码写在readCliPreProcessor方法中。自定义代码通过正则表达式匹配设备返回的CLI报文。如果l2vc节点的值为空，而且不包含secondary，就将primary添加到行尾。

2. **post-config**

```
def configCliPostProcessor(self, aoccontext, request):
    out = CliCustomTransformOutput()
    cli_str = request.data
    re_mpls = r'.*(undo\s+)?mpls l2vc.+primary.*'
    if re.search(re_mpls, cli_str):
        newlines = []
        lines = cli_str.splitlines()
            for line in lines:
                if re.search(re_mpls, line):
                    line = line.replace('primary', '')
                newlines.append(line)
            cli_str = '\n'.join(newlines)
    out.data = cli_str
    return out
```

post-config的自定义代码写在configCliPostProcessor方法中。用户从request.data获取到需要后置处理的CLI报文，然后通过正则表达式等手段对CLI报文进行处理。如果报文中包含mpls l2vc primary，就将primary替换成空字符串。最后处理完，将新CLI报文赋值给out.data。

3.2.4 cli-custom-rpc

范围

rpc顶层节点。

功能

将rpc的YANG与CLI的正向与逆向转换过程分别委托给SND包中的yangToCli和cliToYang两个方法来实现：

1. 用户调用rpc的input接口，传入input的path与data，CLIDriver调用SND包的yangToCli方法，生成下发设备的命令。
2. CLIDriver下发命令到设备，获取设备的返回的报文。
3. CLIDriver调用SND包中的cliToYang方法，传入output的path以及设备返回的报文。

例子

1. **YANG文件**

```
rpc resetIpStatistics {
    cli:cli-custom-rpc;
    input {
        leaf interface-name {
            type string;
        }
    }
    output {
        leaf response {
            type string;
        }
    }
}
```

在上方的YANG文件中，resetIpStatistics是rpc的顶层节点。cli-custom-rpc只能作用于顶层节点。入参path=/resetIpStatistics/input，调用SND 包的yangToCli方法，生成发向设备的CLI命令。出参path=/resetIpStatistics/output，调用SND包的cliToYang方法，生成YANG数据。

2. **SND包**

```
def yangToCli(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:resetIpStatistics/cli-custom:input":
        out.data = "show running-config | block router-id"
    return out


def cliToYang(self, aoccontext, request):
    out = CliCustomTransformOutput()
    if request.path == "/cli-custom:resetIpStatistics/cli-custom:output":
        out.data = "<resetIpStatistics xmlns=\"https://huawei.com/cli-custom\">" \
                   "<response>hello_world</response>" \
                   "</resetIpStatistics>"
        out.endOffset = str(request.data).__len__()
    return out
```

在上方的SND包代码中，RPC的正向自定义代码写在yangToCli方法里，request.path指向了RPC YANG的input节点，out.data的内容是希望下发设备的CLI命令。RPC的逆向自定义代码写在cliToYang方法里，request.path指向RPC Yang的output节点，out.data对应需要返回的YANG数据。

4 RESTCONF 自定义拓展

4.1 背景

RestconfDriver作为通用引擎，为了减少不必要的特例代码、兼容性代码，开发以下自定义扩展项，可以将YANG与RESTCONF之间的正逆向转换逻辑转移到Specific Ne Driver（以下简称SND）包中实现。用户可以根据不同设备的不同RESTCONF报文特性，通过编码的方式，定制出其中的转换逻辑，实现RestconfDriver在解析RESTCONF与YANG功能上的灵活性。

4.2 拓展

4.2.1 custom-yang-to-restconf

范围

container，list，leaf。

说明

rpc操作只能作用在顶层节点。

功能

由SND包定制被标记custom-yang-to-restconf container结点、list结点以及leaf结点的正向YangToRestconf的转换逻辑。此扩展可能有6个参数，可以用逗号进行组合，分别是create, update, delete, merge，read, rpc 参数指定了使此扩展生效的操作类型。标记为custom-yang-to-restconf节点以及其子节点的相对应操作的yang数据到restconf报文的转换将由SND包给出。

例子

1. **YANG文件**

```
rpc resetIpStatistics {
    restconf:custom-yang-to-restconf 'rpc';
    input {
        leaf interface-name {
            type string;
        }
    }
    output {
        leaf response {
            type string;
        }
    }
}
```

在上方的YANG文件中，resetIpStatistics是rpc的顶层节点。入参path=/resetIpStatistics/input，调用SND 包的对应操作的YangToRestconf方法，生成发向设备的restconf请求报文。

2. **SND包**

参考**2.3.5.1 RPC Yang转Restconf**中rpcCustomYangToRestconf方法的实现。

4.2.2 custom-restconf-to-yang

范围

container，list，leaf。

功能

由SND包定制被标记custom-restconf-to-yang container结点、list结点以及leaf结点的逆向RestconfToYang的转换逻辑。此扩展可能有6个参数，可以用逗号进行组合，分别是create, update, delete, merge，read, rpc 参数指定了使此扩展生效的操作类型。标记为custom-restconf-to-yang节点以及其子节点的相对应操作的restconf报文到yang数据的转换将由SND包给出。

例子

1. **YANG文件**

```
container l3vpn-svcs {
    restconf:custom-restconf-to-yang 'read';
    description
      "Service information set.";
    uses gp-l3vpn-svcs;
  }
}
```

在上方的YANG文件中，l3vpn-svcs是一个顶层节点。对该节点以及其所有子节点的read操作，都会调用SND包中的readCustomRestconfToYang的方法，以完成对该节点以及其子节点的所有read操作的restconf报文到yang数据的转换。

2. **SND包**

参考**2.3.5.6 READ Restconf转Yang**中readCustomRestconfToYang方法的实现。

4.2.3 custom-post-process

范围

container，list，leaf。

功能

由SND包对被标记custom-post-process的container结点，leaf结点，list结点的由RestconfDriver生成的request请求信息做对应的后置处理，包括对requestBody的修改，HTTPMethod的修改以及是否调用restful接口等。此扩展可能有6个参数，可以用逗号进行组合，分别是create, update, delete, merge，read, rpc 参数指定了使此扩展生效的操作类型。

例子

1. **YANG文件**

```
grouping gp-l3vpn-svcs {
    description
        "Service information set.";
    list l3vpn-svc {
        restconf:custom-post-process 'create';
        key "vpn-id";
        description
            "Service information set.";
        uses gp-l3vpn-svc;
    }
}
```

在上方的YANG文件中，对l3vpn-svc list的新增操作，都会调用SND包中的configCustomPostProcess方法，用来自定义完成对RestconfDriver生成的request信息的后置处理。

2. **SND包**

参考**2.3.5.12 CONFIG 后置处理**中configCustomPostProcess方法的实现。

4.2.4 custom-pre-process

范围

container，list，leaf。

功能

由SND包对被标记custom-pre-process的container结点，leaf结点，list结点的由设备端返回的Restconf响应信息做对应的前置处理，处理成满足RestconfDriver操作的response报文后再交由RestconfDriver处理。此扩展可能有7个参数，可以用逗号进行组合，分别是create, update, delete, merge，read, rpc,error 参数指定了使此扩展生效的操作类型。

例子

1. **YANG文件**

```
grouping gp-l3vpn-svcs {
    description
        "Service information set.";
    list l3vpn-svc {
```

```
        restconf:custom-pre-process 'create';
        key "vpn-id";
        description
            "Service information set.";
        uses gp-l3vpn-svc;
    }
}
```

在上方的YANG文件中，对l3vpn-svc list的新增操作，都会调用SND包中的configCustomPreProcess方法，用来自定义完成由设备侧返回的响应信息的前置处理。。

2. **SND包**

   参考**2.3.5.11 CONFIG 前置处理**中configCustomPreProcess方法的实现。