数通网络开放可编程
V100R020C00
开发指南

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

目 录

1 预备知识

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

1.1 了解编程语言和协议

在进行开放可编程开发前，需要掌握下表所述编程语言和协议。

表 1-1 开放可编程编程语言和协议

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

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

1.2 了解软件包结构

从系统中导出的软件包结构统一如下。

- 支持Python语言软件包。

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

Python开发语言

package.zip

|---package -------文件夹名字，和压缩包同名。

| |---pkg.json 【必填项】包描述文件。

| |---Python Python开发的代码目录。

| |--- 用户开发的Python

| |---yang YANG模型存放目录。

| |---template 模板存放目录。

| |---resources 资源存放目录。

| |---doc 存放相关文档，可以没有。

1.3 了解日志记录

1.3.1 查看日志

日志路径

日志路径：/opt/oss/log/NCECOMMONE/OpenEMTestService/log/users/${user}.log

{$user}为用户登陆名 如：/opt/oss/log/NCECOMMONE/OpenEMTestService/log/users/admin.log

日志格式

[transId：事务ID前7位] [module：模块名] [pkgName：包名] [serviceName：业务名][deviceId：设备ID]日志内容

运行态与设计态日志记录方式

- 运行态日志：用户日志只在ExtendedRTService的对应路径下记录。用户日志所在的绝对路径：/opt/oss/log/NCE/ExtendedPkgRTService。

  用户日志文件名：oss.{$group_name}.{$pkg_name$pkg_version}.trace，例如：oss.group1.helloworld1.0.0.trace。

  - {$group_name}为用户开发的三方包所在的group名字。
  - {pkg_namepkg_version}为用户开发的三方包的名字+三方包的版本号。
- 设计态日志：在debug模式下，用户日志同时记录在ExtendedRTService、OpenEMService的指定路径下。

1.3.2 设置日志级别

支持用户自定义设置用户日志级别。

日志配置文件 logger.conf 介绍

logger.conf内容：

[logger_user]

level=INFO

说明

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

日志配置文件路径

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

说明

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

{$group_name}为用户开发的三方包所在的group名。

{$package_name}为用户开发的三方包名。

2 开发流程

软件包开发流程如下图所示。

图 2-1 软件包开发流程

3 开发准备

本节介绍开发软件包前所需的配套资源。

编号	名称	用途说明	获取方式
1	aoc_api-2.0.0-py3-noneany.whl	开放可编程SDK是由华为提供的一套基于Python的编程接口，里面包含需要继承的父类NcsService以及所有可能调用到的接口。	登录数通网络开放可编程社区：**https://devzone.huawei.com/test/aoc/resDownload.html**，下载SDK文件。
2	yang-offline-util.zip	开放可编程YANG模型离线验证工具。	登录数通网络开放可编程社区：**https://devzone.huawei.com/test/aoc/resDownload.html**，下载验证工具。
3	开放可编程典型场景样例包	提供典型场景代码样例包，方便用户定制。	登录数通网络开放可编程社区：**https://devzone.huawei.com/test/aoc/resDownload.html**，获取对应的代码样例包。
4	开放可编程mini版本软件包	提供开放可编程轻量化的软件包，用户可以在下载安装后在本地电脑执行设备和业务相关配置操作。	登录数通网络开放可编程社区：**https://devzone.huawei.com/test/aoc/resDownload.html**，获取软件包。

4 开发环境部署

开发环境的部署操作以Windows 10系统为例进行介绍。

4.1 安装 Python 解析器

下载并安装

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

验证

在安装成功后，可以按下述方式进行安装后验证。

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

步骤2 在当前提示符下输入“python”，回显应如下图所示。

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

----结束

手动配置环境变量

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

步骤1 在“此电脑”上单击右键菜单“属性”，打开“系统”界面。

步骤2 在导航栏单击“高级系统设置”，打开“系统属性”界面。

步骤3 在“高级”页签页面，单击“环境变量”，打开“环境变量”界面。

步骤4 在用户变量区域，双击“Path”变量，弹出“编辑环境变量”界面。

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

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

步骤7 单击“确定”。

步骤8 按照验证步骤能正常回显，则说明环境变量配置成功。

----结束

4.2 安装开放可编程 SDK

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

说明

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

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

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

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

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

----结束

4.3 安装集成开发环境

下载并安装

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

配置 Python 解析器

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

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

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

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

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

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

配置完成后，PyCharm即可正常使用。

----结束

4.4 安装 Gpg4Win 及生成签名用密钥

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

下载安装

您可以访问**Gpg4Win官网**，下载并安装最新版的Gpg4Win。

生成签名文件

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

步骤1 在开始菜单中，单击“Kleopatra”运行签名工具。

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

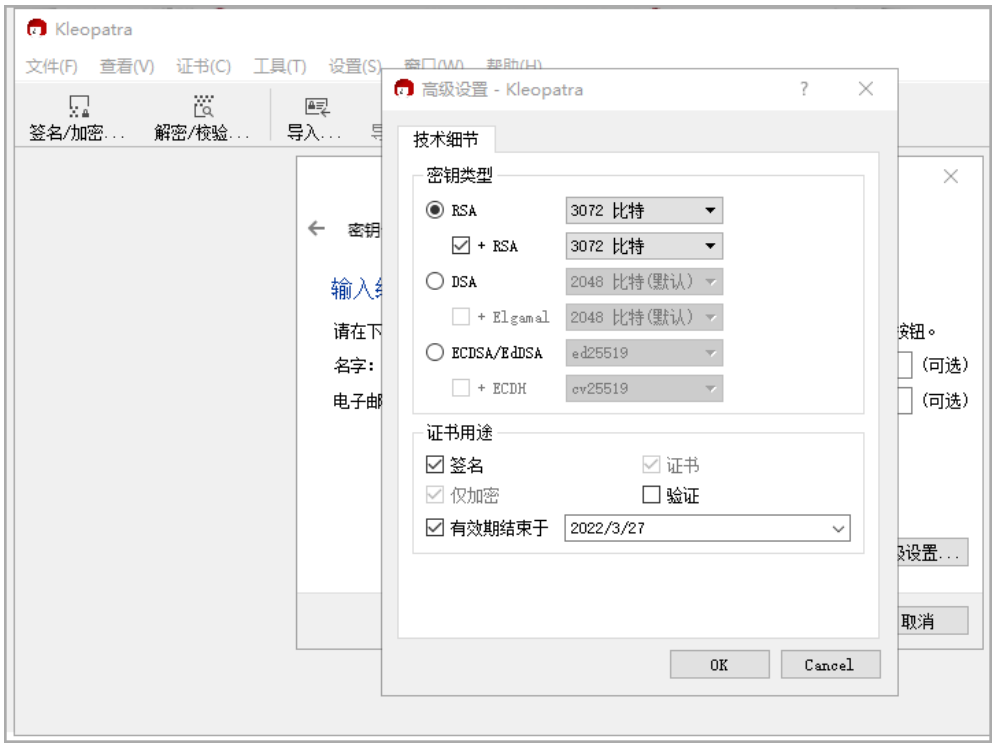
步骤3 在“输入细节”中输入“名字”和“电子邮件”。

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

说明

在“证书用途”区域，务必不要勾选“验证”。

图 4-1 高级设置



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

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

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

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

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

----结束

4.5 安装开放可编程 mini 软件包

安装**开放可编程mini软件包**及启动服务步骤如下所述：

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

步骤2 解压软件包到当前路径下。

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

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

----结束

5 开发业务插件包

业务插件包可以实现三部分功能：

- EasyMap：将网络层业务分解成网元层业务，实现设备纳管后由网络层向设备直接下发业务。
- RPC：用户自定义功能。标准的NETCONF/YANG配置类模型不能满足要求，用户可以灵活自定义功能，如一些查询类操作。
- Discover：业务还原，将网元层业务重新组织成网络层业务，即EasyMap的逆过程。设备纳管前已存在网元层业务，那么纳管后需要将网元层业务还原到网络层。

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

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

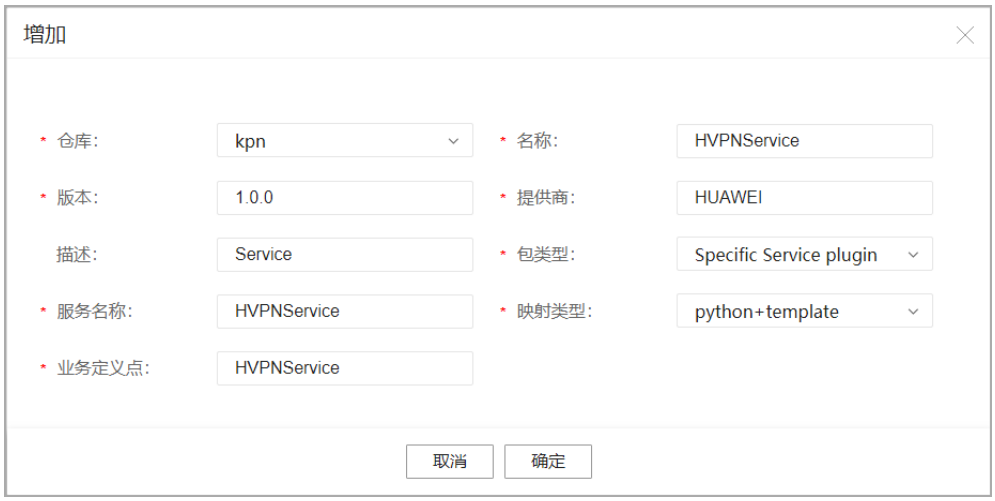
本章节以一个简单示例，介绍如何通过下载的默认SSP包，开发一个业务插件包。该例子中业务是创建一个VPN，并将子接口绑定到VPN上。

5.1 创建业务包模板

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

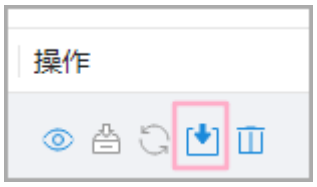
步骤2 并在弹出的“增加”对话框中设置软件包的相关属性。

请根据软件包类型确定包类型的值：开发业务插件包（简称业务包），包类型选择“Specific Service plugin”。



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

步骤4 单击这条软件包记录“操作”栏中的 下载该软件包到本地。



----结束

5.2 开发业务包

5.2.1 编辑软件包配置文件

步骤1 打开PyCharm社区版，单击“Create New Project”，进入“New Project”对话框。展开“Project Interpreter”区域，确认配置无误后，单击“Create”。

步骤2 将下载下来的软件包解压到当前工程所在路径。

步骤3 在集成开发环境左侧导航树中，双击打开pkg.json文件。

步骤4 按照下表修改对应参数的值。

表 5-1 pkg 文件参数表

参数名称	含义	必选/可选
name	软件包名称	必选
version	软件包版本号。仅支持三段式xxxx.xxxx.xxxx，每个x取值为[0到9]，每段最多支持4位。	必选
description	对包的说明	可选

参数名称	含义	必选/可选
package-type	包类型	必选
group	包分组信息	可选
producer	软件包的提供商。目前对于 snd 包和 gnd 包，只支持该字段为"huawei"的包。	必选
augment-isolation	模型范围标识，布尔值，只能为 true 或 false。	可选
nce-min-versions	兼容的开放可编程系统最小版本信息，不填表示兼容所有版本。	可选
package-dependencies	对其他三方包的依赖。如果配置，则必须包含name（必选）， version（必选）字段。	可选
snd-id	"package-type"为 snd包才有值，id信息区分不同snd包，不同名的包不能使用相同的snd-id。	可选
devices	snd, gnd包特有值，驱动匹配的设备信息。用于软件包的描述，不用于设备纳管。如果配置，则必须包含vendor（必选），device-type（必选），device-version（必选）字段。	可选
service-name	ssp包特有值，服务名，不同名的包不能使用相同的service-name。	可选
hooks	回调映射信息。相同包里hooks组合唯一，但在不同包里可以相同。snd包时，snd-id组合要唯一，不同名的包不能使用相同的snd-id。如果配置，则需包含type（必选）， key（必选）， java-class-name（可选），Python-class-name（可选）， groovy-class-name（可选）， template（可选）字段。EasyMap，type设置为mapping；RPC，type设置为service-rpc；Discover，type设置为discover。多项功能组合时可以设置多个hooks。	必选

示例如下：

```
{
   "name": "HVPNService",
   "version": "1.0.0",
   "description": "5G",
   "package-type": "ssp",
   "producer": "HUAWEI",
   "service-name":"HVPNService",
   "nce-min-versions":[
      "2.0.0"
   ],
   "hooks": [
     {
       "type": "mapping",
       "key": "HVPNService",
       "Python-class-name": "HVPNService.HVPNService.AocNcs_servicepoint"
     }
```



```
  ]
}
```

----结束

5.2.2 开发业务 YANG 模型

在集成开发环境左侧导航树中，双击打开yang目录下的HVPNService.yang文件，编写YANG模型。yang文件名称根据pkg.json中"service-name"值命名。当前系统仅支持在application的根节点下挂接list和存在类型container的业务定义点。在父节点上定义业务定义点，则在子节点上不允许再定义业务定义点。如下面示例中在list bng_binding已打上app:application-definition "example"，则不能在container master_info上再打业务定义点。

EasyMap 和 Discover

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

如当前业务示例，VPN的name由UI输入，子接口的节点name、mtu、des由UI输入，需要在YANG模型中进行定义。

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

    import huawei-ac-applications {
        prefix app;
    }

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

    description
        "The module for HVPNService example.";

    revision 2018-12-09 {
        description "Initial revision.";
    }

    augment "/app:applications"{
        list hvpnService {
            /*app:application-definition 表示定义一个系统可识别的servicepoint，
            这里的service-point必须要和pkg.json文件hooks参数中的key字段值一致。*/
            app:application-definition "HVPNService";
            key "instanceName";
            leaf instanceName {
                type string;
            }
            list deviceList {
                key "deviceName";
                leaf deviceName {
                    type leafref {
                        path "/device:nes/device:ne/device:operate-name";
                    }
                    mandatory true;
                }
            }
            leaf name {
                type string;
            }

            leaf des {
                type string;
```

```
        }

        leaf mtu {
            type int32 {
              range "46..9600";
            }
        }

        leaf vpn_instance_name {
            type string;
        }
    }
  }
}
```

RPC

当标准的NETCONF/YANG配置类模型不能满足要求，用户可以自定义RPC功能，定义功能的输入和输出。RPC功能主要用于定义不需要保存数据的一次性操作，如ping命令。

```
rpc service-rpc
{
    description "get devices by device group id RPC";

    input
    {
        leaf ne-id
        {
description "device ne id.";
            type string;
        }
    }

    output
    {
    list aoc-ecs-output{
        leaf result {
            type string;
        }
        list output-attribute-map {
             leaf name{
                 type string;
              }
             leaf value{
                 type string;
              }
           }
        }
    }
}
```

加密处理

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

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

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

表 5-2 security.xml 文件配置说明

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

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

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

自定义模型权限

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

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

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

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

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

5.2.3 开发 Mapping 代码

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

批量模板操作和批量设备配置操作，影响范围大，建议用户在开发代码时仔细评估风险。

用户可以通过UserException方式抛出自己的异常，并可以在界面上查看异常信息。

EasyMap

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

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

# AocNcs_servicepoint继承NcsService并复写ncs_map方法
class AocNcs_servicepoint(NcsService):
  # 复写ncs_map方法，编写获取设备创建接口需要的ifNumb，ParentName参数。
  def ncs_map(self, request, aoccontext=None, template=None):
    self.getIfNumb(request)
    self.getParentName(request)

    """
    render函数实现网络层业务向网元层的映射，request.xmldictnode为封装的北向数据，
    HVPNService/servicepoint.j2是网元层模板。
    """
    return self.render('HVPNService/servicepoint.j2', request.xmldictnode)


  # 定义函数getIfNumb用于从创建的子接口中获取ifNumb并更新到参数字典中。
  def getIfNumb(self, request):
    # 如果要获取YANG模型相关的节点数据，可以通过x.y.z这样的方式便捷操作模型数据。
    sub_if_name = request.xmldictnode.hvpnService.name
    pointIndex = sub_if_name.find('.')
    ifNumb = sub_if_name[pointIndex + 1:]
    request.xmldictnode.update({"ifNumber": ifNumb})


  # 定义函数getParentName用于从创建的子接口中获取出ParentName并更新到字典中。
  def getParentName(self, request):
    sub_if_name = request.xmldictnode.hvpnService.name
    pointIndex = sub_if_name.find('.')
    parentName = sub_if_name[0:pointIndex]
    request.xmldictnode.update({"parentName": parentName})
```

RPC

当用户自定义RPC功能时，需要继承NcsService并复写ncs_rpc方法，按照业务需求实现自定义功能的逻辑，最终返回处理结果。

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

class AocNcsexample(NcsService):
    def ncs_rpc(self, asrequest, arg1, arg2):
        ...
        return result
```

Discover

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

```
# 必选，导入NcsService类，NcsService是aoc_api提供的供SSP包继承的父类。
from aoc.ncs.ncsservice import NcsService
from aoc.ncs.ncs_model_pb2.discover_pb2 import DiscoverOutput
from aoc.sys import datastore

class AocNcsAaaService(NcsService):
    def ncs_map(self, request, aoccontext=None, template=None):
        self.logger.info(request.xml)
        result = self.render('template_Aaa.j2', request.xmldictnode)
        self.logger.info(result)
        return result


    def discover(self, discoverInput, aoc_context):
        self.logger.info(discoverInput)
        self.logger.info(aoc_context)

        # 从输入获取设备信息
        deviceId = discoverInput.deviceInfo[0].deviceId
        deviceName = discoverInput.deviceInfo[0].deviceName

        # 构造env
        env = dict()
        env['device'] = deviceName

        # 读取设备RDB
        path = '/huawei-ac-nes:inventory-cfg/nes/ne/' + deviceId + '/huawei-aaa:aaa/lam/users'
        users = datastore.read_datastore_rdb(aoc_context, path)
        root = self.xmltodictnode(users)
        result = DiscoverOutput()

        # 解析RDB结果
        for user in root.users.user:
            self.parse_rdb_user(env, user, result)
        self.logger.info(result)
        return result


    def parse_rdb_user(self, env, user, result):
        aaamin = result.serviceConfig.add()
        aaamin.servicePath = '/huawei-ac-applications:applications/aaamini:aaamini/' + user.userName
        env['username'] = user.userName
        env['password'] = user.password
        aaamin.serviceData = self.render('aaamin-discover.j2', env)
```

5.2.4 开发 Jinja2 模板

开发Jinja2模板的思路主要有两个：

- 设备已纳管时，通过网元管理获取对应操作NETCONF报文，再推导出业务Jinja2模板。
- 根据YANG模型直接推导编写Jinja2模板。开发难度相对大一些，适合有经验的开发者。

下面通过一个实例，介绍通过NETCONF报文推导出业务的Jinja2模板的具体操作过程：

1. 通过“网元配置”界面在指定设备上创建子接口，创建VPN实例。
2. 然后通过开放可编程的Dry-Run能力查看待下发的NETCONF报文。
3. 最后基于获取到的NETCONF报文推导出业务的Jinja2模板，甚至是YANG模型。

具体步骤如下：

步骤1 在主菜单中选择“设备配置 > 设备配置”。进入“设备管理”界面，在需要配置的设备上单击“编辑”。

步骤2 在当前设备的“设备管理”左树中单击“huawei-ifm”，打开接口相关的全局操作配置页面。在“Interface”区域单击添加，然后输入接口名称并单击“创建”。

步骤3 在当前设备的“设备管理”左树中单击“huawei-l3vpn”，打开vpn相关的操作配置页面。在“l3vpnInstance”区域单击增加，然后输入vrfName名称并单击“创建”。

步骤4 单击“全局操作”区域上方的“试运行”，查看并复制待下发设备的NETCONF报文。

步骤5 粘贴复制好的NETCONF报文到软件包“template”目录下的“servicepoint.j2”文件中。

步骤6 根据业务需要，修改Jinja模板。如果参数来自于北向输入，则将参数改为变量；如果参数采取固定值，则直接写为常量即可。

步骤7 最终成型的Jinja模板示例如下。

SSP分解的配置涉及到关键的配置，例如主接口相关。如果在业务实例删除后，删除掉关键配置会有业务安全问题。用户可以基于业务逻辑打上“no_delete”或“no_delete_nested”属性，标记该数据是不会随着业务实例的删除而删除掉。

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

```
<inventory-cfg xmlns="urn:huawei:yang:huawei-ac-nes">
  <nes>
    {% for neName in hvpnService.deviceList %}
    <ne>
      <neid>{{neName.deviceName| to_ne_id}}</neid>
      <ifm xmlns="http://www.huawei.com/netconf/vrp/huawei-ifm">
        <interfaces>
          <interface>
            <ifName>{{hvpnService.name}}</ifName>
            <ifNumber>{{ifNumber}}</ifNumber>
            <ifMtu>{{hvpnService.mtu}}</ifMtu>
            <ifParentIfName>{{parentName}}</ifParentIfName>
            <ifClass>subInterface</ifClass>
            <ifDescr>{{hvpnService.des}}</ifDescr>
          </interface>
        </interfaces>
      </ifm>
      {%- if hvpnService.vpn_instance_name %}
      <l3vpn xmlns="http://www.huawei.com/netconf/vrp/huawei-l3vpn">
```



```
          <l3vpncomm>
            <l3vpnInstances>
              <l3vpnInstance>
                <vrfName>{{hvpnService.vpn_instance_name}}</vrfName>
                <l3vpnIfs>
                  <l3vpnIf>
                    <ifName>{{hvpnService.name}}</ifName>
                  </l3vpnIf>
                </l3vpnIfs>
              </l3vpnInstance>
            </l3vpnInstances>
          </l3vpncomm>
        </l3vpn>
        {%- endif %}
      </ne>
      {% endfor %}
    </nes>
</inventory-cfg>
```

----结束

5.3 验证业务包

5.3.1 验证 YANG 文件

在完成软件包的开发后，可以使用YANG模型校验工具校验软件包中YANG文件的合法性。

步骤1 解压“yang-offline-util.zip”。

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

步骤3 运行如下命令验证YANG文件的正确性。

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

当运行结果为空时，表明YANG文件格式正确。

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

----结束

5.3.2 执行单元测试

步骤1 使用“yang-offline-util.zip”工具，根据YANG模型生成空的NETCONF报文。请执行如下命令。

C:\Users\demo\yang-offline-util>java -jar yang-offline-util.jar generateSubtree .

当运行结果为空时，表明“subtree.xml”文件成功生成。

步骤2 打开“subtree.xml”文件，并在<application>标签下各标签中填上适当的值。

```
<hvpnService xmlns="http://example.com/HVPNService">
    <deviceList>
        <deviceName>NE1</deviceName>
    </deviceList>
    <deviceList>
```



```
        <deviceName>NE2</deviceName>
    </deviceList>
    <name>GigabitEthernet3/0/0.801</name>
    <des>des</des>
    <mtu>1500</mtu>
    <vpn_instance_name>LTE_OMC</vpn_instance_name>
</hvpnService>
```

步骤3 双击打开test目录下的.py文件，将前面步骤修改好的<application>标签内的代码拷贝粘贴到如下<hvpnService>标签位置。示例如下。

```
import unittest
import sys
sys.path.insert(0, "../../Python")
from HVPNService.HVPNService import AocNcs_servicepoint

class Test(unittest.TestCase):
    xml = '''

    # 下面hvpnService标签的代码根据实际情况替换
    <hvpnService xmlns="http://example.com/HVPNService">
        <deviceList>
            <deviceName>NE1</deviceName>
        </deviceList>
        <deviceList>
            <deviceName>NE2</deviceName>
        </deviceList>
        <name>GigabitEthernet3/0/0.801</name>
        <des>des</des>
        <mtu>1500</mtu>
        <vpn_instance_name>LTE_OMC</vpn_instance_name>
    </hvpnService>
    '''


    def test_case1(self):
        result = AocNcs_servicepoint().ncs_map_test(self.xml)
        print(result)

if __name__ == "__main__":
    unittest.main()
```

步骤4 运行该测试代码，可看到生成的报文，检查输出是否正确。

回显提示“Process finished with exit code 0”，表明单元测试成功。否则请仔细检查Jinija模板文件和业务YANG模型文件中各属性是否一一对应。

----结束

5.4 编译业务包

按照如下步骤编译软件包。

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

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

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

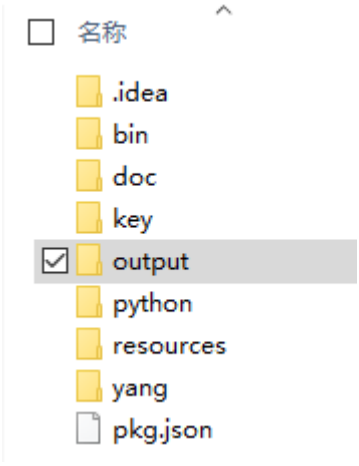
步骤3 运行软件包路径下“bin”目录中的“makeFile.bat”文件开始编译软件包。

在命令行中运行如下命令：

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


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

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




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

----结束

问题定位

问题：编译软件包时出错，提示“NO JAVA_HOME”，怎么处理?

回答：需要安装JDK软件，可以从官网下载并安装（推荐JDK1.8）。

6 导入并安装软件包

开发完成软件包后，将软件包导入开放可编程环境并安装好，方便进行在线调测或使用。

6.1 登录系统

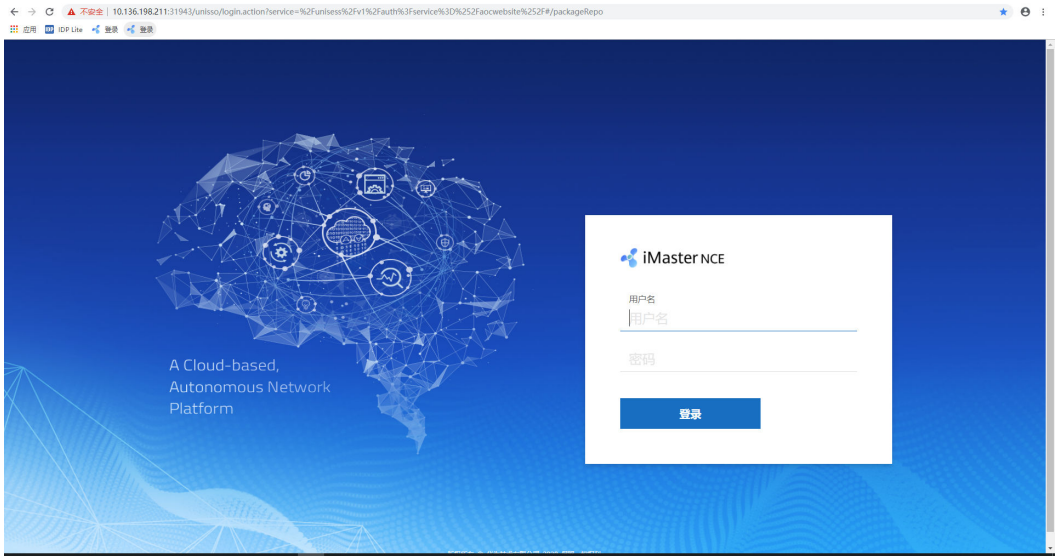
开放可编程提供了两种部署方式：一种是集成到NCE-IP，作为其中一个App（业务开放可编程）发布，随NCE-IP统一部署；一种是独立软件包发布，开放可编程Mini软件包单独安装。本节描述了如何通过浏览器登录系统。目前支持的浏览器列表如下：

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

登录业务开放可编程 App

步骤1 登录NCE运维面。在浏览器地址栏中输入“https://运维面IP地址:31943”，按Enter进入登录界面。输入“用户名”和“密码”，单击“登录”。

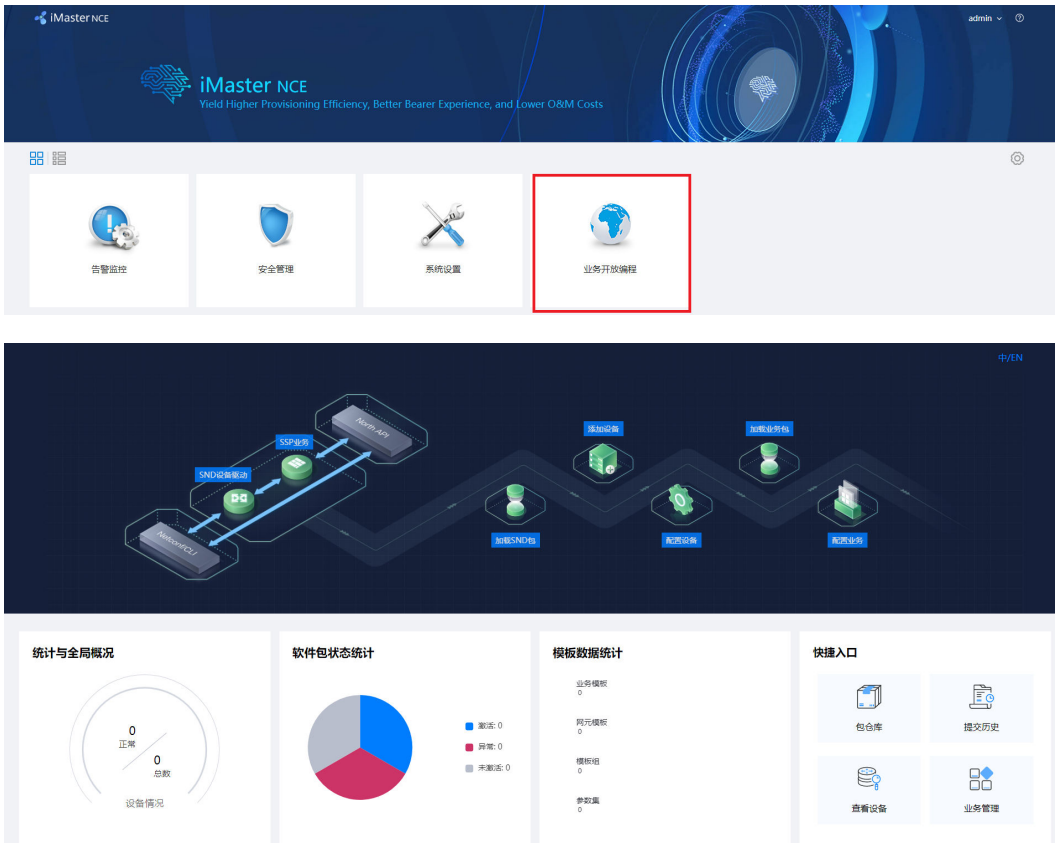
图 6-1 登录 NCE 界面



说明

- 首次登录系统需要修改密码，请妥善保存修改后的密码。为了提高系统安全性，建议用户定时修改密码，以防密码被暴力破解等安全风险。
- 运维面IP地址是指Common_Service节点配置的客户端登录IP地址。当Common_Service节点为集群部署时，此处的IP为Common_Service界点集群的浮动地址。当Common_Service节点为单实例时，此处的IP为Common_Service节点的客户端登录IP地址。

步骤2 登录系统后，选择“业务开放可编程”App，进入“业务开放可编程”首页。



步骤3 在首页根据实际使用场景单击对应的快捷入口或者单击任意快捷入口进入系统主菜单。

----结束

登录开放可编程 Mini 系统

步骤1 登录开发者社区的“资源下载”页签，下载开放可编程Mini软件包（AOCmini_V100R020C00.zip）。

步骤2 解压开放可编程Mini软件包，双击start.bat启动服务，弹出窗口。

```
C:\Users\swx944510\Desktop\AOCmini\envs\Product-AOCService\controller>..\..\..\rtsp\tomcat\bin
\catalina.bat start
Using CATALINA_BASE:   "C:\Users\swx944510\Desktop\AOCmini/envs/Product-AOCService/"
Using CATALINA_HOME:   "C:\Users\swx944510\Desktop\AOCmini/rtsp/tomcat"
Using CATALINA_TMPDIR: "C:\Users\swx944510\Desktop\AOCmini/envs/Product-AOCService/\temp"
Using JRE_HOME:        "C:\Users\swx944510\Desktop\AOCmini/rtsp/jdk/"
Using CLASSPATH:       "C:\Users\swx944510\Desktop\AOCmini/rtsp/tomcat\bin\bootstrap.jar;C:\Users
\swx944510\Desktop\AOCmini/rtsp/tomcat\bin\tomcat-juli.jar"

=========================================
AOCmini is starting, please wait a moment.
```

步骤3 等待系统启动完成。等待3分钟后，系统启动完成。

```
2020-09-10 16:10:28 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:33 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:39 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:43 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:48 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:53 Console message: AOCmini is starting, progress: 95.70%
2020-09-10 16:10:58 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:03 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:08 Console message: AOCmini is starting, progress: 97.68%
2020-09-10 16:11:13 Console message: AOCmini started successfully, please visit https://127.0.0.1:32018/
aocwebsite/ in browser.
```

步骤4 登录界面。在浏览器地址栏中输入“https://127.0.0.1:32018/aocwebsite”，按Enter进入开放可编程首页。

图 6-2 开放可编程首页

步骤5 在首页根据实际使用场景单击对应的快捷入口或者单击任意快捷入口进入系统主菜单。

----结束

6.2 导入并激活软件包

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

图 6-3 软件包导入

步骤2 在弹出的界面中，选择需要导入的软件包和签名文件。

图 6-4 软件包导入选择对话框

步骤3 选择完成，单击“上传”。

图 6-5 单击“上传”

导入完成后，在“包管理”界面可以看到导入的软件包。

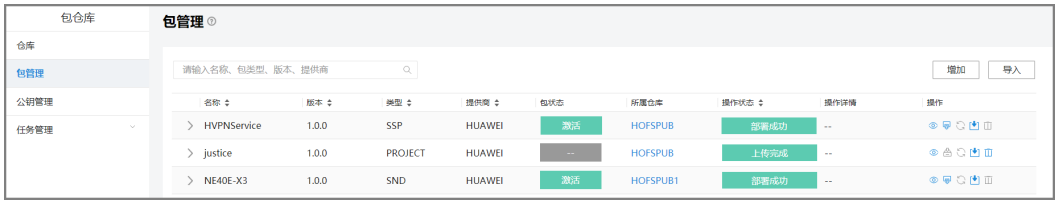
图 6-6 软件包导入成功

步骤4 在主菜单中选择“包仓库”，从左侧导航栏选择进入“包管理”界面，单击。

图 6-7 软件包安装

步骤5 软件包安装需要一定时间，请耐心等待。如果软件包状态变为激活，则说明安装成功。如果为其他信息，则根据详情列失败信息排除故障后重新安装。

图 6-8 软件包安装成功



----结束

7 软件包升级说明

兼容升级指升级前后软件包中的Yang模型发生的变更都是兼容的。不兼容升级指升级前后软件包中的Yang模型发生的变更存在不兼容的项。

当前支持的兼容/不兼容升级规格如下表所示。

表 7-1 兼容/不兼容升级规格

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

schema node（container, list, leaf, leaf-list, anyxml, anydata, choice, case, rpc, action, input, output, notification）	新增可选schema node	新YANG接口中出现了原Yang接口中没有的可选schema node	兼容	修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { type string; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	新增必选schema node	新YANG接口中出现了原Yang接口中没有的必选schema node（如key、mandatory）	不兼容	修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type string;
				}
				leaf name { mandatory true; type string; } }

	删除schema node	原YANG接口中的schema node，在新Yang接口中无法找到	不兼容	修改前 container topology-config { leaf id { type string; } leaf name { type string; } } 修改后 container topology-config { leaf id { type string; } }

typedef,feature	新增	新增了一个typedef/feature定义	兼容	修改前：无
				修改后：
				typedef my-type {
				status deprecated;
				type int32;
				}

	删除	删除了一个typedef/feature定义	不兼容	修改前： typedef my-type { status deprecated; type int32; } 修改后：无

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

数据类型(type)	整形类型(int8、int16、int32、int64、uint8、uint16、uint32和uint64)之间的变更，且变更后的范围扩大	比如，int8变更为int32，变更后范围扩大，属于兼容变更。	兼容	修改前：
				leaf id {
				type uint32;
				}
				修改后：
				leaf id {
				type uint64; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	数据类型变更（除上述场景）或类型不变范围缩小	1. 整形类型之间的变更，但变更后范围缩小（变更后范围不能完全包含变更前的范围，都属于范围缩小，如int8变更为uint16）。
		2. 非整形类型之间的变更及整形非整形之间的变更。（比如，string变更为leafref，uint变更为string）
		3. 数据类型不变，但范围缩小（变更后范围不能完全包含变更前的范围，都属于范围缩小，比如，string类型，变更前范围是0..20, 变更后范围是1..30）
			不兼容	举例1：
				修改前
				container topology-config {
				leaf id {
				type string;
				}
				}
				修改后
				container topology-config {
				leaf id {
				type uint32; } }
				举例2：
				修改前
				typedef interface-ref { type unit8; }
				修改后
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

数量范围	数量范围扩大	同一element的数量范围发生了兼容变化：min-elements变小或max-elements变大	兼容	修改前 leaf-list id { type string; min-elements 2 max-elements 5 } 修改后 leaf-list id { type string; min-elements 1 max-elements 6 }}
	数量范围缩小	同一element的数量范围发生了不兼容变化：min-elements变大或max-elements变小	不兼容	修改前 leaf-list id { type string; min-elements 1 max-elements 6 } 修改后 leaf-list id { type string; min-elements 2 max-elements 5 }}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
取值范围	取值范围扩大	同一属性的取值范围发生了兼容变化：range 变大	兼容	修改前 type uint8 { range "0..60"; } 修改后 type uint8 { range "0..63"; }
	取值范围缩小	同一属性的取值范围发生了不兼容变化：range 变小	不兼容	修改前 type uint8 { range "0..64"; } 修改后 type uint8 { range "0..63"; }
长度范围	长度范围扩大	同一属性的长度范围发生了兼容变化：length 变大	兼容	修改前 type string { length "0..64"; } 修改后 type string { range "0..70"; }
	长度范围缩小	同一属性的长度范围发生了不兼容变化：length 变小	不兼容	修改前 type string { length "0..64"; } 修改后 type string { range "0..50"; }
	长度范围标签从有到无	同一属性的长度范围标签删除	兼容	修改前 type string { length "0..64"; } 修改后 type string { }

	长度范围标签从无到有	同一属性新增长度范围标签	不兼容	修改前
				type string {
				}
				修改后
				type string {
				range "0..50"; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	固定长度变化	属性长度采用固定长度（length）约束时，当固定长度发生变化时，无论变大还是变小，都属于不兼容变化	不兼容	固定长度当作上下限一样的范围长度。 固定长度变为可变长度，可能是业务设计方案发生了变化，所以定义为不兼容变更。
	固定长度标签从有到无	同一属性的固定长度标签删除	不兼容	
	固定长度标签从无到有	同一属性新增固定长度标签	不兼容	

枚举	枚举项新增	同一属性的枚举值中，出现了原来不存在的枚举项	兼容	修改前
				type enumeration {
				enum aaa;
				}
				修改后
				type enumeration {
				enum aaa;
				enum bbb; }

	枚举项删除	同一属性的枚举值中，原枚举项在新的YANG接口中无法找到	不兼容	修改前 type enumeration { enum aaa; enum bbb; } 修改后 type enumeration { enum aaa; }

	枚举项修改	同一枚举的枚举项发生了变化	不兼容	修改前
				type enumeration {
				enum aaa;
				}
				修改后
				type enumeration {
				enum bbb; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	枚举值新增	同一属性的枚举值中，出现了原来不存在的枚举值	不兼容	兼容变更：
				修改前
				type enumeration {
				enum one;
				}
				修改后
				type enumeration {
				enum one {
				value 0; } }
				不兼容变更：
				修改前 type enumeration { enum one; } 修改后 type enumeration { enum one { value 1; } }

	枚举值删除	同一属性的枚举值中，原枚举值在新的YANG接口中无法找到	不兼容	兼容变更
				修改前
				type enumeration {
				enum aaa {
				value 0;
				}
				enum bbb { value 1; } }
				修改后 type enumeration { enum aaa { value 0; } enum bbb; }
				不兼容变更
				修改前 type enumeration { enum aaa { value 0; } enum bbb { value 3; } }
				修改后 type enumeration { enum aaa { value 0; } enum bbb; }

	枚举值变化	同一枚举的值发生了变化	不兼容	修改前
				type enumeration {
				enum aaa {
				value 0; } }
				修改后
				type enumeration { enum aaa { value 1; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
默认值	默认值变更	同一属性的默认值发生了变化	不兼容	修改前 leaf spfInitInterval { type "uint32"; default 50; } 修改后 leaf spfInitInterval { type "uint32"; default 40; }
	默认值标签从有到无	同一属性的默认值标签删除	不兼容	修改前 leaf spfInitInterval { type "uint32"; default 50; } 修改后 leaf spfInitInterval { type "uint32"; }

	默认值标签从无到有	同一属性新增默认值标签	不兼容	修改前
				leaf spfInitInterval {
				type "uint32";
				}
				修改后
				leaf spfInitInterval {
				type "uint32";
				default 40; }

正则表达式	正则表达式变化	同一属性的正则表达式发生了变化	不兼容	修改前 type string { pattern "[0-9\.]*"; } 修改后 type string { pattern "[0-9]*"; }
	正则表达式标签从有到无或删除一个或多个正则表达式	同一属性的正则表达式标签删除，表示约束范围放大	兼容	修改前 type string { pattern "[0-9\.]*"; } 修改后 type string { }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	正则表达式标签从无到有或增加一个或多个正则表达式	同一属性新增正则表达式标签，表示约束范围缩小	不兼容	修改前
				type string {
				}
				修改后
				type string {
				pattern "[0-9]*"; }

	正则表达式pattern的子语句modifier从有到无或从无到有	同一个正则表达式pattern，其子语句modifier从有到无或从无到有	不兼容	修改前：
				type string {
				length "1..max";
				pattern ’[a-zA-Z_][a-zA-Z0-9\-_.]*’;
				pattern ’[xX][mM][lL].*’ {
				modifier invert-match;
				}
				}
				修改后：
				type string {
				length "1..max";
				pattern ’[a-zA-Z_][a-zA-Z0-9\-_.]*’;
				pattern ’[xX][mM][lL].*’;
				}

key约束	key约束删除	同一list的key约束删除	不兼容	修改前 list link { key "id name"; uses link-config; } 修改后 list link { key "id"; uses link-config; }
	key约束新增	同一list的key约束新增	不兼容	修改前 list link { key "id"; uses link-config; } 修改后 list link { key "id name"; uses link-config; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	key约束修改	同一list的key约束修改	不兼容	修改前 list link { key "id"; uses link-config; } 修改后 list link { key "name"; uses link-config; }
	key顺序变化	同一list的key顺序变化	不兼容	修改前 list link { key "protocol ip"; leaf protocol { type string; } leaf ip { type string; } } 修改后 list link { key "ip protocol"; leaf ip { type string; } leaf protocol { type string; } }
mandatory约束	mandatory约束删除	同一属性的mandatory约束由true变为false	兼容	修改前 leaf name { mandatory true; type string; } 修改后 leaf name { mandatory false; type string; }
	mandatory约束新增	同一属性的mandatory约束由false变为true	不兼容	修改前 leaf name { mandatory false; type string; } 修改后 leaf name { mandatory true; type string; }

		同一对象的新增属性的mandatory约束为true	不兼容	修改前
				leaf name {
				type string;
				}
				修改后
				leaf name {
				mandatory true; type string; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

status约束	status约束使用	同一属性的status约束由obsolete变为current或者deprecated;
		同一属性的status约束由deprecated变为current;
			兼容	修改前:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status obsolete;
				description "Matching mode of nodes.";
				}
				修改后:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}

	status约束废弃	同一属性的status约束由current变为deprecated或者obsolete;
		同一属性的status约束由deprecated变成obsolete;
			不兼容	修改前:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status current;
				description "Matching mode of nodes.";
				}
				修改后:
				leaf matchMode {
				type rtpMatchMode;
				mandatory true;
				status deprecated;
				description "Matching mode of nodes.";
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
unique约束	unique约束删除	同一属性的约束删除	兼容	修改前 list server { … unique "ip port"; … } 修改后 list server { … … }

	unique约束新增	同一属性的约束新增	不兼容	修改前
				list server {
				…
				…
				}
				修改后
				list server {
				…
				unique "ip port"; … }

	unique约束修改	同一对象的unique属性个数增加	兼容	修改前
				list server {
				…
				unique "ip";
				…
				}
				修改后
				list server {
				…
				unique "ip port"; … }

		同一对象的unique属性个数减少	不兼容	修改前
				list server {
				…
				unique "ip port";
				…
				}
				修改后
				list server {
				…
				unique "ip"; … }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
		同一对象的unique属性变更	不兼容	修改前 list server { ... unique "ip port"; ... } 修改后 list server { ... unique "ip port-b"; ... }
config约束	config约束新增	同一属性的config约束（包括继承得到的场景）由true变为false	不兼容	修改前 container network-topology-oper { config true; ... } 修改后 container network-topology-oper { config false; ... }
	config约束删除	同一属性的config约束（包括继承得到的场景）由false变为true	兼容	修改前 container network-topology-oper { config false; ... } 修改后 container network-topology-oper { config true; ... }

bits约束	bits的position值增加	同一位域属性的bits的position增加	不兼容	不兼容变更：
				修改前
				type bits {
				bit disable-nagle;
				}
				修改后
				type bits {
				bit disable-nagle {
				position 1; } }
				兼容变更：
				修改前
				type bits { bit disable-nagle; }
				修改后
				type bits { bit disable-nagle { position 0; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	bits的position值删除	同一位域属性的bits的position删除	不兼容	不兼容变更：
				修改前
				type bits {
				bit disable-nagle {
				position 1; } }
				修改后 type bits { bit disable-nagle; }
				兼容变更：
				修改前
				type bits { bit disable-nagle { position 0; } }
				修改后
				type bits { bit disable-nagle; }

	bits的position值修改	同一位域属性的bits的position修改	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0; } }
				修改后
				type bits { bit disable-nagle{ position 1; } }

	新增bit项	同一位域属性的bits的新增bit项	兼容	修改前
				type bits {
				bit disable-nagle {
				position 0; } }
				修改后
				type bits { bit disable-nagle { position 0; }; bit auto-sense-speed { position 1; } }

	bit项删除	同一属性的bit项删除	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed { position 1; } } 修改后
				type bits { bit disable-nagle { position 0; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	bit项修改	同一属性新增bit项修改	不兼容	修改前
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed {
				position 1;
				}
				}
				修改后
				type bits {
				bit disable-nagle {
				position 0;
				}
				bit auto-sense-speed-xxx
				{ position 1; } }

私有扩展语法support-filter	support-filter从true改为false	某个非key leaf节点的support-filter属性从true改为false（不定义默认也为false）	不兼容	修改前： leaf flag { type string; ext:support-filter true; } 修改后： leaf flag { type string; }
	support-filter从false改为true	某个非key leaf节点的support-filter属性从false（不定义默认也为false）改为true	兼容	修改前： leaf flag { type string; } 修改后： leaf flag { type string; ext:support-filter true; }
私有扩展语法value-meaning，item，meaning	value-meaning中的item和meaning对应关系变化	某个leaf节点的类型定义中，value-meaning属性中的item和meaning对应关系变化	不兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning IGMP; } } } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	value-meaning从无到有或某条item和meaning从无到有	某个leaf节点的类型定义中，value-meaning属性整体增加，或者增加某条item和meaning	兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } }
	value-meaning从有到无或某条item和meaning从有到无	某个leaf节点的类型定义中，value-meaning属性整体删除，或者删除某条item和meaning	不兼容	修改前： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } ext:item 1 { ext:meaning ICMP; } } } } 修改后： leaf abc { type uint8 { ext:value-meaning { ext:item 0 { ext:meaning IP; } } } }
私有扩展语法case-sensitivity	case-sensitivity从有到无	某个leaf节点的case-sensitivity属性取值删除，或从非lower-and-upper改为lower-and-upper（不定义默认也为lower-and-upper）	不兼容	修改前： leaf def { type string; ext:case-sensitivity upper2lower; } 修改后： leaf def { type string; }
	case-sensitivity从无到有	某个leaf节点的case-sensitivity属性取值新增，或从lower-and-upper（不定义默认也为lower-and-upper）改为非lower-and-upper	不兼容	修改前： leaf def { type string; } 修改后： leaf def { type string; ext:case-sensitivity upper2lower; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	case-sensitivity属性取值变更	某个leaf节点的case-sensitivity属性取值变更	不兼容	修改前： leaf def { type string; ext:case-sensitivity lower2upper; } 修改后： leaf def { type string; ext:case-sensitivity upper2lower; }

私有扩展语法value-range	value-range属性范围扩大	可输入的数值范围value-range扩大	兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "0..128"; } }

	value-range属性范围缩小	可输入的数值范围value-range缩小	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; ext:value-range "10..63"; } }

	value-range属性从无到有	增加可输入的数值范围value-range属性	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				}
				}
				修改后：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	value-range属性从有到无	删除可输入的数值范围value-range属性	不兼容	修改前：
				leaf dscpValues {
				type pub-type: id-range {
				pattern "xxx";
				ext:value-range "0..63"; } }
				修改后：
				leaf dscpValues { type pub-type: id-range { pattern "xxx"; } }

私有扩展语法masklen，bit，position	masklen属性范围扩大	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中masklen范围扩大	兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

	masklen属性范围缩小	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中masklen范围缩小	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen 3; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	原有的bit与position对应关系变化	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中原有的bit与position对应关系变化	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } } 修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Fri { position 1; } ext:bit Tue { position 2; } ... } }

	某条bit和position从无到有	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中新增某条bit和position的对应关系。	兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ... } }
				修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	某条bit和position从有到无	某个leaf节点类型为自定义类型bits8/bits16/bits32/bits64，类型定义中删除某条bit和position的对应关系。	不兼容	修改前：
				leaf dayOfWeek{
				type pub-type:bits8 {
				ext:masklen 7;
				ext:bit Sun {
				position 0;
				}
				ext:bit Mon { position 1; } ext:bit Tue { position 2; } ... } }
				修改后：
				leaf dayOfWeek{ type pub-type:bits8 { ext:masklen7; ext:bit Sun { position 0; } ext:bit Mon { position 1; } ... } }

私有扩展语法task-name	task-name从无到有	同一模块的task-name标签从无到有	兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } }
	task-name从有到无	同一模块的task-name标签从有到无	不兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … container aaa { … } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	task-name变更	同一模块的task-name属性值变更	不兼容	修改前： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaa"; container aaa { … } } 修改后： module huawei-aaa { namespace "urn:huawei:yang:huawei-aaa"; prefix aaa; … ext:task-name "aaaom"; container aaa { … } }
私有扩展语法node-ref	增加node-ref	同一个rpc增加node-ref	兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }
	删除node-ref	同一个rpc删除node-ref。一个rpc可能有多个node-ref，包括删除其中一个或所有node-ref	不兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } }
	node-ref变更	同一个node-ref属性的值发生变更。	不兼容	修改前： rpc mm { ext:node-ref "/aa/bbs/bb"; input { leaf name { type string; mandatory true; } } } 修改后： rpc mm { ext:node-ref "/aa/ccs/cc"; input { leaf name { type string; mandatory true; } } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
私有扩展语法operation-exclude	operation-exclude新增	某一个节点新增operation-exclude属性，或operation-exclude属性值增加	不兼容	修改前： container aaa { leaf bbb { type uint32; } } 修改后： container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	operation-exclude删除	某一个节点删除operation-exclude属性，或operation-exclude属性值删除	兼容	修改前： container aaa { leaf bbb { ext:operation-exclude update|create; type uint32; } } 修改后： container aaa { leaf bbb { ext:operation-exclude update; type uint32; } }
	operation-exclude变更	包括：子语句when条件变更；子语句filter过滤条件变更。	不兼容	修改前： list aaa { ext:operation-exclude create|delete { ext:filter "name = '_public_'"; } ... } 修改后： list aaa { ext:operation-exclude create|delete { ext:filter "name = '_default_'"; } ... }
私有扩展语法dynamic-default	无默认值变更为dynamic-default	某一个叶子节点从无默认值变更成dynamic-default	不兼容	修改前： container aaa { leaf bbb { type uint32; } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
	dynamic-default变更为无默认值	某一个叶子节点从dynamic-default变更为无默认值	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	静态默认值变更为dynamic-default	某一个叶子节点从静态默认值（default属性）变更为dynamic-default	不兼容	修改前： container aaa { leaf bbb { type uint32; default 12; } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
	dynamic-default变更为静态默认值	某一个叶子节点从dynamic-default变更为静态默认值（default属性）	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; default 12; } }
	dynamic-default变更	dynamic-default属性的default-value取值变更或default-value的子语句when条件发生变更	不兼容	修改前： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘true’”; } } } } 修改后： container aaa { leaf bbb { type uint32; ext:dynamic-default { ext:default-value "../bbb * 3" { when “../ccc = ‘false’”; } } } }
私有扩展语法generated-by	generated-by新增、修改、删除		不兼容	

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
私有扩展语法 refine-ext	refine-ext中新增扩展 operation-exclude	某一个节点的refine-ext中新增扩展 operation-exclude	不兼容	
	refine-ext中删除扩展 operation-exclude	某一个节点的refine-ext中删除扩展 operation-exclude	兼容	
	refine-ext中 operation-exclude发生变更	某一个节点的refine-ext中的 operation-exclude发生变更	不兼容	
	refine-ext中 generated-by新增、删除、变更	某一个节点的refine-ext中的 generated-by发生变更	不兼容	
description标签	description 标签变更	同一属性description标签变更（包括私有语法中的description标签变更）	兼容	description "xxx";
leafref约束	新增leafref	新增了leafref关系，进行了关联限制约束	不兼容	修改前 typedef interface-ref { type unit8; } 修改后 typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	删除leafref	删除了leafref关系，放开了关联限制约束。	兼容	修改前 typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } } 修改后 typedef interface-ref { type string; }

	同一字段的path变化	leafref中同一对象关联的path变化，对关联限制约束进行了修改	不兼容	修改前
				typedef interface-ref {
				type leafref {
				path "/if:interfaces/if:interface/if:ip"; } } 修改后
				typedef interface-ref { type leafref { path "/if:interfaces/if:interface/if:name"; } }

	require-instance从true改为false	leafref的子语句require-instance默认为true，由true改为false	兼容	修改前：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}
				修改后：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	require-instance从false改为true	leafref的子语句require-instance默认为true，由false改为true	不兼容	修改前：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				require-instance false;
				}
				}
				修改后：
				leaf ip {
				type leafref {
				path "/if:interfaces/if:interface/if:ip";
				}
				}

decimal64	fraction-digits变更	decimal64类型的子语句fraction-digits变更，比如从2变更为3	不兼容	修改前：
				typedef my-decimal {
				type decimal64 {
				fraction-digits 2;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}
				修改后：
				typedef my-decimal {
				type decimal64 {
				fraction-digits 3;
				range "1 .. 3.14 | 10 | 20..max";
				}
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

units标签	units标签值变更	同一属性的units标签值发生了变化	不兼容	修改前
				leaf interval {
				type uint16;
				default 30;
				units minutes;
				}
				修改后
				leaf interval {
				type uint16;
				default 30;
				units seconds; }

	units标签从有到无	同一属性的units标签删除	兼容	修改前 leaf interval { type uint16; default 30; units minutes; } 修改后 leaf interval { type uint16; default 30; }

	units标签从无到有	同一属性新增units标签	不兼容	修改前
				leaf interval {
				type uint16;
				default 30;
				}
				修改后
				leaf interval {
				type uint16;
				default 30;
				units minutes; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
reference标签	reference变更	同一属性的reference发生了变化	兼容	修改前 typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... } 修改后 typedef uri { type string; reference "RFC 5040: Uniform Resource Identifier (URI): Generic Syntax"; ... }

	reference标签删除	同一属性的reference标签删除，表示约束范围放大	兼容	修改前
				typedef uri {
				type string;
				reference
				"RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
				...
				}
				修改后
				typedef uri {
				type string; }

	reference标签新增	同一属性新增reference标签，表示约束范围缩小	兼容	修改前 typedef uri { type string; } 修改后 typedef uri { type string; reference "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax"; ... }
when/must	when/must变更	同一属性的when/must发生了变化	不兼容	修改前 leaf isFlowKey { when "../../setId = 2"; } 修改后 leaf isFlowKey { when "setName = 3"; }
	when/must标签从有到无	同一属性的when/must标签删除，表示约束范围放大	兼容	修改前 leaf isFlowKey { when "../../setId = 2"; } 修改后 leaf isFlowKey { ⋯... }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	when/must标签从无到有	同一属性新增when/must标签，表示约束范围缩小	不兼容	修改前
				leaf isFlowKey {
				…...
				}
				修改后
				leaf isFlowKey {
				when "setName = 3"; }

	must条件的error-message变更，从有到无，从无到有	must条件的子语句error-message变更，从有到无，从无到有	兼容	修改前：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-message "An ethernet MTU must be 1500";
				}
				修改后：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

	must条件的error-app-tag变更，从有到无，从无到有	must条件的子语句error-app-tag变更，从有到无，从无到有	兼容	修改前：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)" {
				error-app-tag "must-violation";
				}
				修改后：
				must "ifType != 'ethernet' or " +
				"(ifType = 'ethernet' and ifMTU = 1500)";

if-feature标签	if-feature标签值变更	同一属性的if-feature标签值发生了变化	不兼容	修改前 container location { if-feature has-gps; leaf longitude { mandatory true; ... } } 修改后 container location { if-feature has-gps-xxx; leaf longitude { mandatory true; ... } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例
	if-feature标签从有到无	同一属性的if-feature标签删除	兼容	修改前 container location { if-feature has-gps; leaf longitude { mandatory true; ... } } 修改后 container location { leaf longitude { mandatory true; ... } }

	if-feature标签从无到有	同一属性新增if-feature标签	不兼容	修改前
				container location {
				leaf longitude {
				mandatory true;
				...
				}
				}
				修改后
				container location {
				if-feature has-gps; leaf longitude { mandatory true; ... } }

submodule标签	submodule标签值变更	同一模块的submodule标签值发生了变化	不兼容	修改前： submodule huawei-ac-ne-staticrt-staticrtbase 修改后： submodule huawei-ac-ne-staticrt-staticrtbase-xxx
	submodule标签新增	同一模块新增submodule标签	兼容	
	submodule标签删除	同一模块删除submodule标签	不兼容	

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

belongs-to	belongs-to变更、新增、删除	同一个子模块的belongs-to属性变更	不兼容	修改前：
				submodule huawei-qos-cbqos {
				belongs-to huawei-qos {
				prefix qos;
				}
				修改后：
				submodule huawei-qos-cbqos {
				belongs-to huawei-acl {
				prefix qos;
				}

module	module标签值变更	同一模块的module标签值发生了变化	不兼容	修改前 module huawei-ac-ne-staticrt 修改后 module huawei-ac-ne-staticrt-xxx
prefix	prefix声明变更	同一模块的prefix声明发生了变化	不兼容	修改前 prefix "ac-ne-ifm"; 修改后 prefix "ac-ne-ifm-xxx";
yang-version	yang-version变更	当前yang-version有两个取值，如下： yang-version 1; yang-version 1.1; 当不显式指定时，yang-version的默认值为1. 当yang-version从1变更为1.1或者从1.1变更为1，都属于不兼容变更。	不兼容	修改前： yang-version 1.1; 修改后： 没有显式指定yang-version

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

organization/ contact/ revision	organization/ contact/ revision的变更，从有到无，从无到有	organization/ contact/ revision的变更，从有到无，从无到有	兼容	示例：
				organization
				"Huawei Technologies Co., Ltd.";
				contact
				"Huawei Industrial Base
				Bantian, Longgang
				Shenzhen 518129
				People's Republic of China
				Website: http://www.huawei.com
				Email: support@huawei.com";
				revision 2018-07-02 {
				description "
				Initial version.";
				}

case	Case新增	同一节点中出现了原来不存在的Case	兼容	修改前 choice interface-type { case a { leaf ethernet { ... } } } 修改后 choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } }
	Case删除	同一节点中删除一个Case	不兼容	修改前 choice interface-type { case a { leaf ethernet { ... } } case b { leaf loopback { ...} } } 修改后 choice interface-type { case a { leaf ethernet { ... } } }
namespace	namespace变更	namespace标签值变更	不兼容	修改前 namespace "urn:huawei:yang:huawei-ac-ne-ifm"; 修改后 namespace "urn:huawei:yang:huawei-ac-ne-ifm-xxx";

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

notification携带的参数	元素新增	同一个notification中新增参数，只能从后面追加，不能从中间插入新参数	兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				}
				修改后
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status {
				type types:admin-status;
				}
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	元素删除	同一个notification的参数删除	不兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				修改后
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	元素变更	同一个notification的同一参数变更	不兼容	修改前
				notification node-down{
				leaf id {
				type leafref {
				path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id";
				}
				}
				leaf admin-status { type types:admin-status; } }
				修改后
				notification node-down{ leaf id { type leafref { path "/net-topology:network-topology-oper/net-topology:topologies/net-topology:topology/net-topology:nodes/net-topology:node/net-topology:id"; } } leaf admin-status-xxx { type types:admin-status; } }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

identity	identity新增	新增identity定义	兼容	修改前
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				修改后
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	identity删除	删除identity定义	不兼容	修改后
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius {
				base authentication-method;
				description
				"Indicates user authentication using RADIUS.";
				}
				identity local-users { base authentication-method; description "Indicates password-based authentication of locally configured users."; } 修改前
				identity authentication-method { description "Base identity for user authentication methods."; } identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	identity变更	identity定义变更	不兼容	修改后
				identity authentication-method {
				description
				"Base identity for user authentication methods.";
				}
				identity radius { base authentication-method; description "Indicates user authentication using RADIUS."; }
				修改前
				identity authentication-method { description "Base identity for user authentication methods."; }
				identity radius-xxx { base authentication-method; description "Indicates user authentication using RADIUS."; }

base	base新增	同一个identity下，新增一个base语句。	兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	base删除	同一个identity下，删除一个base语句。	不兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}

	base变更	同一个identity下，base发生变更。	不兼容	修改前：
				identity des {
				base "crypto:crypto-alg";
				description "DES crypto algorithm.";
				}
				修改后：
				identity des {
				base "crypto:symmetric-key";
				description "DES crypto algorithm.";
				}

extension	extension新增	新增extension定义	兼容	修改前
				extension item {
				argument value;
				}
				修改后
				extension item {
				argument value;
				}
				extension index { argument value; }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	extension删除	删除extension定义	不兼容	修改前
				extension item {
				argument value;
				}
				extension index { argument value; }
				修改后
				extension item { argument value; }

	extension变更	extension定义变更	不兼容	修改前
				extension item {
				argument value;
				}
				修改后
				extension item-xxx {
				argument value;
				}

argument	argument变更	extension的子语句argument变更，argument后指定的是参数名称	不兼容	修改前：
				extension c-define {
				argument index;
				}
				修改后：
				extension c-define {
				argument value;
				}

	argument从有到无，从无到有	extension的子语句argument从有到无，说明该extension原来有参数改为无参数
		extension的子语句argument从无到有，说明该extension原来无参数改为有参数
			不兼容	修改前：
				extension c-define {
				argument index;
				}
				修改后：
				extension c-define;

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

yin-element	yin-element变更	argument的子语句yin-element有两个取值，true和false，默认值为false，当yin-element由true变false或由false变true	不兼容	修改前：
				extension c-define {
				argument index {
				yin-element true;
				}
				}
				修改后：
				extension c-define {
				argument index;
				}

revision-date	新增	在import和include中新增revision-date子语句	不兼容	include huawei-ac-ne-evnbgp-type {
				revision-date 2016-01-05;
				}

	删除	在import和include中删除revision-date子语句	不兼容	
	变更	在import和include中修改revision-date子语句	不兼容	

augment	新增	新增augment定义	兼容	新增如下定义：
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	删除、变更	删除或者变更augment定义	不兼容	删除或者修改如下定义：
				augment "/l3vpn:l3vpn-svc/l3vpn:sites/l3vpn:site/l3vpn:site-network-accesses/l3vpn:site-network-access/l3vpn:service/l3vpn:qos/l3vpn:qos-profile/l3vpn:qos-profile" {
				case custom-unicom {
				container inbound-classes {
				uses class-profile;
				}
				container outbound-classes {
				uses class-profile;
				}
				}
				}

refine	新增、删除、变更	变更都算不兼容	不兼容	container connection {
				container source {
				uses target {
				refine "address" {
				description "Source IP address";
				}
				refine "port" {
				description "Source port number";
				}
				}
				}

presence	变更	presence描述变更	兼容	修改前：
				container "ssh" { presence "Enables SSH"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }
				修改后：
				container "ssh" { presence "Enable SSH configuration"; description "SSH service specific configuration"; // more leafs, containers and stuff here... }

变更项	变更名称	变更描述	兼容/不兼容	Yang举例

	新增、删除	presence属性新增、删除	不兼容	container "ssh" {
				presence "Enables SSH";
				description "SSH service specific configuration";
				// more leafs, containers and stuff here...
				}

ordered-by	新增、删除、变更	变更都算不兼容	不兼容	leaf-list cipher {
				type string;
				ordered-by user;
				description "A list of ciphers";
				}

8 业务安全

下面描述开放可编程业务安全注意事项：

- 映射模板定制：

  支持nodelete标签，不支持create标签。nodelete标签用于场景：当业务删除或者更新时会触发分解删除设备配置，该标签会决定删除设备配置是否执行。

  用户可以基于业务逻辑打上“no_delete”或“no_delete_nested”属性，标记该属性的数据是不会随着业务实例的删除删除掉。

  - no_delete：表示打上标记的本层节点不会被删除，子节点是可以被删除的。
  - no_delete_nested：表示打上标记的本层节点以及子节点都不会被删除。

  该标签不会删除数据，但会删除数据源。数据源（配置源）：SSP包中的SSP-实例1 分解接口配置interface 0/0/1和里面的接口属性attr1（value1）和attr2（value2），在网元配置interface 0/0/1以及属性attr1和attr2上面会打上SSP-实例1的path作为标记，该标记成为数据源。
- 数据一致性定制：如果用户定制了对账可删除的开关，就会在数据对账时，删除设备配置。
- 对账时是否支持删除转发器的配置，true值为支持，false值为不支持。

  例子：

  ```
  syncToDel.key = "sync-to-del-enable"
    syncToDel.value = "true"
  ```
- 回退逻辑：
  - 支持事务的设备回退能力：利用设备的事务能力，执行命令行报错时，执行cancel命令行（cancel命令行可以在SND包getCliDriverInfo中配置）。
  - 不支持事务的设备回退能力：根据设备的返回结果，识别出已经执行成功的命令行；将下发成功的命令行生成逆向报文下发给设备。

