从引流说起

如何写引流路由器的插件

CONTENTS 目 录

- 01 需求
  - 为什么要引流
- 02 理论
  - 如何引流
- 03 实践
  - 当前插件的写法
- 04 再实践
  - 推荐的插件的写法

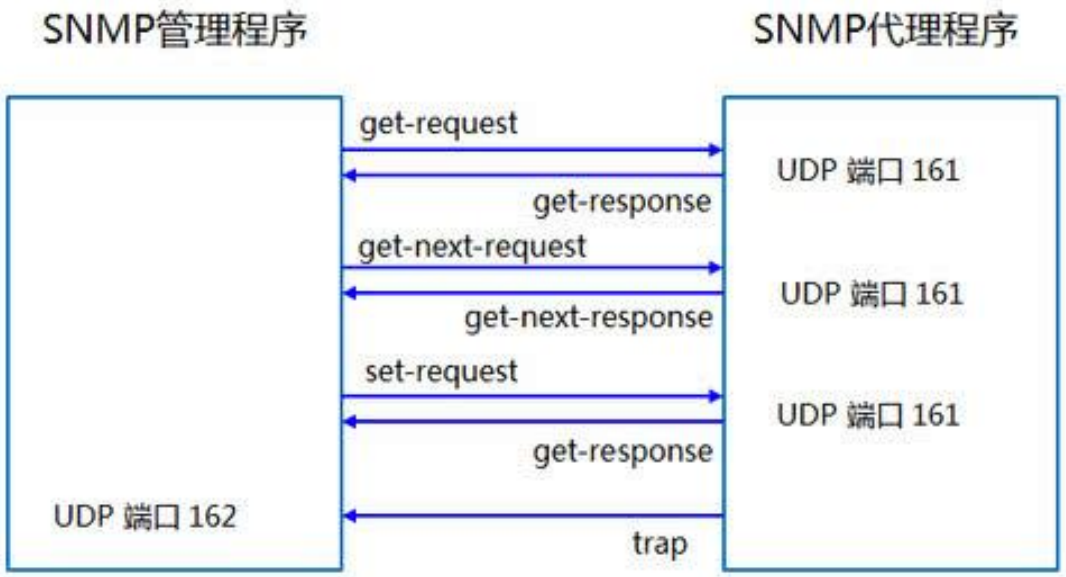
02

理论

如何引流

- 引流手段
  - DNS
  - BGP
  - **PBR**
- 配置协议
  - SNMP
  - OPENFLOW
  - **NETCONF**

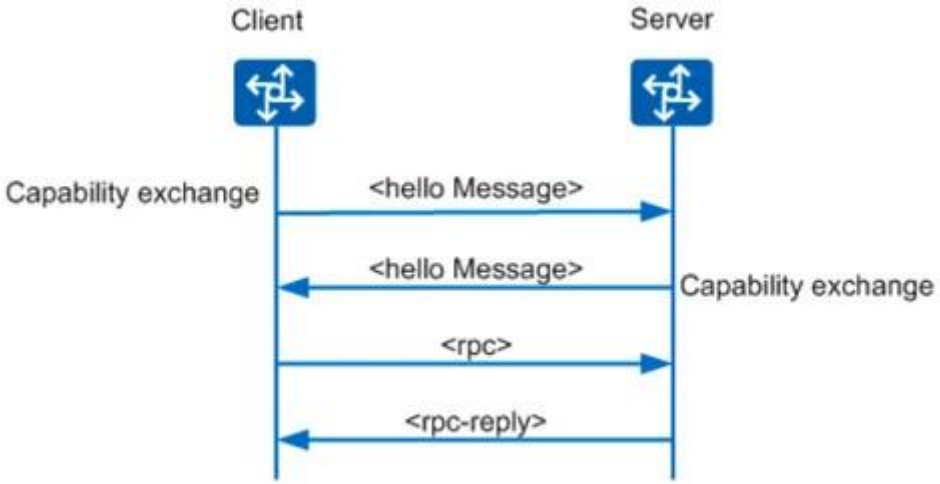
SNMP



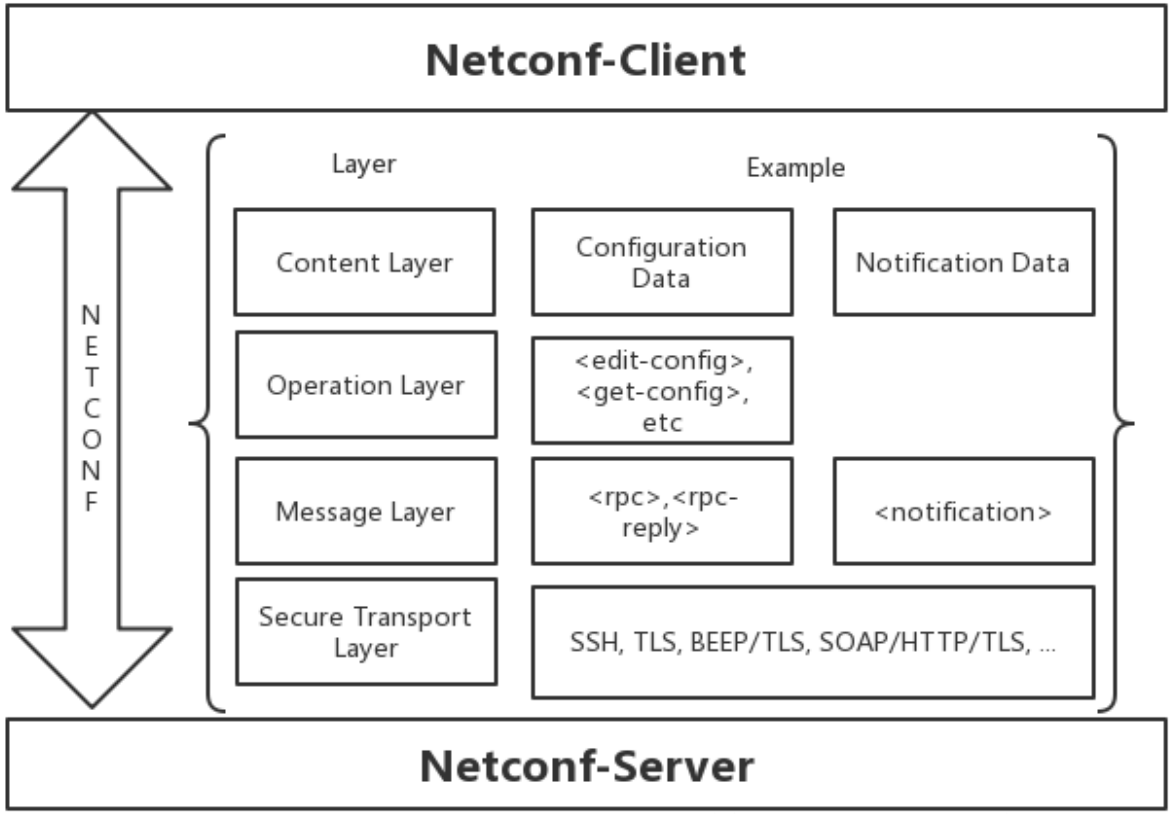
- 具有一些配置的功能，但本身并不是面向配置的协议，也不适合开发用于配置的客户端应用。
- 更多是作为网络设备状态监控的工具，而在配置管理层面，不具备成为一个成熟工具的能力。
- 即便在网络设备监控层面，SNMP也还存在着其他的问题，比如：
  - 性能差，效率低；
  - 基于UDP协议传输，可靠性较差；

NETCONF

- NETCONF = The **Net**work **Conf**iguration Protocol
- NETCONF本身是一个应用层协议
- NETCONF采用的是C/S的模式



NETCONF协议模型



NETCONF把网络的模型数据分为两大类：

1. 状态数据

一般指设备的固有属性数据和当前运行的状态数据等，这类数据仅能查询

2. 配置数据

指由用户（以某种方式）配置到server上的数据。可以存在多个数据库

- <running/>库用于保存当前已经生效的配置；
- <candidate/>用于保存可以提交为生效的数据；
- <startup/>用于保存启动时的配置数据。

NETCONF协议详解

NETCONF协议内部分为4层，由下至上分别是：

- 安全传输层

  支持两种安全传输网络配置协议分别是Secure Shell(SSH)和传输层安全(TLS)

- 消息层

  NETCONF中定义了三种消息类型，分别是hello， rpc和rpc-reply, notification

- 操作层

  仅承载在<rpc>和<rpc-reply>消息上，<hello>和<notification>消息无操作层

- 内容层

  开放：协议本身没有对内容层的数据结构做任何的限定

  规范：而其规范则体现在其内容层需要使用Yang语言对其数据进行建模

NETCONF协议详解-消息层

<hello>

<hello>仅用于会话刚刚建立时netconf-server和netconf-client之间进行能力交换。C/S双方互发<hello>且协商版本成功后，认为netconf会话建立成功。

几种常用的能力

XPath Capability

client可以在filter中使用XPath表达式作为过滤条件

Writable-Running Capability

server支持直接对<running/>库进行修改操作。

Candidate Configuration Capability

server具有一个candidate数据库，并且可以将candidate数据库中的配置提交生效并更新running数据库

Rollback-on-Error Capability

server在执行client发送的配置数据出错后可以进行回滚

Validate Capability

server可以校验client发送的配置数据是否正确

Distinct startup Capability

server有一个startup数据库，用于保存启动配置

NETCONF协议详解-消息层

<rpc>和<rpc-reply>

<rpc>是由client发起的发送到server的消息,用于client请求server执行某项具体的操作。<rpc>包含一个强制属性”message-id”，这个id是一个单调递增的正整数，同一会话内不能重复。

<rpc-reply>是server发送给client的rpc响应。定义了两种默认的元素<ok>和<rpc-error>。

要点:

- netconf-client必须保证server收到的rpc请求的顺序和message-id的顺序是一致的。
- netconf-server在能保证数据不冲突的前提下可以并行处理收到的rpc请求。
- netconf-server在发送<rpc-reply>时必须严格按照收到的<rpc>的顺序。

NETCONF协议详解-消息层

<notification>

Since NETCONF1.1版本。支持Notification上报的netconf-server需在能力交换时上报能力

要点：

- Netconf的通知采用的是订阅发布机制，server仅会向发送过订阅请求的client发送通知。
- Netconf的通知是以Stream进行分类的，不同类的Stream以不同的stream-name进行区分。

NETCONF协议详解-操作层

NETCONF协议规定了9种简单的rpc操作：

- **<get>**用于查询状态数据，另外如果server支持，还可以使用filter进行条件查询
- **<get-config>**用于查询配置数据，可以通过 <source/>来指定不同的配置库
- **<edit-config>**用于对指定配置数据库的内容进行修改，支持merge，replace，create，delete，remove几种操作
- **<copy-config>**将一个库的数据复制到另一个库。
- **<delete-config>**删除一个数据库。但是<running/>库不能被删除。
- **<lock>**获取指定数据库的锁，当某个client获得了指定数据库的锁之后，在其没有释放该锁之前，其余client均不能获得该数据库的锁，也不能对其进行修改操作。同一client也不能在没有释放锁之前，重复申请锁。
- **<unlock>**释放指定数据库的锁。client只能释放自己持有的锁，不能释放其它client的锁。
- **<close-session>**优雅关闭netconf会话，netconf-server将释放该client持有的锁和为其分配的资源，并优雅的关闭与该client链接。所有在<close-session>之后收到的操作均会被忽略。
- **<kill-session>**强制关闭netconf会话。

NETCONF协议详解-内容层

- 内容层未指定具体的模型结构，而是指定了一套建模语言–yang。
- 使用yang定义的数据模型，均可以作为netconf的内容层。所以扩展对netconf来说就是不断的增加和修改yang文件而已。
- 主流设备厂商设备均可以支持NETCONF功能。
- 对于运维人员来说只需拿到设备厂商的YANG模型，根据YANG模型的要求将相关的配置或数据参数填入，即可通过NETCONF发起相关RPC操作，完成配置或数据读取任务。
- 通过YANG和NETCONF，就实现了基于可编程技术的网络配置和管理，为自动化运维提供了基础。
- YANG是针对NETCONF访问的数据建模语言，NETCONF用它对各种操作、数据、以及通知进行建模，具有良好的可读性和可扩展性。
- 设备和客户端都可以使用YANG进行建模。设备侧提供了YANG数据模型后，客户端可依据工具自动生成对应的访问模型代码，以节省开发工作量

NETCONF的问题

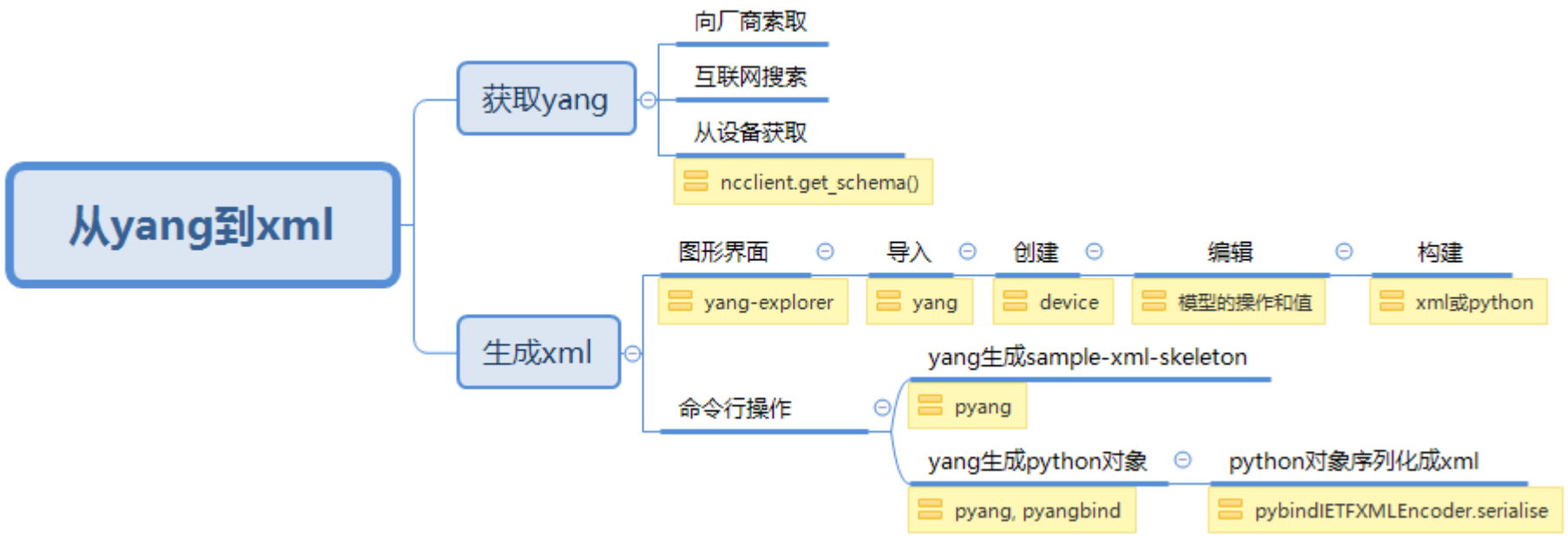
- 各厂商设备支持NETCONF基本都使用私有YANG模型， 不同厂商设备不能识别其他厂商YANG文件；
- 对于使用NETCONF运维平台的使用者，需要学习熟悉各厂商的NETCONF文档和YANG文件；
- NETCONF YANG Model整体的标准化进程十分缓慢

03

实践

流程

1. 设备环境准备
   - 开启netconf over SSH
   - 添加账号权限
   - ~~配置固定的acl组等~~
2. 构造需要的xml



流程

3. 使用标准的客户端python库发送请求

- **pip install ncclient　　#必选**
- **pip install pyang pyangbind xmltodict #可选**

Supported device handlers

When instantiating a connection to a known type of NETCONF server:

- Juniper: `device_params={'name':'junos'}`
- Cisco:
  - CSR: `device_params={'name':'csr'}`
  - Nexus: `device_params={'name':'nexus'}`
  - IOS XR: `device_params={'name':'iosxr'}`
  - IOS XE: `device_params={'name':'iosxe'}`
- Huawei:
  - `device_params={'name':'huawei'}`
  - `device_params={'name':'huaweiyang'}`
- Alcatel Lucent: `device_params={'name':'alu'}`
- H3C: `device_params={'name':'h3c'}`
- HP Comware: `device_params={'name':'hpcomware'}`
- Server or anything not in above: `device_params={'name':'default'}`

```
├── devices
│   ├── alu.py
│   ├── csr.py
│   ├── default.py
│   ├── h3c.py
│   ├── hpcomware.py
│   ├── huawei.py
│   ├── iosxe.py
│   ├── iosxr.py
│   ├── junos.py
│   └── nexus.py
├── operations
│   ├── third_party
│   │   ├── h3c
│   │   │   └── rpc.py
│   │   ├── hpcomware
│   │   │   └── rpc.py
│   │   ├── huawei
│   │   │   └── rpc.py
│   │   ├── iosxe
│   │   │   └── rpc.py
│   │   ├── juniper
│   │   │   └── rpc.py
│   │   └── nexus
│   │       └── rpc.py
│   ├── edit.py
│   ├── errors.py
│   ├── flowmon.py
│   ├── lock.py
│   ├── notify.py
│   ├── retrieve.py
│   ├── rpc.py
│   ├── session.py
│   ├── subscribe.py
│   └── util.py
├── transport
│   ├── third_party
│   │   └── junos
│   │       └── ioproc.py
│   ├── errors.py
│   ├── session.py
│   └── ssh.py
├── capabilities.py
├── debug.py
├── manager.py
└── xml_.py
```

流程

3. 使用标准的客户端python库发送请求

```
OPERATIONS = {
    "get": operations.Get,
    "get_config": operations.GetConfig,
    "get_schema": operations.GetSchema,
    "dispatch": operations.Dispatch,
    "edit_config": operations.EditConfig,
    "copy_config": operations.CopyConfig,
    "validate": operations.Validate,
    "commit": operations.Commit,
    "discard_changes": opera
    "delete_config": operati
    "lock": operations.Lock,
    "unlock": operations.Unl
    "create_subscription": o
    "close_session": operati
    "kill_session": operatio
    "poweroff_machine": oper
    "reboot_machine": operat
}
```

```
class JunosDeviceHandler(DefaultDeviceHandler):
    """
    def __init__(self, device_params):

    def add_additional_operations(self):
        dict = {}
        dict["rpc"] = ExecuteRpc
        dict["get_configuration"] = GetConfiguration
        dict["load_configuration"] = LoadConfiguration
        dict["compare_configuration"] = CompareConfiguration
        dict["command"] = Command
        dict["reboot"] = Reboot
        dict["halt"] = Halt
        dict["commit"] = Commit
        return dict
```

```
class HuaweiDeviceHandler(DefaultDeviceHandler):
    """
    _EXEMPT_ERRORS = []

    def __init__(self, device_params):


    def add_additional_operations(self):
        dict = {}
        dict["cli"] = CLI
        dict["action"] = Action
        return dict
```

```
class H3cDeviceHandler(DefaultDeviceHandler):
    """
    _EXEMPT_ERRORS = []

    def __init__(self, device_params):

    def add_additional_operations(self):
        dict = {}
        dict["get_bulk"] = GetBulk
        dict["get_bulk_config"] = GetBulkConfig
        dict["cli"] = CLI
        dict["action"] = Action
        dict["save"] = Save
        dict["load"] = Load
        dict["rollback"] = Rollback
        return dict
```

yang

- Juniper
https://github.com/Juniper/yang

- 华为
https://github.com/Huawei/yang

- Cisco
https://github.com/YangModels/yang/tree/master/vendor/cisco

- 其它
https://github.com/YangModels/yang/tree/master/vendor

Tools :

Commercial:

- Applied Informatics
- GoAhead
- SNMP Research
- Tail-f Systems

Open Source:

- Ncclient (client)
- netopeer (client/server)
- YencaP (client/server)
- Yuma (client/server)

工具1-Yang explorer

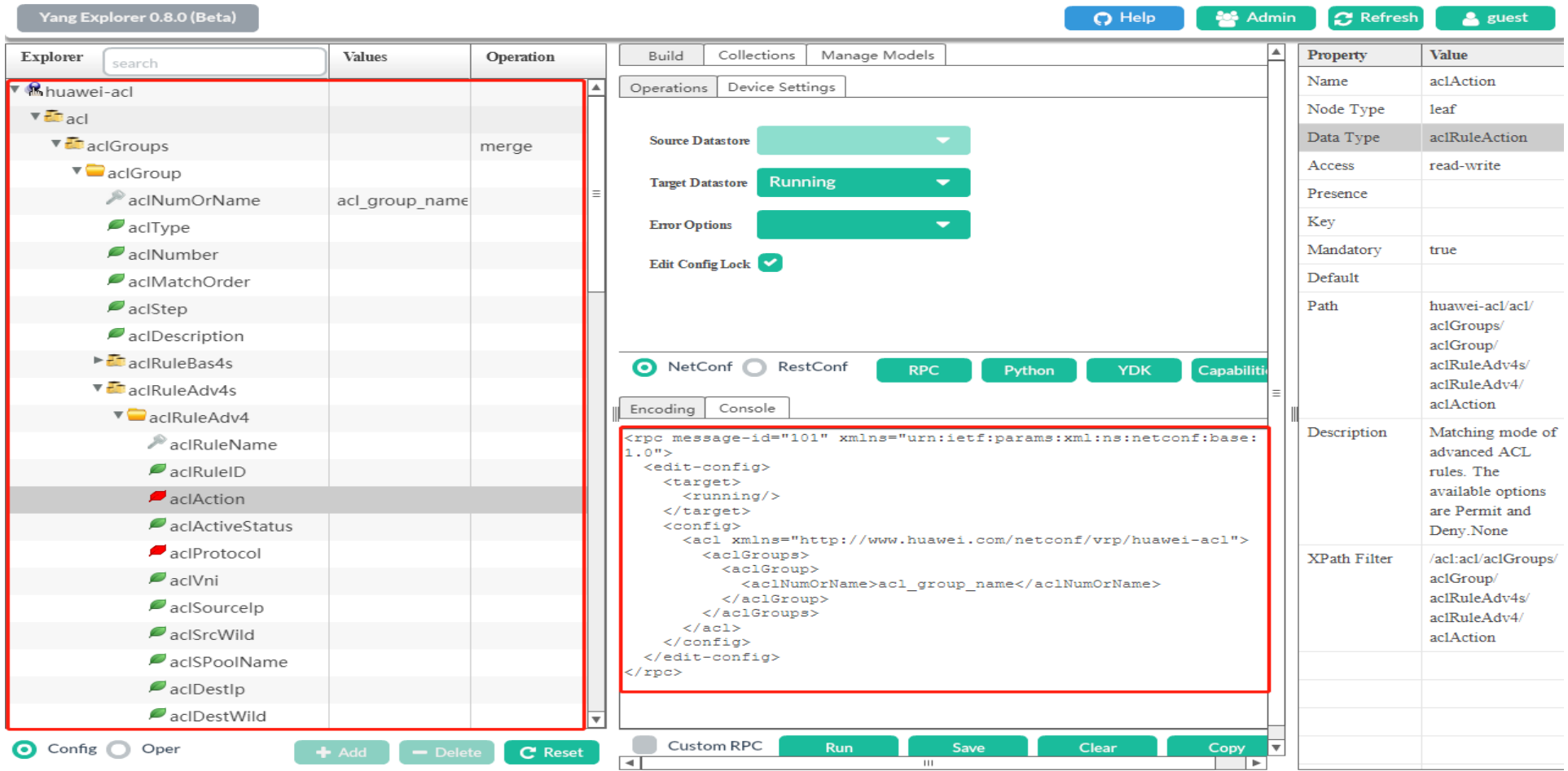
https://github.com/CiscoDevNet/yang-explorer

Features

- 从用户界面或者 命令行 上传/编译yang模型
- 生成 NetConf RPC
- 生成 python 示例代码 **[new]**
- 搜索 yang **[new]**
- 针对真正的netconf服务器执行 RPC
- 将创建的rpc保存到收藏集以便以后使用
- 建立模型的依赖关系图
- 浏览数据模型树并检查yang属性

工具1-Yang explorer

http://10.5.0.20:8088/static/YangExplorer.html
Login:guest/guest



工具2-pyang&pyangbind

pip install pyang
pip install pyangbind

pyang -f sample-xml-skeleton --sample-xml-skeleton-annotations --sample-xml-skeleton-defaults --sample-xml-skeleton-doctype=config --sample-xml-skeleton-path=acl/aclGroups -o huawei_acl.xml ./yang/network-router/8.9.10/huawei/huawei-acl.yang -p ./yang/network-router/8.9.10/huawei/

http://10.5.0.12/huawei_acl.xml

工具2-pyang&pyangbind

```
pip  install pyang
pip install pyangbind


export PYBINDPLUGIN=`/usr/bin/env python -c 'import
pyangbind; import os; print
("{}/plugin".format(os.path.dirname(pyangbind.__file__)
))'`
echo $PYBINDPLUGIN

pyang --plugindir $PYBINDPLUGIN -f pybind -o
huawei_acl.py ./yang/network-
router/8.9.10/huawei/huawei-acl.yang -
p ./yang/network-router/8.9.10/huawei/
```

https://github.com/robshakir/pyangbind

```
7  import pyangbind.lib.pybindJSON as pybindJSON
8  from pyangbind.lib.serialise import pybindIETFXMLEncoder
9
10 from huawei_acl_model import huawei_acl
11
12 if __name__ == '__main__':
13     root = huawei_acl()
14     acl_group = root.acl.aclGroups.aclGroup.add('group_a')
15     rule = acl_group.aclRuleAdv4s.aclRuleAdv4.add('rule_a')
16     rule.aclDestIp = '1.1.1.1'
17
18     # Dump the entire instance as RFC 7950 XML
19     print pybindIETFXMLEncoder.serialise(root)
20
```

Console | PyUnit | Search

<terminated> S:\workspace\PyFirst\src\yang\huawei\huawei_acl_test.py

```
<huawei-acl xmlns="http://www.huawei.com/netconf/vrp">
  <acl xmlns="http://www.huawei.com/netconf/vrp/huawei-acl">
    <aclGroups>
      <aclGroup>
        <aclNumOrName>group_a</aclNumOrName>
        <aclRuleAdv4s>
          <aclRuleAdv4>
            <aclDestIp>1.1.1.1</aclDestIp>
            <aclRuleName>rule_a</aclRuleName>
          </aclRuleAdv4>
        </aclRuleAdv4s>
      </aclGroup>
    </aclGroups>
  </acl>
</huawei-acl>
```

工具3-MG-SOFT NETCONF*

- MG-SOFT NETCONF/YANG Python Scripting System
- MG-SOFT NETCONF Browser Professional Edition

商用，30天试用

谢谢！