**NETCONF Java开源框架比较**

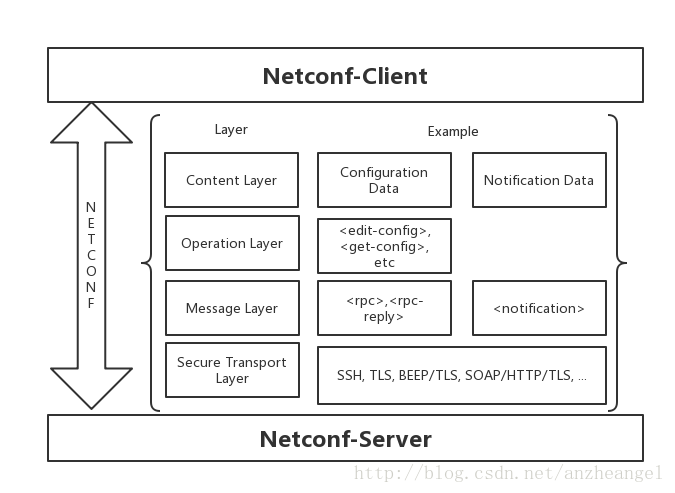
**目录**

随着SDN的大热，一个诞生了十年之久的协议焕发了第二春，它就是NETCONF协议。如果你在两年前去搜索NETCONF协议，基本得到的信息都是“这个协议是一个网管协议，主要目的是弥补SNMP协议的不足，希望可以取代SNMP协议”。SNMP有哪些不足，而NETCONF是否真的能够弥补，这都不是重点，重点是NETCONF诞生至今SNMP依旧活的好好的。所以如果我们还是把NETCONF当做一个网管协议的话，估计它会在冷板凳上一直坐下去，而如果我们换一个角度去看待NETCONF协议，你会发现也许它是最适合SDN的一个协议。

**0. 概述**

NETCONF = The **Network** Configuration Protocol   
SDN = Software Define **Network**   
从文字含义上就觉得NETCONF和SDN可以在一起搞事情，搞什么事情？ 搞Network啊。   
SDN要用软件去定义网络，如何定义？简单就是要用软件去配置网络，有人说”配置网络太空泛了，能不能具体点？”，但我真的没法说的具体，因为网络能配置的东西太多了，网络的设备类型多种多样，业务类型更是成百上千了，但SDN就是想在这么复杂的网络上开辟一番新的天地，NETCONF可以说是其不二的选择。   
接下来就对NETCONF1.1(RFC6241)版本进行详细分析。

**1. NETCONF1.1协议详解**

NETCONF采用的是C/S的模式：   
  
从上图中可以看出NETCONF协议内部分为4层，由下至上分别是安全传输层，消息层，操作层和内容层。   
在详细介绍这四层之前，需要提前建立一个概念，NETCONF认为网络的模型数据可以分为两大类，即状态数据和配置数据。状态数据一般指server（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。而配置数据则是指由用户（以某种方式）配置到server上的数据。而配置数据本身又可以存在多个数据库，标准中提到了<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；以及<startup/>用于保存启动时的配置数据。

**1.1. 安全传输层**

NETCONF的第一大优势就是其从协议层面就已经规定其传输层**必须**使用带有安全加密的通信协议，例如SSH，TLS等。相比与其它也允许明文传输的协议来说其在协议层面就已经对数据安全做了第一道守护。由于NETCONF协议规定必须要支持SSH，所以目前SSH是NETCONF使用最广泛的传输层协议。   
NETCONF的协议内容是承载在安全传输层之上的，所以NETCONF本身是一个应用层协议，所以NETCONF协议中并没有规定**建链**和**保活**相关的内容。

**1.2. 消息层**

NETCONF中定义了三种消息类型，分别是

**hello**，**rpc**和**rpc-reply**,**notification**。

**1.2.1. <hello>**

<hello>仅用于回话刚刚建立时netconf-server和netconf-client之间进行能力交换。   
server和client需要在回话建立后互相发送<hello>消息，并在<hello>消息中携带自身支持的能力，以及支持的netconf协议的版本号，server和client根据自身和对方的能力信息协商使用的netconf版本。   
一般来说，C/S双方互发<hello>且协商版本成功后，认为netconf会话建立成功。

**1.2.1.1. 几种常用的能力**

**(1) XPath Capability**   
该能力表示client可以在filter中使用XPath表达式作为过滤条件   
Capability Identifier:   
urn:ietf:params:netconf:capability:xpath:1.0   
**(2) Writable-Running Capability**   
该能力表示server支持直接对<running/>库进行修改操作。   
Capability Identifier:   
urn:ietf:params:netconf:capability:writable-running:1.0   
**(3) Candidate Configuration Capability**   
该能力表示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   
Capability Identifier:   
urn:ietf:params:netconf:capability:candidate:1.0   
**(４) Rollback-on-Error Capability**   
该能力表示server在执行client发送的配置数据出错后可以进行回滚   
Capability Identifier:   
urn:ietf:params:netconf:capability:rollback-on-error:1.0   
**(５) Validate Capability**   
该能力表示server可以校验client发送的配置数据是否正确   
Capability Identifier:   
urn:ietf:params:netconf:capability:validate:1.1   
**(６) Distinct startup Capability**   
该能力表示server有一个startup数据库，用于保存启动配置   
Capability Identifier:   
urn:ietf:params:netconf:capability:startup:1.0

**1.2.2. <rpc>和<rpc-reply>**

<rpc>是由netconf-client发起的发送到netconf-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。该id用于<rpc>和<rpc-reply>的配对。   
<rpc-reply>是有netconf-server发送给netconf-client的rpc响应。不能主动发起，仅能在收到<rpc>之后回复，切必须携带与收到的rpc相同的message-id。   
在<rpc-reply>定义了两种默认的元素分别是<ok>和<rpc-error>。<ok>表示未定义响应内容的rpc执行成功，而<rpc-error>表示rpc执行失败。   
关于RPC最重要的一点： 原文如下：   
NETCONF<rpc>requests MUST be processed serially by the managed device. Additional<rpc>requests MAY be sent before previous ones have been completed. The managed device MUST send responses only in the order the requests were received.   
个人对这段话的理解是这样的：   
1. 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。   
2. 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。   
3. netconf-server在发送<rpc-reply>时必须严格按照收到的<rpc>的顺序。

**1.2.3. <notification>**

在netconf的1.0版本中还没有加入Notification相关的内容，而在1.1版本已经将RFC5277(NETCONF Event Notifications)囊落在内了。支持Notification上报的netconf-server需在能力交换时上报能力：   
“urn:ietf:params:netconf:capability:notification:1.0”   
几个关键的知识点：   
1. 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。   
2. 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-server默认需要支持的stream-name是”NETCONF”。   
3. client不能重复下发订阅，即同一Stream的订阅不能重复下发，也不能同时订阅多个Stream，订阅可以设置定时取消，如果没有设置终止时间，取消订阅需要使用close-session或者kill-session。定时取消的订阅netconf的会话还是激活的，而使用close-session或者kill-session来取消的话，netconf会话会关闭。   
Stream发现：   
例如：

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

<get>

<filter type="subtree">

<netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">

<streams/>

</netconf>

</filter>

</get>

</rpc>

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

<data>

<netconf xmlns="urn:ietf:params:xml:ns:netmod:notification">

<streams>

<stream>

<name>NETCONF</name>

<description>default NETCONF event stream</description>

<replaySupport>true</replaySupport>

<replayLogCreationTime>2007-07-08T00:00:00Z</replayLogCreationTime>

</stream>

<stream>

<name>SNMP</name>

<description>SNMP notifications</description>

<replaySupport>false</replaySupport>

</stream>

<stream>

<name>syslog-critical</name>

<description>Critical and higher severity</description>

<replaySupport>true</replaySupport>

<replayLogCreationTime>2007-07-01T00:00:00Z</replayLogCreationTime>

</stream>

</streams>

</netconf>

</data>

</rpc-reply>

发起订阅：

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

<create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">

<stream>SNMP</stream>

</create-subscription>

</rpc>

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

<ok/>

</rpc-reply>

通知内容举例:

<notification xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">

<eventTime>2007-07-08T00:10:00Z</eventTime>

<event xmlns="http://example.com/event/1.0">

<eventClass>state</eventClass>

<reportingEntity>

<card>Ethernet0</card>

</reportingEntity>

<operState>enabled</operState>

</event>

</notification>

**1.3. 操作层**

操作层仅承载在仅<rpc>和<rpc-reply>消息上，<hello>和<notification>消息无操作层。   
NETCONF协议规定了9种简单的rpc操作，同时也支持用户自定义rpc操作。有关自定义操作的内容放到内容层来讲。

**1.3.1. <get>**

用于查询状态数据，另外如果server支持能力：urn:ietf:params:netconf:capability:xpath:1.0则还可以使用filter进行条件查询，例如：

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<get>

<filter type="subtree">

<top

xmlns="http://example.com/schema/1.2/stats">

<interfaces>

<interface>

<ifName>eth0</ifName>

</interface>

</interfaces>

</top>

</filter>

</get>

</rpc>

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

<data>

<top xmlns="http://example.com/schema/1.2/stats">

<interfaces>

<interface>

<ifName>eth0</ifName>

<ifInOctets>45621</ifInOctets>

<ifOutOctets>774344</ifOutOctets>

</interface>

</interfaces>

</top>

</data>

</rpc-reply>

**1.3.2. <get-config>**

用于查询配置数据，可以通过 <source/>来指定不同的配置库，例如：

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

<get-config>

<source>

<running/>

</source>

<filter type="subtree">

<top xmlns="http://example.com/schema/1.2/config">

<users/>

</top>

</filter>

</get-config>

</rpc>

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

<data>

<top xmlns="http://example.com/schema/1.2/config">

<users>

<user>

<name>root</name>

<type>superuser</type>

<full-name>Charlie Root</full-name>

<company-info>

<dept>1</dept>

<id>1</id>

</company-info>

</user>

<!-- additional <user> elements appear here... -->

</users>

</top>

</data>

</rpc-reply>

**1.3.3. <edit-config>**

用于对指定配置数据库的内容进行修改，支持以下几种操作：   
**merge:** 合并操作，此操作为默认操作。   
**replace:** 替换操作，如果对象已经存在则替换，不存在则创建。   
**create:** 创建操作，如果对象已经存在，则报错误“data-exists”。   
**delete:** 删除操作，如果对象存在则删除，不存在则报错 “data-missing”。   
**remove:** 删除操作，如果对象存在则删除，不存在则忽略。   
举例：

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<edit-config>

<target>

<running/>

</target>

<config

xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">

<top

xmlns="http://example.com/schema/1.2/config">

<interface xc:operation="replace">

<name>Ethernet0/0</name>

<mtu>1500</mtu>

<address>

<name>192.0.2.4</name>

<prefix-length>24</prefix-length>

</address>

</interface>

</top>

</config>

</edit-config>

</rpc>

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<edit-config>

<target>

<running/>

</target>

<default-operation>none</default-operation>

<config

xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">

<top

xmlns="http://example.com/schema/1.2/config">

<protocols>

<ospf>

<area>

<name>0.0.0.0</name>

<interfaces>

<interface xc:operation="delete">

<name>192.0.2.4</name>

</interface>

</interfaces>

</area>

</ospf>

</protocols>

</top>

</config>

</edit-config>

</rpc>

**1.3.4. <copy-config>**

将一个库的数据复制到另一个库。   
举例：

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<copy-config>

<target>

<running/>

</target>

<source>

<url>https://user:password@example.com/cfg/new.txt</url>

</source>

</copy-config>

</rpc>

**1.3.5. <delete-config>**

删除一个数据库。但是<running/>库不能被删除。

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<delete-config>

<target>

<startup/>

</target>

</delete-config>

</rpc>

**1.3.6. <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。   
获取锁的主要目的就是避免并发导致数据冲突。   
举例：

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<lock>

<target>

<running/>

</target>

</lock>

</rpc>

**1.3.7. <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。   
举例：

<rpc message-id="101"

xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">

<unlock>

<target>

<running/>

</target>

</unlock>

</rpc>

**1.3.8. <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。

**1.3.9. <kill-session>**

强制关闭netconf会话。

**1.4. 内容层**

开放但规范的内容层是netconf协议的精髓所在。其开放体现在netconf协议本身没有对内容层的数据结构做任何的限定。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。   
在netconf出现之前，我们所熟知且常用的协议，均采用在协议中规定报文的结构体，并按字节流读取并解析的架构。为了更好的在字节流中表达更丰富的报文结构，我们采用TLV等方式来定义对象。但不知大家是否发现，这种方式几乎不具备任何扩展性，一旦扩充对象，或修改对象就需要变更代码。而如果对一个协议扩展了大量的私有数据，那么首先协议不在标准，其次协议栈的代码几乎是完全重写。   
而netconf的出现可以说直接对上述问题进行了一次”降维“打击，它完全站在了一个更高的维度来解决上述问题。其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。也就是说使用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。   
做一个比喻，传统基于字节流的协议可以比作一份具有特定可执行的程序，而NETCONF则是写程序的编程语言。   
另外一点我们在上面将操作的时候提到netcon支持用户自定义操作。也就是说我们不必纠结标准制定的9个操作类型是否够用，完全可以根据实际的需求在yang文件中定义相应的rpc操作。   
Netconf协议本身的一些扩展也是采用在标准中增加默认支持的yang文件的方式来实现的，比如yang-moudle-monitor。

**2. NETCONF Java开源框架介绍**

**2.1. jnc（java netconf client）**

JNC是NETCONF客户端编码的库和pyang（python中可扩展的YANG验证器和转换器）的java输出格式插件。可用于从YANG模型中生成Java类，生成的文件的不同类型有：

**根类**—这个类有YANG模型的前缀名，包含前缀和名称空间的字段，以及使JNC库能够在与NETCONF服务器交互时使用其他生成的类的方法。

**YangElement**—每个YangElement对应于YANG模型中的一个容器或一个列表。它们表示配置的树的节点，并提供根据生成它们的YANG模型修改配置的方法。YANG模型中的顶级容器或列表将在输出目录中与根类一起生成相应的YangElement类。它们各自的子容器和子列表在子包中生成，其名称与父容器或列表的名称相对应。

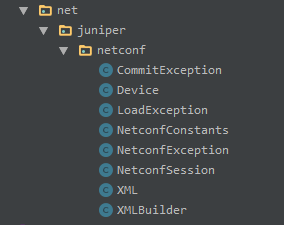
**YangTypes**—对于YANG模型中的每个派生类型，都会在输出目录的根目录下生成一个类。派生类型可以扩展另一个派生类型类，也可以扩展与内置YANG类型相对应的JNC类型类。**Packageinfo**—

代码原生支持NETCONF自带操作，可通过pyang以及jnc插件为自定义操作生成jnc类以支持用户自定义操作。

**不支持NETCONF 1.1（主要是Notification）**

**2.2. netconf-java**

实现非常简单，SSH基于jsch实现，该库代码实现上比较乱，没有ganymed-ssh2优雅高效。



定义的操作很少，不支持notification，没有yanglib集成，只有基本的操作原语，大部分应用层功能需要写一定封装逻辑。**代码量较少，通过阅读可以对netconf实现有一个较深的认识**。

**工程不推荐使用！！！**

**2.3. netconf4j**

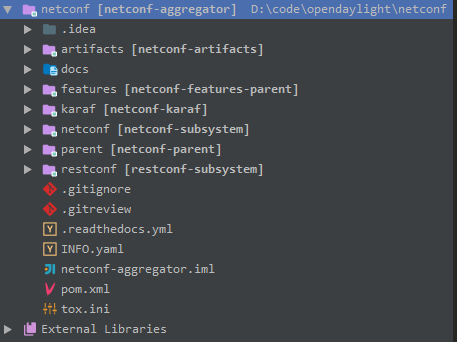
该项目是client实现，SSH基于ganymed-ssh2库，应该是配套其netconf4j-server使用，因此调试过程中发现有一些实现是对不上其他的server实现，目前经过调试能够与netopper2发起session建立，hello，rpc-lock，rpc-unlock，session关闭消息。其他操作暂未测试。

**不推荐使用！！！**

**2.4. netconf-aggregator**

属于opendaylight项目的子项目（基于OSGI框架karaf），实现比较全面，质量很高，SSH实现采用netty框架，具备yanglib，cli，有一套基于opendaylight的数据模型，但正因为如此，该项目复杂度与集成度过高，项目精简需要投入很多精力，到现在也没有把项目搭建起来。

**现阶段不推荐使用！！！**



其中netconf是基于ssh实现，restconf基于http实现。

