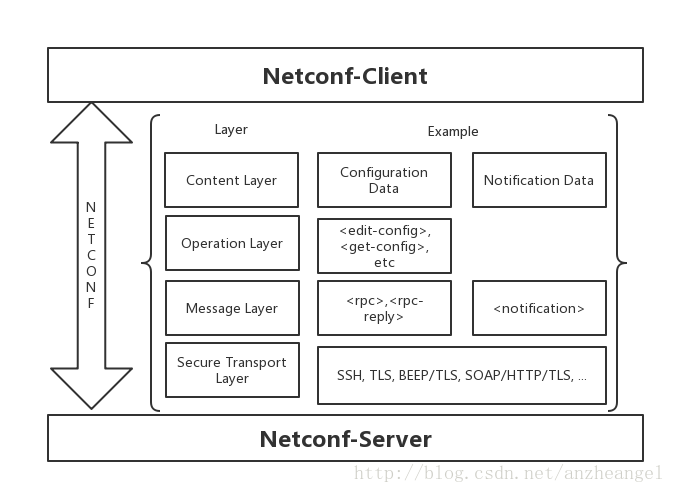
# NETCONF

## 概述

Netconf=the network configuration protocol，NETCONF协议提供了一个标准框架和一个标准远程过程调用 RPC方法的集合。网络管理员和应用开发人员可以根据此框架和集合来操作网络设备的配置。设备的配置数据和NETCONF协议都采用XML格式。

## Netconf协议详解



NETCONF认为网络的模型数据可以分为两大类：

* **状态数据**（一般指server的固有属性和当前运行的状态数据等，这类数据仅能查询）
* **配置数据**（指由用户配置到server上的数据，而数据本身有可以存在多个数据库。）

NETCONF协议框架：

| 层面 | 示例 | 说明 |
| --- | --- | --- |
| 第一层：传输层 | BEEP，SSH，SSL | 传输层为NETCONF Manager和NETCONF Agent之间交互提供通信路径。  NETCONF协议可以使用任何符合基本要求的传输层协议承载，对承载协议基本要求如下：  面向连接，NETCONF Manager和NETCONF Agent之间必须建立持久的连接，连接建立后，必须提供可靠的序列化的数据传输服务。  提供用户认证，数据完整、安全加密，NETCONF协议的用户认证、数据完整、安全保密全部依赖传输层提供。  承载协议必须向NETCONF协议提供区分会话类型（client或server）的机制。  说明：  当前，设备只支持SSH协议作为NETCONF协议的承载协议。 |
| 第二层：RPC层 | <rpc>, <rpc-reply> | RPC层提供了一种简单的、不依赖于传输协议的RPC请求和响应机制。client采用<rpc>元素封装操作请求信息，并通过一个安全的、面向连接的会话将请求发送给服务器，而服务器将采用<rpc-reply>元素封装RPC请求的响应信息（即操作层和内容层的内容），然后将此响应信息发送给请求者。  正常情况下，**<rpc-reply>元素封装客户端所需的数据或配置成功的提示信息**。当客户端请求报文存在错误或服务器处理不成功时，服务器端在<rpc-reply>元素中会封装一个包含详细错误信息的<rpc-error>元素反馈给客户端。 |
| 第三层：操作层 | <get-config>, <edit-config>, <notification> | 操作层定义了一系列在RPC中应用的基本操作，这些操作组成了NETCONF基本能力。 |
| 第四层：内容层 | 配置数据 | 内容层描述了网络管理所涉及的配置数据，由于NETCONF内容层是唯一没有被标准化的层，没有标准的NETCONF数据建模语言和数据模型，所以各制造商设备的配置数据可能会不相同。 |

### 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。

### 1.2.2<rpc>和<rpc-reply>

<rpc>元素封装了从客户端到服务器端的协议操作请求。RPC消息格式为：

<rpc message-id="101"

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

<some-method>

<!--method parameters here -->

</some-method>

</rpc>

<rpc-reply>元素消息是对<rpc>消息的响应。NETCONF agent必须把<rpc>元素中的附加属性放在<rpc-reply>中返回给NETCONF Manager。例如，<rpc>通过调用NETCONF的<get>操作取得”user-id”属性的值，在<rpc-reply>中，将返回该属性的值：

<rpc message-id="101"

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

xmlns:ex="http://example.net/content/1.0"

ex:user-id="fred"

<get/>

</rpc>

<rpc>时由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，同一会话内部能重复，该id用于<rpc>和<rpc-reply>的配对。

<rpc-reply>是由netconf-server发送给netconf-client的rpc响应。部能主动发起，只在收到<rpc>之后恢复，且必须携带与收到rpc相同的message-id。

在<rpc-reply>定义了两种默认的元素，分别为<ok>和<rpc-error>。<ok>表示定义响应内容的rpc执行成功，而<rpc-error>表示rpc执行失败。

**敲黑板：**

* **Netconf-client必须保证server收到的rpc请求的顺序和message-id的顺序一致的。**
* **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。**

### 1.2.3<notification>

几个关键知识点：

* 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-server默认需要支持的stream-name是“netconf”。
* Client不能重复下发订阅。

### 1.3操作层

NETCONF协议的核心在于操作层。NETCONF操作层定义了1个底层操作的集合用以配置和获取设备信息，而设备信息一般存放在数据库中，所以定义的操作提供了对数据库信息的获取、配置、复制和删除等功能。值得注意的是，**NETCONF定义的基本操作只是必须实现的功能的最小集合，而不是功能的全集。**

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

**表3-3**  NETCONF协议基本能力定义的基本操作

| **基本操作** | **说明** |
| --- | --- |
| <get-config> | 用来从<running/>、<candidate/>和<startup/>配置数据库中获取全部或指定的一部分配置数据。 |
| <get> | 用来从<running/>配置数据库中获取部分或全部运行的配置数据和设备的状态数据。 |
| <edit-config> | 用来修改、创建、删除配置数据。 |
| <copy-config> | 用源配置数据库替换目标配置数据库。如果目标配置数据库没有创建，则直接创建配置数据库，否则用源配置数据库直接覆盖目标配置数据库。 |
| <delete-config> | 用来删除一个配置数据库，但不能删除<running/>配置数据库。 |
| <lock> | 用来锁定设备的<running/>数据库，独占配置数据库的修改权。这种锁定允许用户在配置时，不会受到如NETCONF Mannager、SNMP或者CLI脚本的配置影响，防止产生冲突。 |
| <unlock> | 用来取消用户自己之前执行的<lock>操作，但不能取消其他用户的<lock>操作。 |
| <close-session> | 用来正常关闭当前NETCONF会话。 |
| <kill-session> | 用来强制关闭另一个NETCONF会话，只有管理员用户才有权限执行<kill-session>操作。 |

## 1.4NETCONF内容层

目前，NETCONF协议有三个标准概念配置数据库。

1. <running/>

此数据库代表目前在设备上允许的生效配置。

<running/>数据库同样也包含当前设备上可用的所有概念状态信息。<get>可以在<running/>数据库上获得状态信息、统计信息和配置参数。<get-config>操作只能获得配置数据。

1. <candidate/>

NETCONF支持candidate能力，可以在<candidate/>数据库上进行操作。对<candidate/>数据库的任何改变不会马上直接影响网络设备。管理员需要使用<commit>操作来运行<commit/>数据库上变化的配置，并使这些变化的配置成为<running/>数据库的一部分。<commit>操作执行后，<candidate/>和<running/>数据库的内容是一样的。<candidate/>数据库是全局数据库，因此管理员可以用<discard-changes>操作来放弃已变化但是不想执行的配置。