目录

[1 产生背景 4](#_Toc386961470)

[1.1 NETCONF协议 4](#_Toc386961471)

[1.1.1 Why YANG? 5](#_Toc386961472)

[2 YANG使用入门 5](#_Toc386961473)

[2.1 Modules 5](#_Toc386961474)

[2.1.1 模块 5](#_Toc386961475)

[2.1.2 子模块 6](#_Toc386961476)

[2.1.3 命名空间 8](#_Toc386961477)

[2.1.4 import(include) revision 8](#_Toc386961478)

[2.1.5 YANG文件 9](#_Toc386961479)

[2.2 YANG数据类型 9](#_Toc386961480)

[2.2.1 内建数据类型 9](#_Toc386961481)

[2.2.2 派生类型 9](#_Toc386961482)

[2.3 YANG数据节点 10](#_Toc386961483)

[2.3.1 leaf 10](#_Toc386961484)

[2.3.2 leaf-list 10](#_Toc386961485)

[2.3.3 container 10](#_Toc386961486)

[2.3.4 list 11](#_Toc386961487)

[2.3.5 grouping, use and refine 11](#_Toc386961488)

[2.3.6 choice 12](#_Toc386961489)

[2.3.7 presence 13](#_Toc386961490)

[2.3.8 anyxml 13](#_Toc386961491)

[2.3.9 augment 13](#_Toc386961492)

[2.4 YANG引用 14](#_Toc386961493)

[2.4.1 leafref 14](#_Toc386961494)

[2.5 数据约束 14](#_Toc386961495)

[2.5.1 config 15](#_Toc386961496)

[2.5.2 mandatory 15](#_Toc386961497)

[2.5.3 unique 15](#_Toc386961498)

[2.5.4 must 15](#_Toc386961499)

[2.6 notification 16](#_Toc386961500)

[2.7 rpc 16](#_Toc386961501)

# 产生背景

## NETCONF协议

随着网络的飞速发展，网络规模不断扩大，网络的异构性和复杂性不断增加，网络设备的种类和数目日益增多，地理位置分布更加广泛，对网络配置管理的要求越来越高。简单网络管理协议SNMP(Simple Network Management Protocol)是网管领域事实上的工业标准，但SNMP主要用于监控和故障管理，其配置管理功能薄弱，不能解决大型异构网络的多需求配置问题；此外SNMP安全性不高，实时性不强，在设备之间进行交互操作，设备所需要的复杂配置操作以及信息模型的表达方面都存在一定缺陷。尽管经过SNMPv2和SNMPv3两个版本的改进，还是不能满足现阶段网络配置管理的需求。而现有的配置管理领域使用最广泛的CLI(Command Line Language)命令方式，虽然较大程度地满足配置管理的需求。但存在不同厂商的种类数量繁多的CLI命令，配置过程比较复杂，无法实现标准化和自动化的配置。

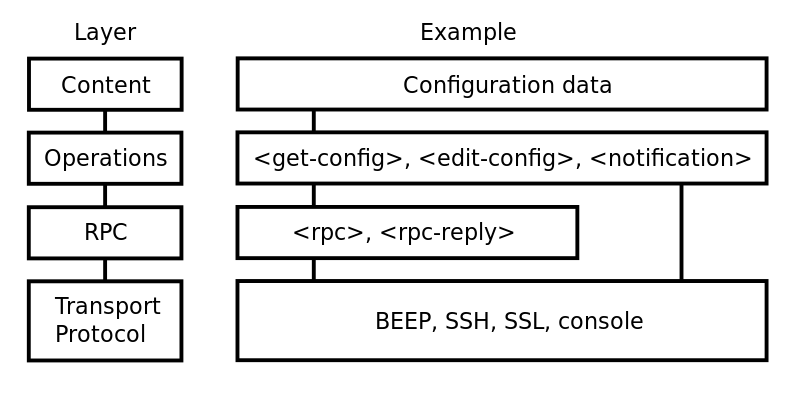
2003年，IETF成立了NETCONF工作组，提出一种基于XML的网络配置管理协议——NETCONF(Network Configuration Protocol)，其配置功能强大，并且兼顾监控和故障管理，安全验证和访问控制，可扩展性强，主要有如下特点：

（1）它将配置数据和状态数据分开，避免操作对不相关数据进行处理，例如可以避免对状态数据进行配置操作，进一步提高效率。

（2）配置功能强大，不仅定义了多种取值和配置操作，还定义了配置修改事务、配置测试和校验等多种能力，并且支持能力的添加，具有较强的扩展性。可满足大型异构网络的配置要求。

（3）规定必须采用安全的面向连接的通信传输方式，相比SNMP采用UDP进行传输的方式来说，NETCONF进一步加强了系统的安全性。同时也避免了UDP传输数据最大值的局限性。

NETCONF协议分为传输层、RPC层、操作层和内容层。其中，内容层是唯一没有标准化的层，于是一种新的建模语言YANG产生了，它的目标是对NETCONF数据模型、操作进行建模，覆盖NETCONF协议的操作层和内容层。



### Why YANG?

为什么不使用现有的建模语言，比如XSD、DSDL？这是因为：

（1）简单，容易入门。

YANG语法简单，层次清晰，元素角色明显。YANG在加强可读性时，同时提供灵活性和可扩展性，能适应于大型模式。

使用人员不需要了解也不需要去学习XSD或DSDL，就可以入门。

（2）YANG是为NETCONF而生的，专门用于网络管理。

它是NETCONF指定的，不是要与XSD或其它建模语言竞争。YANG并不需要你懂得这些技术，它是自给自足的。

YANG将数据模型作为应用与设备间的合约，这些模型相当于API，如果一个设备违反合约，将被视为损坏的或是不可用的。假设我们想对厂商依赖的fork()方法返回一个可移植的C代码，这种协调代价是巨大的。而YANG使应用有能力相信，设备会如同数据模型约定的那样进行表现。

（3）模块化的、灵活的

YANG是完全模块化的。像SIMv2一样，YANG允许只开发模型中的一部分，而不影响其它不需要的部分。

YANG提供足够的灵活性，使得扩展不会影响数据模型的基础合约。对于同一标准数据模型的不同实现，允许使用不同的值。

YANG提供足够的特性来支持网络管理配置，却不增加建模的复杂性，易于使用、扩展和模块化。

（4）继承于IETF网络管理社区

YANG有着很长时间的积累，根本上说，它不是新东西。它是在NETCONF数据建模语言多年研究基础上发展起来的，并和SMIng项目和SMIv2有一段交叉。

（5）被多个NETCONF实现者支持

（6）开源

（7）认可

YANG已经在YANG团队之外，在IETF的其它项目中得到使用。

# YANG使用入门

本节将介绍YANG Model入门，关于更多更全的YANG规范请参考

<http://tools.ietf.org/html/rfc6020> - YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)

## Modules

### 模块

模块（Module）是YANG的基本单元，它是一组YANG语句集合，包括三种类型语句：head语句(yang-version/namespace/prefix)、连接语句(import/include)、元信息(organization/contract)、revision语句、定义语句(container/leaf/leaf-list/grouping等等)。

YANG模块如下布局：

module <module-name> {

// header information

<yang-version statement>

<namespace statement>

<prefix statement>

// linkage statements

<import statements>

<include statements>

// meta information

<organization statement>

<contact statement>

<description statement>

<reference statement>

// revision history

<revision statements>

// module definitions

<other statements>

}

下面是一个module例子：

module controller-network {

yang-version 1;

namespace "urn:opendaylight:controller:network";

prefix "topos";

import ietf-inet-types { prefix "inet"; }

revision 2013-05-20 {

description "Initial demo";

}

### 子模块

子模块(SubModule)是部分模块定义，用于对模块提供支持。

子模块文件内容布局如下：

submodule <module-name> {

<yang-version statement>

// module identification

<belongs-to statement>

// linkage statements

<import statements>

<include statements>

// meta information

<organization statement>

<contact statement>

<description statement>

<reference statement>

// revision history

<revision statements>

// module definitions

<other statements>

}

一个子模块示例如下：

submodule acme-types {

belongs-to "acme-system" {

prefix "acme";

}

import ietf-yang-types {

prefix "yang";

}

organization "ACME Inc.";

contact

"Joe L. User";

description

"This submodule defines common ACME types.";

revision "2007-06-09" {

description "Initial revision.";

}

// definitions follows...

}

#### belongs-to

belongs-to语句指定submodule所属的module，后面跟上的参数就是module名称。submodule只能由belong-to指定的module或这个指定module的其它submodule进行include，其它module(包括submodule)都不能include。

子模块不可以import它的父模块

### 命名空间

模块中定义的所有节点都属于一个特定的XML命名人空间，由一个URI指定命名空间。一个良好的做法是使模块和命名空间一一对应。例如，

urn:ietf:params:xml:ns:yang:1

### import(include) revision

import或include语句包含一个revision-date子句，所有引入的YANG定义来自要导入或引入的指定revision模块。通过引入特定的模块修订，我们可以独立发布演变中的模块。YANG提供了指定的规则，如何发布进化中的模块并且保持向后的兼容性(=可能会增加新的定义，但不会删除或改变已发布的定义)。下面是一个module的例子：

module acme-module {

namespace "http://acme.example.com/module";

prefix acme;

import "yang-types" {

prefix yang;

}

include "acme-system";

organization "ACME Inc.";

contact joe@acme.example.com;

description "The module for entities

implementing the ACME products";

revision 2007-06-09 {

description "Initial revision.";

}

. . .

}

### YANG文件

YANG的模块和子模块通常保存在文件中，每个文件是一个模块或者子模块。文件名按如下格式：

module-or-submodule-name ['@' revision-date] ( '.yang' / '.yin' )

当YANG文件编译时，如果有import和include语句，它会根据上面的文件名约定去寻找YANG文件。

## YANG数据类型

### 内建数据类型

YANG内建数据类型大部分继承自SIMng的类型，兼容XSD、RelaxNG的类型。

|  |  |  |
| --- | --- | --- |
| 类别 | 类型 | 限制 |
| Integral | Integral {u,}int{8,16,32,64} | range |
| Decimals | decimal64 | range, fraction-digits |
| String | string | length,pattern |
| Enumeration | enumeration | enum |
| Bool and Bits | boolean, bits |  |
| Binary | binary | length |
| References | leafref | path |
| References | identityref | base |
| References | instance-identifier |  |
| Other | union |  |
| Other | empty |  |

### 派生类型

YANG使用typedef语句定义派生类型。如下语句定义了一个派生类型percent，限制类型是内建类型uint8：

typedef percent {

type uint8 {

range "0 .. 100";

}

}

接下的例子定义了一个ip-address类型，它包含了两个派生类型的联合：

typedef ip-address {

type union {

type ipv4-address;

type ipv6-address;

}

}

## YANG数据节点

### leaf

leaf定义的节点只有一个值，没有孩子，只能有一个实例。

一个leaf语句如下：

leaf enabled {

type string;

}

对应的XML样例：

<enabled>false</enabled>

### leaf-list

leaf-list定义一组相同类型的叶节节点，相当于C语言的数组。与leaf区别在于leaf只有一个实例，而leaf-list可以有多个实例。

下面例子定义一个名称为cipher的leaf-list

leaf-list cipher {

type string;

}

对应的XML样例：

<cipher>blowfish-cbc</cipher>

<cipher>3des-cbc</cipher>

### container

container主要定义一个schema树的内部节点，它本身没有任何值和意义，只是作为一系列子节点的父亲存在，只有一个实例。如下container语句定义了一个timeout，它有两个组件，access-timeout和retries

container timeout {

leaf access-timeout {

type uint32;

}

leaf retries {

type uint8;

}

}

对应的XML样例：

<timeout>

<access-timeout>60</access-timeout>

<retries>2</retries>

</timeout>

### list

list描述了一组节点的集合，它像一张数据库表一样，表的每一行用key来标识其主键，有多个实例。如下list语句定义了一个名为user的list：

list user {

key "login-name";

leaf login-name {

type string;

}

leaf full-name {

type string;

}

}

对应的XML样例：

<user>

<login-name>hakanm</login-name>

<full-name>Hakan Millroth</fullname>

</user>

<user>

<login-name>mbj</login-name>

<full-name>Martin Bjorklund</fullname>

</user>

### grouping, use and refine

一个grouping定义一个可以重复使用的节点集合，使用时通过use语句，并可通过refine语句进行改进。如下，定义一个grouping endpoint：

grouping endpoint {

leaf address {

type ip-address;

}

leaf port {

type port-number;

}

}

使用grouping时，通过use语句，它会将grouping定义的节点复制到当前的schema树(如果有refine，并进行更新)。继续上面那个例子，我们重复使用endpoint来定义源节点到目的目的节点的连接。

container connection {

container source {

uses endpoint {

refine port {

default 161;

}

}

}

container destination {

uses endpoint {

refine port {

default 161;

}

}

}

}

这里我们使用可选的refine语句，XML示例如下：

<connection>

<source>

<address>192.168.0.3</address>

<port>8080</port>

</source>

<destination>

<address>192.168.0.4</address>

<port>8080</port>

</destination>

</connection>

### choice

一个choice节点定义了一个可供选择项的集合，每一个选择项都会在某种情况下存在。一个choice由许多分枝，通过case子语句定义。

choice route-distinguisher {

case ip-address-based {

leaf ip-address {

type ipv4-address;

}

leaf ip-address-number {

type uint16;

}

}

case asn32-based {

leaf asn32 {

type uint32;

}

leaf two-byte-number {

type uint16;

}

}

}

一个XML实例：

<asn32>12356789</asn32>

<two-byte-number>2468</two-byte-number>

### presence

presence语句用于提供语义说明，它会附带一个字符串参数，描述模型设计者想要在节点出现时说明什么。

container logging {

presence "Enables logging";

leaf directory {

type string;

default "/var/log/myapplication";

}

}

### anyxml

anyxml表示任何未知的数据定义。

### augment

YANG提供augment语句，扩大一个模块层次，将节点添加到一个已存在的模块或子模块当中。目标结点可以是一个container, list, choice, case, rpc, input, output, notification等。

augment可以是条件的，使用when语句，当特定条件满足时，新节点才会出现。

在下面给出的例子(不带条件的)中，我们对router module中的RIP routing stack进行扩大

module rip\_router {

import router {

prefix r;

}

augment "/r:system/r:router" {

container rip {

leaf version { . . . }

list network-ip { . . . }

list network-ifname { . . . }

list neighbor { . . . }

. . .

}

｝

}

注意import语句，要求指定导入模块的prefix(前缀)。当从导入模块引入对象时，必须使用前缀，例如/r:system/r:router

## YANG引用

### leafref

leafref内建类型，用来引用数据树上的一个叶子实例，通过path参数指定。path使用XPATH，一种类似于unix/linux文件目录的字符串。例如下面这个例子：

leaf interface-name {

type leafref {

path "/interface/name";

}

}

这里的interface-name使用绝对路径参考另一个leaf。接下来的例子使用的是相对路径：

leaf interface-name {

type leafref {

path "../interface/name";

}

}

## 数据约束

YANG提供了若干约束类型来对数据进行约束。

### config

**数据节点既可以表示配置数据，也可以表示状态数据**。而config语句用来区分两者。当config=true时，表示配置数据；config=false表示状态数据。如果config没有指定，默认和父节点一样，如果顶级节点没有指定config语句，默认config=true。

### mandatory

YANG提供了mandatory语句表示一个节点在数据树中必须存在。如下面的例子所示：

leaf host-name {

type string;

mandatory true;

config true;

}

### unique

unique语句用于保证list节点的兄弟间具有唯一的值。这里有一个例子(注意约束是指定两个leaf的值ip和port的联合，必须唯一)：

list server {

key "name";

unique "ip port";

leaf name {

type string;

}

leaf ip {

type ip-address;

}

leaf port {

type port-number;

}

}

### must

must语句用来表示在它所定义的数据结构内，每一个数据节点必须满足约束。下面这样例子表示ip地址不能以192和127开头：

leaf mirrorIp {

type ip-address;

default 0.0.0.0;

must "false() = starts-with(current(), '192')" {

error-message "The mirrorIp is in the forbidden "+

"192.\* network";

}

must "false() = starts-with(current(), '127')" {

error-message "The mirrorIp is in the occupied "+

"127.\* network";

}

}

## notification

notification语句用来定义Notification内容，内容数据的定义同YANG数据定义一样。下面例子使用上文例子定义的connection。

notification link-failure {

description "A link failure has been detected";

container link {

uses connection

}

}

## rpc

rpc语句用来定义NETCONF协议的RPCs，input和output使用YANG数据定义语句。如下例子，建模一个激活软件图片的RPC：

rpc activate-software-image {

input {

leaf image-name {

type string;

}

}

output {

leaf status {

type string;

}

}

}