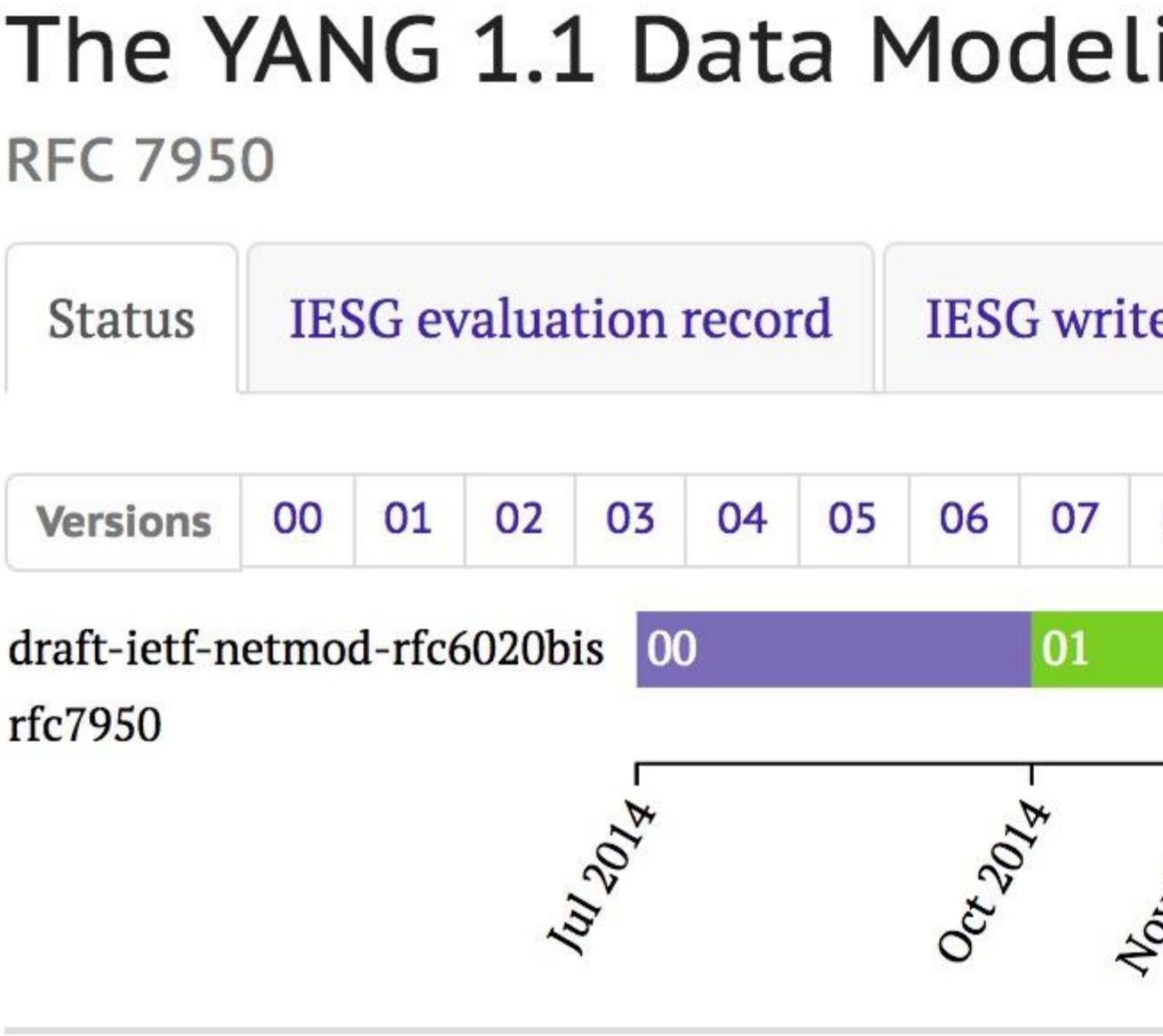
YANG 1.1 数据建模语言

RFC7950 (The YANG 1.1 Data Modeling Language) 中文翻译版本



抽象

YANG 是一种数据建模语言(data modeling language)，用于对配置数据(model configuration data)，状态数据(state data)，远程过程调用(Remote Procedure Calls)和网络管理协议通知(notifications for network management protocols)进行建模。

本文档描述了 YANG 语言版本 1.1 的语法和语义。 YANG 版本 1.1 是 YANG 语言的维护版本，解决了原始规范中的模糊性和缺陷。 YANG 版本 1 中有少量后向不兼容。本文档还指定了到网络配置协议（NETCONF）的 YANG 映射。

本备忘录的状态

这是一个互联网标准跟踪文件。

本文档是 Internet 工程任务组（IETF）的产品。 它代表了 IETF 社区的共识。 已经接受公众评议，并获得互联网工程指导组（IESG）的批准。 有关 Internet 标准的更多信息，请参阅 RFC 7841 的第 2 部分。

有关本文档当前状态的信息，所有勘误表，以及如何提供反馈意见，请访问 http://www.rfc-editor.org/info/rfc7950。

版权声明

1. 介绍

YANG 是一种数据建模语言，最初设计用于模拟由网络配置协议（NETCONF），NETCONF 远程过程调用和 NETCONF 通知 RFC6241 操作的配置和状态数据。自从 YANG 版本 1 RFC6020 发布以来，YANG 已被用于或被提议用于其他协议（例如，RESTCONF 和约束应用协议（CoAP）管理接口 CoMI）。此外，已经提出了不同于 XML 的编码（例如，JSON RFC7951）。
本文档描述了 YANG 语言版本 1.1 的语法和语义。它还描述了如何在可扩展标记语言（XML）XML 中对 YANG 模块中定义的数据模型进行编码，以及如何使用 NETCONF 操作来操作数据。其他协议和编码也是可能的，但不在本规范的范围之内。
在开发 YANG 数据模型方面，YANG-Guidelines 提供了一些指导和建议。
请注意，RFC 6020 这个文件不会被废弃。

1.1. RFC 6020 的变化摘要

这个文件定义了 YANG 语言的版本 1.1。 YANG 版本 1.1 是 YANG 语言的维护版本，解决了原始规范 RFC6020 中的模糊性和缺陷。

以下更改不能与 YANG 版本 1 向后兼容：

- 更改了双引号字符串中转义字符解释的规则。这是从 YANG 版本 1 向后不兼容的变化。当更新 YANG 版本 1 模块为 1.1，并且模块使用现在是非法的字符序列时，必须更改字符串以匹配新规则。详情请参见第 6.1.3 节。
- 没有引号的字符串不能包含任何单引号或双引号字符。这是从 YANG 版本 1 向后不兼容的变化。当更新 YANG 版本 1 模块为 1.1，并且模块使用这样的引号字符时，必须更改字符串以匹配新的规则。详情请参见第 6.1.3 节。
- 在列表键上使“when”和“if-feature”非法。这是从 YANG 版本 1 向后不兼容的变化。当更新 YANG 版本 1 模块为 1.1，并且模块使用这些结构时，必须删除它们以匹配新规则。
- 定义了 YANG 模块中的合法字符。将 YANG 版本 1 模块更新到 1.1 时，必须删除现在非法的任何字符。详情请参阅第 6 章。
- 在内置的“字符串”中使非字符非法。此更改会影响基于 YANG 的协议的运行时行为。

对 YANG 做了以下更改：

- 将 YANG 版本从“1”更改为“1.1”。
- 在“1.1”版本中做了“yang-version”声明。
- 将“if-feature”语法扩展为功能名称上的布尔表达式。
- 允许“比特”(bit)，“枚举”(enum)和“身份”(identity)中包含“if-feature”。
- 允许“refine”中包含“if-feature”。
- 允许“choice”作为简短的“case”陈述（见第 7.9.2 节）。
- 在“pattern”语句中添加了一个新的子句“modifier”（参见第 9.4.6 节）。
- 在“input”，“output”和“notification”中允许包含“must”。
- 在 leafref 中允许包含“require-instance”。
- 允许“import”和“include”中包含“描述”(description)和“引用”(reference)。
- 允许导入模块的多个修订版本。

- 允许“augment”添加有条件的强制节点（见第 7.17 节）。
- 在第 10 节中添加了一组新的 XML 路径语言（XPath）函数。
- 在第 6.4.1 节中阐述了 XPath 上下文的树。
- 在 XPath 表达式中定义了一个 identityref 的字符串值（见 9.10 节）。
- 澄清 typedef 中的 leafrefs 中没有前缀的名字（参见 6.4.1 和 9.9.2 节）。
- 允许从多个基本身份派生身份（见 7.18 和 9.10 节）。

- 允许枚举(enumerations)和位(bits)是子类型的（参见 9.6 和 9.7 节）。
- 允许叶子列表具有默认值（参见第 7.7.2 节）。

- 允许非配置叶子列表中的非唯一值（参见 7.7 节）。

- 在语法中使用区分大小写的字符串的语法（按 RFC7405）。
- 更改了模块发布机制（参见第 5.6.4 节）。
- 更改了子模块中定义的范围规则。子模块现在可以引用所有子模块中的所有定义
- 添加了一个新的语句“action”，用于定义绑定到数据节点的操作。
- 允许通知绑定到数据节点。
- 增加了一个新的数据定义语句“anydata”（见第 7.10 节），当数据可以在 YANG 中建模时，建议使用它来代替“anyxml”。
- 在 unions 中允许类型“empty”和“leafref”。
- 允许在键中键入“empty”。
- 删除了标识符不能以字符“xml”开头的限制。

对 NETCONF 映射进行了以下更改：

- 服务器通过使用 ietf-yang-library RFC7895 来发布对 YANG 1.1 模块的支持，而不是将它们列为<hello>消息中的能力。

2. 关键词

关键词“必须”(MUST)，“不得”(MUST NOT)，“要求”(REQUIRED)，“应该”(SHALL)，“不应该”(SHALL NOT)，“应该”(SHOULD)，“不应该”(SHOULD NOT")，“推荐”(RECOMMENDED)，“不推荐”(NOT RECOMMENDED)，“可能”(MAY)和“可选”(OPTIONAL)“在本文档中的解释如 BCP 14 [RFC2119]中所述。

3. 术语

本文件中使用以下术语：

- 操作(action)：为数据树中的节点定义的操作。
- anydata：一个数据节点，可以包含一个未知的节点集合，除了 anyxml 以外，可以由 YANG 建模。
- anyxml：一个数据节点，可以包含一个未知的 XML 数据块。
- 扩充(augment)：将新的模式节点添加到先前定义的模式节点。
- 基本类型(base type)：衍生派生类型的类型，可以是内置类型，也可以是其他派生类型。
- 内置类型(built-in type)：YANG 语言定义的 YANG 数据类型，如 uint32 或 string。
- 选择(choice)：一个模式节点，其中只有一种标识的可选方案是有效的。

- 客户端(client)：通过某种网络管理协议，可以访问服务器上 YANG 定义数据的实体。
- 一致性(conformance)：衡量服务器遵循数据模型的准确程度。
- 容器(container)：一个内部数据节点，它存在于数据树的大多数实例中。 一个容器只有子节点，没有值。
- 数据定义语句(data definition statement)：定义新数据节点的语句。“container”，“leaf”，“leaf-list”，“list”，“choice”，“case”，“augment”，“uses”，“anydata”和“anyxml”中的一个。
- 数据模型(data model)：数据模型描述如何表示和访问数据。
- 数据节点(data node)：模式树中可以在数据树中实例化的节点。container，leaf，leaf-list，list，anydata 和 anyxml 之一。
- 数据树(data tree)：用 YANG 建模的任何数据的实例化树，例如配置数据，状态数据，组合配置和状态数据，RPC 或操作输入，RPC 或操作输出或通知。
- 派生类型(derived type)：派生自内建类型（如 uint32）或其他派生类型的类型。
- 扩展(extension)：扩展将非 YANG 语义附加到语句中。 “extension”语句定义了新的语句来表达这些语义。
- 特征(feature)：用于将模型的一部分标记为可选的机制。定义可以使用功能名称进行标记，并仅在支持该功能的服务器上有效。
- 分组(grouping)：一组可重复使用的模式节点，可以在模块中本地使用，也可以从其他模块中使用。 “grouping”语句不是数据定义语句，因此不会在模式树中定义任何节点。
- 标识符(identifier)：用于按名称标识不同种类 YANG 项目的字符串。
- 身份(identity)：一个全球唯一的，抽象的，无类型的名字。
- 实例标识符(instance identifier)：用于标识数据树中的特定节点的机制。
- 内部节点(interior node)：层次结构内不是叶节点的节点。
- 叶节点(leaf)：数据树中至多存在一个实例的数据节点。叶子有一个值，但没有子节点。
- 叶列表(leaf-list)：与叶节点类似，但定义了一组唯一可识别的节点，而不是单个节点。每个节点都有一个值，但没有子节点。
- 列表(list)：数据树中可能存在多个实例的内部数据节点。一个列表没有任何价值，而是一组子节点。
- 强制节点(mandatory node)：强制节点是以下之一：
  - 带有值为“true”的“mandatory”语句的 leaf，choice，anydata 或 anyxml 节点。
  - 带有大于零值的“min-elements”语句的列表或叶列表节点。
  - 没有“presence”语句的容器节点，并且至少有一个强制节点作为子节点。
- 模块(module)：YANG 模块定义模式节点的层次结构。通过其定义和从其他地方导入或包含的定义，模块是自包含的和“可编译的”。
- 不存在容器(non-presence container)：一个没有其自身意义的容器，仅存在于包含子节点的容器中。
- 存在容器(presence container)：存在容器的容器本身具有某种意义。
- RPC：远程过程调用。
- RPC 操作(RPC operation)：特定的远程过程调用。

- 模式节点(schema node)：模式树中的节点。 action，container，leaf，leaf-list，list，choice，case，rpc，input，output，notification，anydata 和 anyxml 中的一个。
- 模式节点标识符(schema node identifier)：用于标识模式树中特定节点的机制。
- 模式树(schema tree)：模块中指定的定义层次结构。
- 服务器(server)：通过某种网络管理协议，提供对客户端访问 YANG 定义数据的实体。
- 服务器偏差(server deviation)：服务器忠实地执行模块的失败。
- 子模块(submodule)：部分模块定义，用于为模块提供派生类型，分组，数据节点，RPC，操作和通知。一个 YANG 模块可以由多个子模块构成。
- 顶层数据节点(top-level data node)：一个数据节点，在这个节点和“模块”或“子模块”语句之间没有其他数据节点。
- 使用(uses)：“uses”语句用于实例化“分组”语句中定义的一组模式节点。实例化的节点可以被改进和增加以适应任何特定的需求。
- 值空间(value space)：对于数据类型;数据类型允许的一组值。对于叶或叶列表实例;它的数据类型的值空间。
  以下术语在 RFC6241 中定义：

- 配置数据(configuration data)
- 配置数据存储(configuration datastore)
- 数据存储(datastore)
- 状态数据(state data)
  当用 YANG 建模时，数据存储被实现为一个实例化的数据树。
  在使用 YANG 建模时，配置数据存储实现为具有配置数据的实例化数据树。

3.1. 关于例子的一个注释

在整个文件中，有许多 YANG 的例子。
这些例子是为了说明某些特征，而不是完整的，有效的 YANG 模块。

4. YANG 概述

这个部分是为了给第一次阅读本文档的读者提供一个关于 YANG 的高层次的概述。

4.1. 功能概述

YANG 是最初设计用于为 NETCONF 协议建模的语言。 YANG 模块定义了可用于基于 NETCONF 的操作（包括配置，状态数据，RPC 和通知）的数据层次结构。这允许在 NETCONF 客户端和服务器之间发送的所有数据的完整描述。虽然不在本规范的范围之内，但是也可以使用除 NETCONF 以外的协议。
YANG 将数据的分层组织模型化为一个树，其中每个节点都有一个名称，或者一个值或一组子节点。YANG 提供了对节点的清晰简洁的描述，以及这些节点之间的交互。
YANG 将数据模型组织成模块和子模块。模块可以从其他外部模块导入定义，并可以包含子模块的定义。可以增加层次结构，允许一个模块将数据节点添加到另一个模块中定义的层次结构中。这种增加可以是有条件的，只有在满足某些条件的情况下才会出现新的节点。
YANG 数据模型可以描述对数据执行的约束，根据层次结构中其他节点的存在或值限制节点的存在或值。这些约束可以由客户端或服务器强制执行。
YANG 定义了一组内置的类型，并且有一个类型机制，通过它可以定义附加的类型。派生类型可以使用像客户机或服务器强制执行的范围或模式限制等机制来限制其基本类型的有效值集合。他们还可以定义使用派生类型的用法约定，例如包含主机名的基于字符串的类型。
YANG 允许定义可重复使用的节点组。这些分组的使用可以改进或增加节点，从而使节点适合其特定需求。派生类型和分组可以在一个模块中定义，并在相同的模块或导入它的其他模块中使用。
YANG 数据层次结构包括定义列表，其中列表条目通过将它们彼此区分的键来标识。这样的列表可以被定义为由用户排序或由系统自动排序。对于用户排序列表，操作定义为操作列表条目的顺序。
YANG 模块可以翻译成称为 YANG Independent Notation（YIN）（第 13 节）的等效 XML 语法，允许使用 XML 解析器的应用程序和可扩展样式表语言转换（XSLT）脚本在模型上运行。从 YANG 到 YIN 的转换在语义上是无损的，所以 YIN 中的内容可以回到 YANG 中。
YANG 是一个可扩展的语言，允许标准组织，供应商和个人定义扩展。语句语法允许这些扩展以自然的方式与标准的 YANG 语句共存，而在 YANG 模块中的扩展足以让读者注意到它们。
YANG 拒绝解决所有可能的问题的倾向，限制了问题空间，允许表达网络管理协议（如 NETCONF）的数据模型，而不是任意的 XML 文档或任意的数据模型。
YANG 尽可能保持与简单网络管理协议（SNMP）SMIv2（管理信息结构版本 2 [RFC2578] [RFC2579]）的兼容性。基于 SMIv2 的 MIB 模块可以自动翻译成 YANG 模块进行只读访问[RFC6643]。然而，YANG 不关心从 YANG 到 SMIv2 的逆向翻译。

4.2. 语言概述

本节介绍了 YANG 中使用的一些重要的构造，这将有助于在后面的章节中理解语言细节。

4.2.1. 模块和子模块

YANG 数据模型在模块中定义。一个模块包含相关定义的集合。
一个模块包含三种类型的语句：模块头(module header)语句，“修订”(revision)语句和定义(definition)语句。模块头部语句描述模块并提供关于模块本身的信息，“修订”语句提供关于模块历史的信息，定义语句是定义数据模型的模块的主体。
服务器可以实现多个模块，允许相同数据的多个视图或服务器数据的不相交子部分的多个视图。或者，服务器可以只实现一个定义所有可用数据的模块。

基于模块设计者的需求，模块可以将其部分定义分成子模块。无论子模块的存在或大小如何，外部视图都是单个模块的外观。

“导入”(import)语句允许模块或子模块引用在其他模块中定义的定义。
在模块中使用“包括”(include)语句来标识属于它的每个子模块。

4.2.2. 数据建模基础

YANG 定义了数据建模的四种主要类型的数据节点。 在下面的每个小节中，这些示例都显示了 YANG 语法以及相应的 XML 编码。 YANG 语句的语法在 6.3 节中定义。

4.2.2.1. 叶节点(Leaf Nodes)

叶子实例包含简单的数据，如整数或字符串。 它只有一个特定类型的值，没有子节点。

YANG 例子：

```
leaf host-name {
    type string;
    description
        "Hostname for this system.";
}
```

XML 编码示例：

```
<host-name>my.example.com</host-name>
```

第 7.6 节详细介绍了“leaf”声明。

4.2.2.2. 叶列表节点(Leaf-List Nodes)

叶列表定义了特定类型的值序列。

YANG 例如：

```
leaf-list domain-search {
  type string;
```

```
  description
    "List of domain names to search.";
}
```

XML 编码示例：

```
<domain-search>high.example.com</domain-search>
<domain-search>low.example.com</domain-search>
<domain-search>everywhere.example.com</domain-search>
```

第 7.7 节介绍了“leaf-list”声明。

4.2.2.3. 容器节点(Container Nodes)

一个容器用于分组子树中的相关节点。 一个容器只有子节点，没有值。 容器可以包含任何类型的任何数量的子节点（叶子，列表，容器，叶子列表，动作和通知）。

YANG 例如：

```
container system {
  container login {
    leaf message {
      type string;
      description
        "Message given at start of login session.";
    }
  }
}
```

XML 编码示例：

```
<system>
  <login>
    <message>Good morning</message>
  </login>
</system>
```

第 7.5 节介绍了“container”声明。

4.2.2.4. 列表节点(List Nodes)

列表定义了一系列列表条目。每个条目就像一个容器，如果它定义了任何关键的叶子，它就被其关键叶子的值唯一标识。列表可以定义多个关键叶子，并且可以包含任何类型的任何数量的子节点（包括树叶，列表，容器等）。

YANG 例如：

```
list user {
  key "name";
  leaf name {
    type string;
  }
```

```
  leaf full-name {
    type string;
  }
  leaf class {
    type string;
  }
}
```

XML 编码示例：

```
<user>
  <name>glocks</name>
  <full-name>Goldie Locks</full-name>
  <class>intruder</class>
</user>
<user>
  <name>snowey</name>
  <full-name>Snow White</full-name>
  <class>free-loader</class>
</user>
<user>
  <name>rzell</name>
  <full-name>Rapun Zell</full-name>
  <class>tower</class>
</user>
```

第 7.8 节介绍了“`list`”声明。

4.2.2.5. 示例模块

这些语句被组合来定义模块：

```
// Contents of "example-system.yang"
module example-system {
    yang-version 1.1;
    namespace "urn:example:system";
    prefix "sys";

    organization "Example Inc.";
    contact "joe@example.com";
    description
        "The module for entities implementing the Example system.";

    revision 2007-06-09 {
        description "Initial revision.";
    }
```

```
    container system {
        leaf host-name {
            type string;
            description
                "Hostname for this system.";
        }

        leaf-list domain-search {
            type string;
            description
                "List of domain names to search.";
        }

        container login {
            leaf message {
            type string;
            description
                "Message given at start of login session.";
            }
            list user {
                key "name";
                leaf name {
                    type string;
                }
                leaf full-name {
                    type string;
                }
                leaf class {
                    type string;
                }
            }
        }
    }
}
```

4.2.3. 配置和状态数据(Configuration and State Data)

YANG 可以根据“config”语句为状态数据和配置数据建模。 当一个节点被标记为“config false”时，其子层被标记为状态数据。 如果标记为“config true”，

则其子层被标记为配置数据。 报告父容器，列表和关键叶子，给出状态数据的上下文。
在这个例子中，为每个接口定义了两个叶子，一个配置的速度和一个观察到的速度。

```
list interface {
  key "name";
  config true;

  leaf name {
    type string;
  }
  leaf speed {
    type enumeration {
      enum 10m;
      enum 100m;
      enum auto;
    }
  }
  leaf observed-speed {
    type uint32;
    config false;
  }
}
```

第 7.21.1 节介绍了“config”语句。

4.2.4. 内置类型

YANG有一套内置的类型，类似于许多编程语言，但由于网络管理的特殊要求，有一些不同之处。 下表总结了第 9 节讨论的内置类型：

名称	描述
binary	任何二进制数据
bits	一组比特(bits)或标志(flags)
boolean	"true" 或 "false"
decimal64	64 位有符号十进制数字
empty	一个叶子，没有任何值

名称	描述
enumeration	枚举的一组字符串之一
identityref	对抽象身份的引用
int8	8 位有符号整数
int16	16 位有符号整数
int32	32 位有符号整数
int64	64 位有符号整数
leafref	对叶子实例的引用
string	一个字符串
uint8	8 位无符号整数
uint16	16 位无符号整数
uint32	32 位无符号整数
uint64	64 位无符号整数
union	成员类型的选择

下面是原始内置类型总结表

```
+---------------------+-------------------------------------+
| Name                | Description                         |
+---------------------+-------------------------------------+
| binary              | Any binary data                      |
| bits                | A set of bits or flags               |
| boolean             | "true" or "false"                    |
| decimal64           | 64-bit signed decimal number         |
| empty               | A leaf that does not have any value |
| enumeration         | One of an enumerated set of strings |
| identityref         | A reference to an abstract identity |
| instance-identifier | A reference to a data tree node     |
| int8                | 8-bit signed integer                |
| int16               | 16-bit signed integer                |
| int32               | 32-bit signed integer                |
| int64               | 64-bit signed integer                |
| leafref             | A reference to a leaf instance      |
| string              | A character string                  |
| uint8               | 8-bit unsigned integer               |
| uint16              | 16-bit unsigned integer              |
| uint32              | 32-bit unsigned integer              |
```

```
| uint64              | 64-bit unsigned integer             |
| union               | Choice of member types              |
+---------------------+-------------------------------------+
```

“type”声明在第 7.4 节中介绍。

4.2.5. 派生类型（typedef）(Derived Types (typedef))

YANG 可以使用“typedef”语句从基本类型定义派生类型。 基本类型可以是内置类型或派生类型，允许派生类型的层次结构。
派生类型可以用作“类型”语句的参数。

YANG 示例：

```
typedef percent {
  type uint8 {
    range "0 .. 100";
  }
}

leaf completed {
  type percent;
}
```

XML 编码示例：

```
<completed>20</completed>
```

第 7.3 节介绍了“typedef”语句。

4.2.6. 可重复使用的节点组（分组）(Reusable Node Groups (grouping))

可以使用“grouping”语句将节点组组合成可重复使用的集合。 分组定义了一组使用“uses”语句实例化的节点。
YANG 示例:

```
grouping target {
    leaf address {
      type inet:ip-address;
      description "Target IP address.";
    }
    leaf port {
      type inet:port-number;
```

```
      description "Target port number.";
    }
}
container peer {
  container destination {
    uses target;
  }
}
```

XML 编码示例：

```
<peer>
  <destination>
    <address>2001:db8::2</address>
    <port>830</port>
  </destination>
</peer>
```

分组可以根据使用情况进行细化，从而允许某些语句被覆盖。 在这个例子中，描述被细化：

```
container connection {
  container source {
    uses target {
      refine "address" {
        description "Source IP address.";
      }
      refine "port" {
        description "Source port number.";
      }
    }
  }
  container destination {
    uses target {
      refine "address" {
        description "Destination IP address.";
      }
      refine "port" {
        description "Destination port number.";
      }
    }
  }
}
```

第 7.12 节介绍了“grouping”的声明。

4.2.7. 选择(Choices)

YANG 允许数据模型使用“choice”和“case”语句将不兼容的节点分隔为不同的选项。 “choice”语句包含一组“case”语句，用于定义不能一起出现的模式节点集合。 每个“case”可能包含多个节点，但每个节点可能只出现在“choice”下的一个“case”中。

choice 和 case 节点仅出现在模式树(schema tree)中，而不出现在数据树中。概念模式之外不需要额外的层次结构。 一个 case 的存在是由一个或多个节点的存在表示。

由于只有一个选择的情况在任何时候都是有效的，所以当在数据树中创建一个节点时，所有其他情况下的所有节点都被隐式删除。 服务器处理约束的执行，防止配置中存在不兼容。

YANG 示例：

```
container food {
  choice snack {
    case sports-arena {
      leaf pretzel {
        type empty;
      }
      leaf beer {
        type empty;
      }
    }
    case late-night {
      leaf chocolate {
        type enumeration {
          enum dark;
          enum milk;
          enum first-available;
        }
      }
    }
  }
}
```

XML 编码示例：

```
<food>
  <pretzel/>
  <beer/>
</food>
```

第 7.9 节介绍了“choice”的声明。

4.2.8. 扩展数据模型（扩充）(Extending Data Models (augment))

YANG 允许模块将其他节点插入数据模型，包括当前模块（及其子模块）和外部模块。 例如，这对于供应商以可互操作的方式向标准数据模型添加供应商特定的参数非常有用。

“augment”语句定义插入新节点的数据模型层次结构中的位置，“when”语句定义新节点有效时的条件。

当服务器实现一个包含“augment”语句的模块时，这意味着扩充模块的服务器实现包含附加节点。

YANG 示例：

```
augment /system/login/user {
  when "class != 'wheel'";
  leaf uid {
    type uint16 {
      range "1000 .. 30000";
    }
  }
}
```

这个例子定义了一个“uid”节点，只有当用户的“class”不是“wheel”时才有效。

如果模块扩充另一个模块，则添加到编码中的 XML 元素位于扩充模块的命名空间中。 例如，如果上面的扩充是在一个前缀为“other”的模块中，那么 XML 将如下所示：

XML 编码示例：

```
<user>
  <name>alicew</name>
  <full-name>Alice N. Wonderland</full-name>
  <class>drop-out</class>
  <other:uid>1024</other:uid>
</user>
```

第 7.17 节介绍了“augment”声明。

4.2.9. 操作定义

YANG 允许定义的操作。 使用 YANG 数据定义语句对操作名称，输入参数和输出参数进行建模。 模块顶层的操作用“rpc”语句定义。 操作也可以绑定到容器或列表数据节点。 这些操作用“操作”语句来定义。

YANG 顶层操作示例：

```
rpc activate-software-image {
```

```
  input {
    leaf image-name {
      type string;
    }
  }
  output {
    leaf status {
      type string;
    }
  }
}
```

NETCONF XML 示例：

```
 <rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <activate-software-image xmlns="http://example.com/system">
    <image-name>example-fw-2.3</image-name>
  </activate-software-image>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <status xmlns="http://example.com/system">
    The image example-fw-2.3 is being installed.
  </status>
</rpc-reply>
```

YANG 绑定到列表数据节点的操作示例：

```
list interface {
  key "name";

  leaf name {
    type string;
  }

  action ping {
    input {
      leaf destination {
        type inet:ip-address;
      }
    }
    output {
      leaf packet-loss {
        type uint8;
      }
    }
```

```
  }
}
```

NETCONF XML 示例:

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <interface xmlns="http://example.com/system">
      <name>eth1</name>
      <ping>
        <destination>192.0.2.1</destination>
      </ping>
    </interface>
  </action>
</rpc>

<rpc-reply message-id="102"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
           xmlns:sys="http://example.com/system">
  <sys:packet-loss>60</sys:packet-loss>
</rpc-reply>
```

第 7.14 节介绍“rpc”声明，第 7.15 节介绍“action”声明。

4.2.10. 通知定义

YANG 允许定义通知。 YANG 数据定义语句用于模拟通知的内容。
YANG 示例：

```
notification link-failure {
  description
    "A link failure has been detected.";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf if-admin-status {
    type admin-status;
  }
  leaf if-oper-status {
    type oper-status;
  }
}
```

NETCONF XML 示例：

```
<notification
    xmlns="urn:ietf:params:netconf:capability:notification:1.0">
  <eventTime>2007-09-01T10:00:00Z</eventTime>
  <link-failure xmlns="urn:example:system">
    <if-name>so-1/2/3.0</if-name>
    <if-admin-status>up</if-admin-status>
    <if-oper-status>down</if-oper-status>
  </link-failure>
</notification>
```

第 7.16 节介绍了“notification”的声明。

5. 语言概念

5.1. 模块和子模块

该模块是 YANG 定义的基本单位。一个模块定义一个单一的数据模型。一个模块也可以增加一个现有的数据模型和其他节点。
子模块是为模块提供定义的部分模块。模块可以包括任意数量的子模块，但是每个子模块可以仅属于一个模块。

YANG 模块和子模块的开发人员建议选择其模块名称，与标准模块或其他企业模块冲突的可能性较低，例如，使用企业或组织名称作为模块名称的前缀。
在服务器中，所有模块名称必须是唯一的。
一个模块使用“include”语句来列出其所有的子模块。属于该模块的模块或子模块可以引用模块中包含的定义和模块包含的所有子模块。
模块或子模块使用“import”语句来引用外部模块。模块或子模块中的语句可以使用“import”语句中指定的前缀引用外部模块中的定义。
为了向后兼容 YANG 版本 1，子模块可以使用“include”语句来引用其模块中的其他子模块，但是在版本 1.1 中这不是必需的。子模块可以引用它所属的模块和模块包含的所有子模块中的任何定义。子模块不得包含与其模块包含的修订版不同的其他子模块版本。
模块或子模块不能包含其他模块的子模块，子模块不能导入自己的模块。

“import”和“include”语句用于从其他模块中提供定义：

- 要使模块或子模块引用外部模块中的定义，必须导入外部模块。
- 模块必须包含所有的子模块。

- 属于该模块的模块或子模块可以在模块中引用定义，也可以在模块中包含所有子模块。

  不得有任何引入环形链。例如，如果模块“a”导入模块“b”，“b”不能导入“a”。
  引用外部模块中的定义时，必须使用本地定义的前缀，后跟冒号（“:”），然后是外部标识符。对本地模块中定义的引用可以使用前缀表示法。由于内置数据类型不属于任何模块且不含前缀，因此对内置数据类型（例如 int32）的引用不能使用前缀表示法。定义参考的语法由第 14 节中的“identifier-ref”规则正式定义。

5.1.1. 导入并包含修订

发布的模块随着时间的推移而独立发展为了实现这一发展，可以使用特定的修订版导入模块。最初，模块在写入模块时导入当前其他模块的修订版。随着未来版本导入模块的发布，导入模块不受影响，其内容不变。当模块的作者准备移动到最近发布的导入模块的修订版时，模块将被重新发布，并带有更新的“import”语句。通过重新发布新版本，作者明确表示接受导入模块的任何更改。
对于子模块，这个问题是相关的，但更简单。包含子模块的模块或子模块可以指定包含的子模块的修订版本。如果某个子模块发生更改，则需要更新包含该子模块的任何模块或子模块以引用新的修订版本。

例如，模块“b”导入模块 a。

```
module a {
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

  revision 2008-01-01 { ... }
  grouping a {
    leaf eh { .... }
  }
}

module b {
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";

  import a {
    prefix "p";
    revision-date 2008-01-01;
```

```
  }

  container bee {
    uses p:a;
  }
}
```

当“a”的作者发布一个新的修订版本时，对于“b”的作者来说，这些变化可能是不可接受的。如果新的修订是可以接受的，那么“b”的作者可以在“import”语句中重新发布修订版本。

如果一个模块没有通过特定的修订导入，则未定义使用哪个修订。

5.1.2. 模块层次结构

YANG 允许在多个层次结构中对数据进行建模，其中数据可能有多个顶级节点。模块中的每个顶级数据节点定义一个单独的层次结构。具有多个顶级节点的模型有时很方便，并且得到了 YANG 的支持。

5.1.2.1. NETCONF XML 编码

NETCONF 能够在<config>和<data>元素中携带任何 XML 内容作为有效载荷。 YANG 模块的顶层节点在这些元素内以任何顺序编码为子元素。 这种封装保证了相应的 NETCONF 消息总是格式良好的 XML 文档。

例如，一个实例：

```
module example-config {
  yang-version 1.1;
  namespace "urn:example:config";
  prefix "co";

  container system { ... }
  container routing { ... }
}
```

在 NETCONF 中编码为:

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <!-- system data here -->
      </system>
      <routing xmlns="urn:example:config">
```

```
        <!-- routing data here -->
      </routing>
    </config>
  </edit-config>
</rpc>
```

5.2. 文件布局

YANG 模块和子模块通常存储在文件中，每个文件包含一个“模块”或“子模块”语句。 文件的名字应该是这样的形式：

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

“module-or-submodule-name”是模块或子模块的名称，可选的“revision-date”是模块或子模块的最新版本，由“revision”语句定义（见第 7.1.9 节）。
文件扩展名“.yang”表示文件的内容用 YANG 语法编写（第 6 节），“.yin”表示文件的内容用 YIN 语法编写（第 13 节）。
YANG 解析器可以通过这个约定找到导入的模块和子模块。

5.3. XML 名称空间

所有的 YANG 定义在一个模块中指定。 每个模块绑定到独特的 XML 名称空间[XML-NAMES]，这是一个全球唯一的 URI [RFC3986]。 NETCONF 客户端或服务器在数据的 XML 编码过程中使用名称空间。
在 RFC 流中发布的模块的 XML 名称空间[RFC4844]必须由 IANA 分配; 参见[RFC6020]中的第 14 节。
私有模块的 XML 名称空间由拥有该模块的组织分配，没有中央注册表。 名称空间 URI 必须被选择，以便它们不能与标准或其他企业名称空间冲突 - 例如，通过在名称空间中使用企业或组织名称。
第 7.1.3 节介绍了“命名空间”声明。

5.3.1. YANG XML 命名空间

YANG 为 NETCONF <edit-config>操作，<error-info>内容和<action>元素定义了一个 XML 名称空间。 这个命名空间的名字是“urn:ietf:params:xml:ns:yang:1”。

5.4. 解决分组，类型和标识名称

分组(grouping)，类型(type)和标识名称(identity names)在其定义的上下文中解析，而不是在其使用的上下文中解析。 分组，类型定义和标识的用户不需要导入模块或包含子模块来满足原始定义所做的所有引用。 这就像传统编程语言中的静态范围一样。
例如，如果一个模块定义了一个引用类型的分组，那么在第二个模块中使用该分组时，该类型将在原始模块的上下文中解析，而不是在第二个模块的上下文中解析。 如果两个模块都定义了类型，则没有歧义。

5.5. 嵌套 Typedef 和分组

类型定义和分组可能会出现在许多 YANG 语句之下，允许它们在它们出现的语句层次结构的词汇范围内。这使得类型和分组可以在使用位置附近进行定义，而不是将它们放在层次结构的顶层。靠近增加了可读性。
范围界定还允许定义类型而不用担心不同子模块中的类型之间的命名冲突。可以指定类型名称，而无需添加旨在防止大型模块中的名称冲突的前导字符串。

最后，范围设定允许模块作者将模块或子模块的类型和分组保留为私有模式，以防止其重用。由于只有顶级类型和分组（即，作为“模块”或“子模块”语句的子分支出现的那些）才能在模块或子模块之外使用，开发人员可以更好地控制其模块呈现给外部世界，支持隐藏内部信息的必要性，并保持与外部世界共享的内容与私有内容之间的界限。

作用域定义不能在更高的范围内影射定义。如果语句层次结构中的更高级别具有匹配标识符的定义，则无法定义类型或分组。

对前缀类型或分组的引用，或者使用当前模块的前缀的引用，通过在每个祖先语句的直接子语之间查找匹配的“typedef”或“grouping”语句来解决。

5.6. 一致性

与模型的一致性是衡量服务器遵循模型的准确程度。 一般而言，服务器负责忠实地实现该模型，允许应用程序对待实现该模型的服务器。 与模型的偏差会降低模型的实用性，并增加使用模型的应用程序的脆弱性。

YANG 建模者有三种一致性机制：

- 模型的基本行为
- 作为模型一部分的可选功能
- 与模型的偏差

我们将按顺序考虑每一个。

5.6.1. 基本行为

该模型定义了一个基于 YANG 的客户端和服务器之间的契约；这个契约允许双方相信另一方知道建模数据背后的语法和语义。 YANG 的能力在于这个契约的能力。

5.6.2. 可选功能

在许多模型中，建模者将允许模型的各个部分是有条件的。服务器控制模型的这些条件部分是否受支持或对该特定服务器有效。

例如，一个系统日志数据模型可能会选择包含在本地保存日志的功能，但是建模人员会意识到，只有当服务器具有本地存储时才可能这样做。如果没有本地存储，应用程序不应该通知服务器保存日志。

YANG 使用称为“特征”的构造支持这个条件机制。特征为建模者提供了一种以服务器控制的方式使模块的一部分有条件的机制。该模型可以表达不是普遍存在于所有服务器中的结构。这些功能包含在模型定义中，允许一致的视图，并允许应用程序了解哪些功能是受支持的，并定制到服务器的行为。
一个模块可以声明任何数量的特征，用简单的字符串来标识，并且可以根据这些特征使模块的一部分是可选的。如果服务器支持某项功能，则模块的相应部分对该服务器有效。如果服务器不支持该功能，则模块的这些部分无效，并且应用程序应该相应地运行。

功能是使用“feature”语句定义的。 “if-feature”语句记录模块中对该功能有条件的定义。
更多细节可参见第 7.20.1 节。

5.6.3. 偏差

在理想的情况下，所有的服务器将被要求完全按照定义来实现模型，并且不允许与模型的偏差。但在现实世界中，服务器通常无法或不能按照书面实现模型。对于基于 YANG 的自动化来处理这些服务器偏差，服务器必须有一种机制来通知应用这些偏差的具体情况。
例如，一个 BGP 模块可能允许任意数量的 BGP 对等体，但是一个特定的服务器可能仅支持 16 个 BGP 对等体。配置第 17 个节点的任何应用程序都将收到错误消息。虽然错误可能足以让应用程序知道它不能添加另一个对等体，但是如果应用程序事先知道这个限制并且可以防止用户启动不能成功的路径，那将会好得多。

服务器偏差是使用“deviation”语句来声明的，该语句以一个字符串作为参数，标识模式树中的一个节点。声明的内容详细说明了模块中定义的服务器实现偏离合同的方式。
更多细节可参见第 7.20.3 节。

5.6.4. 在 NETCONF 中公布一致性信息

本文档定义了用于通告一致性信息的以下机制。其他机制可以由未来的规范来定义。

NETCONF 服务器必须宣布它实现了模块（见第 5.6.5）通过实施 YANG 模块“ieft-yang-library”在[RFC7895]中定义和列出所有实现的模块中的“/modules-state/module”列表。
服务器还必须在<hello>消息中宣告以下功能（换行符和空格仅用于格式化）：

```
urn:ietf:params:netconf:capability:yang-
library:1.0?revision=<date>&module-set-id=<id>
```

参数“revision”与服务器实现的“ietf-yang-library”模块的修改日期具有相同的值。这个参数必须存在。
参数“module-set-id”与“ietf-yang-library”中的叶子“/modules-state/module-set-id”具有相同的值。这个参数必须存在。
通过这种机制，客户端可以缓存服务器支持的模块，只有在<hello>消息中的“module-set-id”值发生变化时才更新缓存。

5.6.5. 实现一个模块

如果服务器实现了模块的数据节点(data nodes)，RPCs，操作(actions)，通知(notifications)和偏差(deviations)，则服务器实现一个模块。
一个服务器不能实现一个以上的模块版本。

如果服务器实现了导入模块 B 的模块 A，并且 A 在服务器支持的“augment”或“path”语句中使用了来自 B 的任何节点，则服务器必须实现模块 B 的修订版，该模块具有定义的这些节点。这与模块 B 是否被修改导入无关。
如果一个服务器实现了一个模块 A，它导入一个模块 C 而没有指定模块 C 的修改日期，而服务器没有实现 C（例如，如果 C 只定义了一些类型定义），服务器必须在“/modules-state/module“列表从”ietf-yang-library“[RFC7895]，它必须将叶子”一致性类型(conformance-type)“设置为”import“模块。

如果服务器在“ietf-yang-library”的“/modules-state/module”列表中列出了一个模块 C，并且还有其他的模块 Ms 列出了导入 C 而没有指定模块 C 的修改日期，那么服务器必须使用 C 列出的最新修订版 C 的定义。
这些规则的原因是客户端需要能够知道在服务器中实现的所有叶子和叶子列表的具体数据模型结构和类型。

例如，对于这些模块：

```
module a {
  yang-version 1.1;
  namespace "urn:example:a";
  prefix "a";

  import b {
    revision-date 2015-01-01;
  }
  import c;

  revision 2015-01-01;

  feature foo;

  augment "/b:x" {
    if-feature foo;

    leaf y {
      type b:myenum;
    }
  }

  container a {
    leaf x {
      type c:bar;
    }
  }
}

module b {
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";

  revision 2015-01-01;

  typedef myenum {
```

```
    type enumeration {
      enum zero;
    }
  }

  container x {
  }
}

module b {
  yang-version 1.1;
  namespace "urn:example:b";
  prefix "b";

  revision 2015-04-04;
  revision 2015-01-01;

  typedef myenum {
    type enumeration {
      enum zero; // added in 2015-01-01
      enum one;  // added in 2015-04-04
    }
  }

  container x {  // added in 2015-01-01
    container y; // added in 2015-04-04
  }
}

module c {
  yang-version 1.1;
  namespace "urn:example:c";
  prefix "c";

  revision 2015-02-02;

  typedef bar {
    ...
  }
}

module c {
  yang-version 1.1;
  namespace "urn:example:c";
```

```
  prefix "c";

  revision 2015-03-03;
  revision 2015-02-02;

  typedef bar {
    ...
  }
}
```

执行模块“a”的版本“2015-01-01”并支持功能“foo”的服务器可以实现模块“b”的版本“2015-01-01”或“2015-04-04”。 由于“b”是通过修改导入的，所以叶“/b:x /a:y”的类型是相同的，而不管服务器实现哪个版本的“b”。

实现模块“a”但不支持“foo”功能的服务器不必实现模块“b”。

执行模块“a”的版本“2015-01-01”的服务器选择模块“c”的任何版本，并将其列在“ietf-yang-library”的“/modules-state/module”列表中。

以下 XML 编码示例显示了实现模块“a”的服务器的“/modules-state/module”列表的有效数据：

```
<modules-state
    xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-library">
  <module-set-id>ee1ecb017370cafd</module-set-id>
  <module>
    <name>a</name>
    <revision>2015-01-01</revision>
    <namespace>urn:example:a</namespace>
    <feature>foo</feature>
    <conformance-type>implement</conformance-type>
  </module>
  <module>
    <name>b</name>
    <revision>2015-04-04</revision>
    <namespace>urn:example:b</namespace>
    <conformance-type>implement</conformance-type>
  </module>
  <module>
    <name>c</name>
    <revision>2015-02-02</revision>
    <namespace>urn:example:c</namespace>
    <conformance-type>import</conformance-type>
  </module>
</modules-state>
```

5.7. 数据存储修改

数据模型可以允许服务器以不通过网络管理协议消息明确定向的方式来改变配置数据存储。 例如，数据模型可以定义在客户端不提供系统生成值时分配的叶子。 指定允许这些更改情况的正式机制超出了本规范的范围。

6. YANG 语法

YANG 语法类似于 SMIng [RFC3780]和 C 和 C++等编程语言。这种类 C 语法是专门为了可读性而选择的，因为 YANG 重视模块编写者和 YANG 工具链开发人员的模型读者的时间和精力。本节介绍 YANG 语法。

YANG 模块中的合法字符是 Unicode 和 ISO/IEC 10646 [ISO.10646]字符，包括制表符，回车符和换行符，但不包括其他 C0 控制字符，代理块和非字符。字符语法由第 14 节中的“yang-char”规则正式定义。

使用 UTF-8 [RFC3629]字符编码将 YANG 模块和子模块存储在文件中。

YANG 模块中的行以一个回车换行符或一个换行符结束。一个没有换行的回车可能只出现在一个带引号的字符串内（6.1.3 节）。请注意，带引号的字符串中出现的回车符和换行符不加修改地成为字符串值的一部分;多行引用字符串的值包含与 YANG 模块的行相同的行结束形式。

6.1. 词法标记

YANG 模块被解析为一系列的标记。 本节详细介绍了从输入流中识别令牌的规则。 YANG 标记化规则既简单又强大。 简单性是由于需要保持解析器易于实现的驱动，而功耗是由建模者需要以可读格式表示模型的事实驱动的。

6.1.1. 注释

评论是 C++风格。 单行注释以“//”开始，并在行尾处结束。 块注释以“/*”开始，以最接近的“*/”结束。

请注意，在带引号的字符串中（第 6.1.3 节），这些字符对永远不会被解释为注释的开始或结束。

6.1.2. 令牌

YANG 中的标记是关键字，字符串，分号（“;”）或大括号（“{”或“}”）。 一个字符串可以被引用或不引用。 关键字是本文档中定义的 YANG 关键字之一，或前缀标识符，后跟一个冒号（“:”），后跟一个语言扩展关键字。 关键字区分大小写。 有关标识符的正式定义，请参见第 6.2 节。

6.1.3. 引用

未加引号的字符串是不包含任何空格，制表符，回车符或换行符，单引号或双引号字符，分号（“;”），大括号（“{”或“}”）的任何字符序列，或注释序列（“//”，“/*”或“*/”）。
请注意，任何关键字都可以合法显示为未加引号的字符串。

在不带引号的字符串中，每个字符都被保留。请注意，这意味着反斜杠字符在没有引号的字符串中没有任何特殊含义。

如果双引号字符串包含换行符，后跟用于根据 YANG 文件中的布局缩进文本的空格或制表符，则会将该前导空白符从字符串中删除，直到包括起始列双引号字符或第一个非空白字符，以先发生者为准。必须检查剥离的后续行中的任何制表符首先被转换为 8 个空格字符。
如果双引号字符串在换行符之前包含空格或制表符，则将从字符串中除去该尾随的空格。

单引号字符串（用' '括起来）保留引号内的每个字符。单引号字符不能出现在单引号字符串中，即使前面加了反斜杠。
在双引号字符串（用" "括起来）中，反斜杠字符引入了一个特殊字符的表示，该字符取决于紧跟在反斜杠后面的字符：

- \n 换行符
- \t 制表符
- \" 一个双引号
- \\ 一个反斜杠

反斜杠不能跟随任何其他字符。

如果引用的字符串后跟加号（“+”），后跟另一个带引号的字符串，则这两个字符串将连接成一个字符串，从而允许多个连接来构建一个字符串。被引用的字符串和加号字符之间允许有空格，换行符和注释。
在双引号字符串中，空白修剪在替换反斜线转义字符之前完成。连接是作为最后一步执行的。

6.1.3.1. 引用例子

以下字符串是等效的：

```
hello
"hello"
'hello'
"hel" + "lo"
'hel' + "lo"
```

以下示例显示了一些特殊的字符串：

- `"\""` - 包含双引号的字符串
- `'"'` - 包含双引号的字符串
- `"\n"` - 包含换行符的字符串
- `'\n'` - 包含反斜杠后跟字符 `n` 的字符串

以下示例显示了一些非法字符串：

- `''''` - 单引号字符串不能包含单引号
- `"""` - 一个双引号必须用双引号字符串转义

以下字符串是等效的：

```
"first line
   second line"
"first line\n" + "  second line"
```

6.2. 身份标识

标识符用于标识不同种类的 `YANG` 项目的名称。 每个标识符以大写或小写 `ASCII` 字母或下划线字符开头，后跟零个或多个 `ASCII` 字母，数字，下划线字符，连字符和圆点。 实现必须支持最多 `64` 个字符的标识符，并且可以支持更长的标识符。 标识符区分大小写。 标识符语法由第 14 节中的规则“标识符(`identifier`)”正式定义。标识符可以被指定为带引号或不带引号的字符串。

6.2.1. 标识符及其名称空间

每个标识符在名称空间中都有效，该名称空间取决于所定义的 `YANG` 项目的类型。在命名空间中定义的所有标识符必须是唯一的。

- 所有模块和子模块名称共享相同的全局模块标识符名称空间。
- 在模块及其子模块中定义的所有扩展名称共享相同的扩展标识符名称空间。
- 在模块及其子模块中定义的所有功能名称共享相同的功能标识符名称空间。
- 在模块及其子模块中定义的所有身份名称共享相同的身份标识符名称空间。
- 在父节点或模块或其子模块顶层定义的所有派生类型名称共享相同的类型标识符名称空间。该名称空间的作用域是父节点或模块的所有后代节点。这意味着任何后代节点都可以使用该 `typedef`，并且不能定义一个同名的 `typedef`。
- 在父节点或模块或其子模块顶层定义的所有分组名称共享相同的分组标识符名称空间。 该名称空间的作用域是父节点或模块的所有后代节点。 这意味着任何后代节点都可以使用该分组，并且不能定义具有相同名称的分组。

- 在父节点或模块或其子模块的顶层定义（直接或通过“uses”语句）的所有叶子，叶子列表，容器，选择，rpcs，动作，通知，anydatas 和 anyxmls 共享相同的标识符名称空间。 该名称空间的作用域为父节点或模块，除非父节点是个案节点。 在这种情况下，命名空间的作用域是最靠近的不是大小写或选择节点的祖先节点。
- 选择中的所有案例共享相同的案例标识符名称空间。 该名称空间的作用域是父级选择节点。

YANG 允许转发引用。

6.3. 声明

一个 YANG 模块包含一系列的语句。 每个语句都以一个关键字开头，后跟零个或一个参数，后跟一个分号（“;”）或一个大括号内的子状态块（“{}”）：

```
statement = keyword [argument] (";" / "{" * statement "}")
```

参数是一个字符串，如 6.1.2 节定义。

6.3.1. 语言扩展

一个模块可以通过使用“extension”关键字来引入 YANG 扩展（参见第 7.19 节）。这些扩展可以通过其他模块用“import”语句导入（参见第 7.1.5 节）。当使用导入的扩展名时，扩展名的关键字必须使用扩展名模块导入的前缀进行限定。如果在定义的模块中使用了扩展名，那么扩展名的关键字必须使用该模块的前缀进行限定。

扩展的处理取决于对给定的 YANG 解析器或嵌入其的工具集是否声明对这些扩展的支持。在 YANG 模块中作为未知语句出现的不受支持的扩展（见第 14 节）可能会被忽略。任何受支持的扩展必须按照管理该扩展的规范进行处理。

定义扩展时必须小心，这样使用扩展的模块对于不支持扩展的应用程序也是有意义的。

6.4. XPath 评估

YANG 依靠 XML 路径语言（XPath）1.0 [XPATH]作为指定许多节点间引用和依赖关系的符号。实现不需要实现 XPath 解释器，但必须确保在数据模型中编码的需求得到执行。执行的方式是执行决定。 XPath 表达式必须在语法上是正确的，所有使用的前缀必须存在于 XPath 上下文中（见第 6.4.1 节）。实现可以选择手动实现它们，而不是直接使用 XPath 表达式。

XPath 表达式中使用的数据模型与 XPath 1.0 [XPATH]中使用的数据模型相同，具有与 XSLT 1.0 所使用的根节点子节点相同的扩展名（请参见[XSLT]中

的第 3.1 节）。具体而言，这意味着根节点可以具有任意数量的元素节点作为其子节点。
数据树没有文档顺序的概念。一个实现需要选择一些文档顺序，但是如何做是一个实现的决定。这意味着 YANG 模块中的 XPath 表达式不应该依赖于任何特定的文档顺序。
XPath 1.0 中的数字是 IEEE 754 [IEEE754-2008]双精度浮点值;参见[XPATH]中的第 3.5 节。这意味着 int64，uint64 和 decimal64 类型的某些值（请参见第 9.2 和 9.3 节）不能在 XPath 表达式中精确表示。因此，在 XPath 表达式中使用具有 64 位数值的节点时，应该谨慎行事。特别是涉及到对比的数字比较可能会产生意想不到的结果。
例如，请考虑以下定义：

```
leaf lxiv {
  type decimal64 {
    fraction-digits 18;
  }
  must ". <= 10";
}
```

具有值 10.0000000000000001 的“lxiv”叶的实例将成功通过验证。

6.4.1. XPath 上下文

所有 YANG XPath 表达式共享以下 XPath 上下文定义：

- 命名空间声明的集合是指定了 XPath 表达式的模块中的所有“import”语句的前缀和名称空间对的集合，以及“namespace”语句的 URI 的“prefix”语句的前缀。
- 没有名称空间前缀的名称与当前节点的标识符属于同一个名称空间。在一个分组中，这个名字空间受到分组的使用位置的影响（见第 7.13 节）。在 typedef 中，该名称空间受到引用 typedef 的位置的影响。如果在分组中定义并引用 typedef，则命名空间将受到分组使用位置的影响（请参见第 7.13 节）。
- 函数库是[XPATH]中定义的核心函数库和第 10 节定义的函数。
- 变量绑定的集合是空的。

XPath 2.0 [XPATH2.0]采用了处理前缀名的机制，有助于简化 YANG 中的 XPath 表达式。由于 YANG 节点标识符总是具有非空名称空间 URI 的限定名称，所以不会出现任何歧义。

- 可访问树取决于具有 XPath 表达式的语句的定义位置：
- 如果 XPath 表达式在子状态中定义为表示配置的数据节点，则可访问树是上下文节点所在数据存储区中的数据。根节点将所有模块中的所有顶级配置数据节点作为子节点。

- 如果在子状态中将 XPath 表达式定义为表示状态数据的数据节点，则可访问树是服务器中的所有状态数据，以及正在运行的配置数据存储区。根节点将所有模块中的所有顶级数据节点都作为子节点。
- 如果 XPath 表达式在子语句中定义为“notification”语句，则可访问树是通知实例，服务器中的所有状态数据以及正在运行的配置数据存储区。如果在模块顶层定义通知，则根节点将表示通知的节点定义为所有模块中的顶级数据节点，并将其作为子节点。否则，根节点将所有模块中的所有顶级数据节点都视为子节点。
- 如果 XPath 表达式在子语句中定义为“rpc”或“action”语句中的“input”语句，则可访问的树是 RPC 或操作操作(action)实例，服务器中的所有状态数据以及正在运行的配置数据存储区。根节点在所有模块中都有作为子节点的顶级数据节点。此外，对于 RPC，根节点也具有代表 RPC 操作的节点被定义为子节点。表示正在定义的操作的节点具有操作的输入参数作为子节点。
- 如果 XPath 表达式在子语句中定义为“rpc”或“action”语句中的“output”语句，则可访问的树是 RPC 或操作(action)实例，服务器中的所有状态数据以及正在运行的配置数据存储区。根节点在所有模块中都有作为子节点的顶级数据节点。此外，对于 RPC，根节点也具有代表 RPC 操作的节点被定义为子节点。表示正在定义的操作的节点具有操作的输出参数作为子节点。
  在可访问的树中，存在所有使用默认值的叶子和叶子列表（参见 7.6.1 和 7.7.2 节）。

  如果存在于可访问树中的节点具有不存在容器作为子节点，则不存在容器也存在于可访问树中。

  上下文节点随着 YANG XPath 表达式而变化，并且在定义了具有 XPath 表达式的 YANG 语句的地方被指定。

6.4.1.1. 例子

鉴于以下模块：

```
module example-a {
  yang-version 1.1;
  namespace urn:example:a;
  prefix a;

  container a {
    list b {
      key id;
      leaf id {
        type string;
      }
      notification down {
        leaf reason {
          type string;
```

```
      }
    }
    action reset {
      input {
        leaf delay {
          type uint32;
        }
      }
      output {
        leaf result {
          type string;
        }
      }
    }
  }
  notification failure {
    leaf b-ref {
      type leafref {
        path "/a/b/id";
      }
    }
  }
}
```

并给出了以 XML 格式指定的以下数据树：

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
```

通知“down” 在 /a/b[id ="2"]的可访问树是：

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
    <down>
      <reason>error</reason>
    </down>
  </b>
```

```
</a>
//可能还有其他顶级节点
```

在“when”参数设置为“10”时，对/a/b[id="1"]上的“reset”动作调用的可访问树将是：

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
    <reset>
      <delay>10</delay>
    </reset>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
//可能还有其他顶级节点
```

这个动作输出的可访问树是：

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
    <reset>
      <result>ok</result>
    </reset>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
//可能还有其他顶级节点
```

通知“failure”的可访问树可以是：

```
<a xmlns="urn:example:a">
  <b>
    <id>1</id>
  </b>
  <b>
    <id>2</id>
  </b>
</a>
<failure>
  <b-ref>2</b-ref>
</failure>
//可能还有其他顶级节点
```

6.5. 模式节点标识符

模式节点标识符是标识模式树中节点的字符串。 它有两种形式，分别是第 14 节中的“absolute-schema-nodeid”和“descendant-schema-nodeid”规则定义的“absolute”和“descendant”。 模式节点标识符由标识符的路径组成，用斜线（“/”）分隔。 在绝对模式节点标识符中，前导斜杠后面的第一个标识符是本地模块或导入模块中的任何顶级模式节点。
对外部模块中定义的标识符的引用必须使用适当的前缀进行限定，对当前模块及其子模块中定义的标识符的引用可以使用前缀。

例如，为了识别顶级节点“a”的子节点“b”，可以使用字符串“/a/b”。

7. YANG 声明

以下部分描述了所有的 YANG 语句。
请注意，即使是没有在 YANG 中定义的任何子语句的语句，也可以使用特定于供应商的扩展名作为子语句。 例如，“description”语句没有在 YANG 中定义的任何子语句，但是下面是合法的：

```
description "Some text." {
  ex:documentation-flag 5;
}
```

7.1 "模块"声明

“module”语句定义了模块的名称，并将属于该模块的所有语句分组在一起。
“module”语句的参数是模块的名称，后面跟着一个包含详细模块信息的子语句块。 模块名称是一个标识符（见第 6.2 节）。
在 RFC 流中发布的模块的名称[RFC4844]必须由 IANA 分配; 参见[RFC6020]中的第 14 节。
私有模块名称由拥有该模块的组织分配，没有中央注册表。 有关如何命名模块的建议，请参见第 5.1 节。

模块通常具有以下布局：

```
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>
}
```

7.1.1. 模块的子语句

子语句	章节	基数
anydata	7.10	0..n
anyxml	7.11	0..n
augment	7.17	0..n
choice	7.9	0..n
contact	7.1.8	0..1
container	7.5	0..n
description	7.21.3	0..1
deviation	7.20.3	0..n
extension	7.19	0..n
feature	7.20.1	0..n
grouping	7.12	0..n
identity	7.18	0..n
import	7.1.5	0..n
include	7.1.6	0..n

子语句	章节	基数
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
namespace	7.1.3	1
notification	7.16	0..n
organization	7.1.7	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision	7.1.9	0..n
rpc	7.14	0..n
typedef	7.3	0..n
uses	7.13	0..n
yang-version	7.1.2	1

RFC 原表

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| anydata      | 7.10    | 0..n        |
| anyxml       | 7.11    | 0..n        |
| augment      | 7.17    | 0..n        |
| choice       | 7.9     | 0..n        |
| contact      | 7.1.8   | 0..1        |
| container    | 7.5     | 0..n        |
| description  | 7.21.3  | 0..1        |
| deviation    | 7.20.3  | 0..n        |
| extension    | 7.19    | 0..n        |
| feature      | 7.20.1  | 0..n        |
| grouping     | 7.12    | 0..n        |
| identity     | 7.18    | 0..n        |
| import       | 7.1.5   | 0..n        |
| include      | 7.1.6   | 0..n        |
| leaf         | 7.6     | 0..n        |
| leaf-list    | 7.7     | 0..n        |
```

```
| list         | 7.8     | 0..n        |
| namespace    | 7.1.3   | 1           |
| notification | 7.16    | 0..n        |
| organization | 7.1.7   | 0..1        |
| prefix       | 7.1.4   | 1           |
| reference    | 7.21.4  | 0..1        |
| revision     | 7.1.9   | 0..n        |
| rpc          | 7.14    | 0..n        |
| typedef      | 7.3     | 0..n        |
| uses         | 7.13    | 0..n        |
| yang-version | 7.1.2   | 1           |
+--------------+---------+-------------+
```

7.1.2. “yang-version”声明

“yang-version”语句指定在开发模块时使用了哪种版本的 YANG 语言。 声明的参数是一个字符串。 它必须包含根据此规范定义的 YANG 模块的值“1.1”。
不包含“yang-version”语句的模块或子模块，或者包含值“1”的模块或子模块是针对[RFC6020]中定义的 YANG 版本 1 开发的。
处理“1.1”（此处定义的版本）以外版本的“yang-version”语句超出了本规范的范围。 任何定义更高版本的文档都需要定义这种更高版本的后向兼容性。
有关 YANG 版本 1 和 1.1 之间的兼容性，请参见第 12 节。

7.1.3. “namespace”声明

“namespace”语句定义了 XML 名称空间，模块定义的所有标识符都以 XML 编码进行了限定，除了分组内定义的数据节点(data nodes)，动作节点(action nodes)和通知节点(notification nodes)的标识符（详见 7.13 节）。 “namespace”语句的参数是命名空间的 URI。
另见第 5.3 节。

7.1.4. “prefix”声明

“prefix”语句用于定义与模块及其名称空间关联的前缀。 “prefix”语句的参数是用作访问模块前缀的前缀字符串。前缀字符串可以与模块一起用于引用模块中包含的定义，例如“if:ifName”。前缀是一个标识符（见第 6.2 节）。
在“module”语句中使用时，“prefix”语句定义了在导入模块时建议使用的前缀。
为了提高 NETCONF XML 的可读性，生成使用前缀的 XML 或 XPath 的 NETCONF 客户机或服务器应该使用模块定义的前缀作为 XML 命名空间前缀，除非存在冲突。
在“import”语句中使用时，“prefix”语句定义在导入模块中访问定义时要使用的前缀。当使用从导入的模块引用标识符时，使用导入模块的前缀字符串，后跟冒号（“:”）和标识符，例如“if:ifIndex”。为了提高 YANG 模块的可读性，在模块导入时，应该使用模块定义的前缀，除非有冲突。如果存在冲突，即两个已定义相同前缀的不同模块被导入，则至少其中一个模块必须以不同的前缀导入。

所有前缀，包括模块本身的前缀，必须在模块或子模块中是唯一的。

7.1.5. “import”声明

“import”语句使得一个模块的定义在另一个模块或子模块中可用。参数是要导入的模块的名称，语句之后是一个包含详细导入信息的子状态块。导入模块时，导入模块可以：

- 使用导入模块或其子模块顶层定义的任何分组和 typedef。
- 使用导入模块或其子模块中定义的任何扩展名，特性和标识。

- 在“must”，“path”和“when”语句中使用导入模块的模式树中的任何节点，或者在“augment”和“deviation”语句中使用目标节点。

强制的“prefix”子语句为导入的模块分配一个前缀，该前缀的作用域是导入模块或子模块。可以指定多个“import”语句从不同的模块导入。

当存在可选的“revision-date”子语句时，本地模块中由定义引用的任何 typedef，分组(grouping)，扩展(extension)，特征(feature)和标识(identity)都将取自导入模块的指定修订版本。如果导入的模块的指定修订版不存在，则为错误。如果不存在“修订日期(revision-data)”子语句，则取决于所取模块的哪个版本。

只要使用不同的前缀，可以导入同一模块的多个修订版本。

import 的子语句

substatement	section	cardinality
description	7.21.3	0..1
prefix	7.1.4	1
reference	7.21.4	0..1
revision-date	7.1.5.1	0..1

7.1.5.1. import 的“revision-date”声明

导入的“修订日期(revision-date)”语句用于指定要导入的模块的版本。

7.1.6. “include”声明

“include”语句用于使子模块的内容可用于该子模块的父模块。参数是一个标识符，是要包含的子模块的名称。模块只允许包含属于该模块的子模块，如“belongs-to”语句所定义的（参见第 7.2.2 节）。

当模块包含子模块时，它将子模块的内容合并到模块的节点层次结构中。

为了向后兼容 YANG 版本 1，子模块允许包含属于同一个模块的另一个子模块，但在版本 1.1（见 5.1 节）中这不是必需的。

当存在可选的“revision-date”子状态时，子模块的指定修订包含在模块中。如果子模块的指定修订版不存在，则是错误的。如果没有“revision-date”子语句，则不定义包含子模块的哪个版本。

不得包含同一子模块的多个版本。

include 的子语句

substatement	section	cardinality
description	7.21.3	0..1
reference	7.21.4	0..1
revision-date	7.1.5.1	0..1

7.1.7. “organization”声明

“组织”(organization)声明定义了对此模块负责的一方。 参数是一个字符串，用于指定由本模块开发的组织的文本描述。

7.1.8. “contact”声明

“联系”(contact)声明提供模块的联系信息。 参数是一个字符串，用于指定应该发送有关该模块的技术查询的人员的联系信息，如姓名，邮政地址，电话号码和电子邮件地址。

7.1.9. “revision”声明

“修订”(revision)语句指定模块的编辑修订历史，包括初始修订。 一系列“revision”语句详细说明了模块定义中的变化。 参数是格式为“YYYY-MM-DD”的日期字符串，后跟一个包含详细修订信息的子状态块。 一个模块应该至少有一个“revision”语句。 对于每个已发表的编辑变更，应在修订顺序前添加一个新的修订顺序，以便所有修订顺序都是相反的。

7.1.9.1. revision 的子语句

substatement	section	cardinality
description	7.21.3	0..1
reference	7.21.4	0..1

7.1.10. 使用示例

以下示例依赖于[RFC6991]。

```
module example-system {
  yang-version 1.1;
  namespace "urn:example:system";
  prefix "sys";

  import ietf-yang-types {
    prefix "yang";
```

```
    reference "RFC 6991: Common YANG Data Types";
  }

  include example-types;

  organization "Example Inc.";
  contact
    "Joe L. User

     Example Inc.
     42 Anywhere Drive
     Nowhere, CA 95134
     USA

     Phone: +1 800 555 0100
     Email: joe@example.com";

  description
    "The module for entities implementing the Example system.";

  revision 2007-06-09 {
    description "Initial revision.";
  }

  // definitions follow...
}
```

7.2. “子模块”声明

YANG 中的主要单元是一个模块，而 YANG 模块本身可以由几个子模块构成。子模块允许模块设计人员将复杂模型拆分为多个子模块，其中所有子模块对包含子模块的模块定义的单个名称空间起作用。

“submodule”语句定义子模块的名称，并将属于子模块的所有语句组合在一起。 “submodule”语句的参数是子模块的名称，后面跟着一个包含详细子模块信息的子语句块。子模块名称是一个标识符（见第 6.2 节）。

RFC 流发布的子模块的名称必须由 IANA 分配;参见[RFC6020]中的第 14 节。私有子模块名称由拥有子模块的组织分配，没有中央注册表。有关如何命名子模块的建议，请参见第 5.1 节。

子模块通常具有以下布局：

```
submodule <module-name> {
```

```
  <yang-version statement>

  // module identification
  <belongs-to statement>

  // linkage statements
  <import statements>

  // meta-information
  <organization statement>
  <contact statement>
  <description statement>
  <reference statement>

  // revision history
  <revision statements>

  // module definitions
  <other statements>
}
```

7.2.1. 子模块的子语句

submodule 的子语句基本上和 `module` 的子语句(参见 7.1.1 节)一致，只是多了一个 `belongs-to` 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| anydata      | 7.10    | 0..n        |
| anyxml       | 7.11    | 0..n        |
| augment      | 7.17    | 0..n        |
| belongs-to   | 7.2.2   | 1           |
| choice       | 7.9     | 0..n        |
| contact      | 7.1.8   | 0..1        |
| container    | 7.5     | 0..n        |
| description  | 7.21.3  | 0..1        |
| deviation    | 7.20.3  | 0..n        |
| extension    | 7.19    | 0..n        |
| feature      | 7.20.1  | 0..n        |
| grouping     | 7.12    | 0..n        |
| identity     | 7.18    | 0..n        |
| import       | 7.1.5   | 0..n        |
| include      | 7.1.6   | 0..n        |
| leaf         | 7.6     | 0..n        |
```

```
| leaf-list    | 7.7     | 0..n        |
| list         | 7.8     | 0..n        |
| notification | 7.16    | 0..n        |
| organization | 7.1.7   | 0..1        |
| reference    | 7.21.4  | 0..1        |
| revision     | 7.1.9   | 0..n        |
| rpc          | 7.14    | 0..n        |
| typedef      | 7.3     | 0..n        |
| uses         | 7.13    | 0..n        |
| yang-version | 7.1.2   | 1           |
+--------------+---------+-------------+
```

7.2.2。 “belongs-to”声明

“belongs-to”语句指定子模块所属的模块。 参数是一个标识符，即模块的名称。
一个子模块只能被它所属的模块或属于该模块的另一个子模块所包含。

强制的“prefix”子状态为子模块所属的模块分配一个前缀。 子模块所属模块中的所有定义及其所有子模块都可以使用前缀进行访问。
"belongs-to"的子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| prefix       | 7.1.4   | 1           |
+--------------+---------+-------------+
```

7.2.3 使用示例

```
submodule example-types {
  yang-version 1.1;
  belongs-to "example-system" {
    prefix "sys";
  }

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

  organization "Example Inc.";
  contact
    "Joe L. User
```

```
    Example Inc.
    42 Anywhere Drive
    Nowhere, CA 95134
    USA

    Phone: +1 800 555 0100
    Email: joe@example.com";

  description
    "This submodule defines common Example types.";

  revision "2007-06-09" {
    description "Initial revision.";
  }

  // definitions follow...
}
```

7.3. “typedef”声明

“typedef”语句根据 5.5 节的规则定义了一个新的类型，可以在模块或子模块中本地使用，也可以从其他模块中使用。 新类型称为“派生类型”(derived type)，它的派生类型被称为“基类型”(base type)。 所有派生类型都可以追溯到 YANG 内置类型。

“typedef”语句的参数是一个标识符，它是要定义的类型的名称，必须后跟一个包含详细 typedef 信息的子语句块。

类型的名称不能是 YANG 内置类型之一。 如果 typedef 是在 YANG 模块或子模块的顶层定义的，那么要定义的类型的名称在模块中必须是唯一的。

7.3.1. typedef 的字语句

substatement	section	cardinality
default	7.3.4	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
type	7.3.2	1
units	7.3.3	0..1

7.3.2. typedef 的“type”语句

必须存在的“type”语句定义了派生此类型的基本类型。 细节见第 7.4 节。

7.3.3. “units”声明

“units”语句是可选的，它将一个字符串作为参数，该字符串包含与该类型关联的单位的文本定义。

7.3.4. typedef 的“default”语句

“default”语句将包含新类型默认值的字符串作为参数。
“default”语句的值必须根据“type”语句中指定的类型有效。
如果基类型具有默认值，而新派生类型不指定新的默认值，则基类型的默认值也是新派生类型的默认值。

如果根据派生类型或叶定义中指定的新限制，类型的缺省值无效，则派生类型或叶定义必须指定一个与限制兼容的新缺省值。

7.3.5. 使用示例

```
typedef listen-ipv4-address {
  type inet:ipv4-address;
  default "0.0.0.0";
}
```

7.4. “类型”声明

“type”语句将一个字符串作为参数，这个字符串是一个 YANG 内置类型（参见第 9 节）或派生类型（参见 7.3 节）的名称，后面跟着一个可选的子语句块，用于放置类型的限制。
可以应用的限制取决于受限制的类型。 所有内置类型的限制声明在第 9 节的小节中都有描述。

7.4.1. type 的子语句

substatement	section	cardinality
base	7.18.2	0..n
bit	9.7.4	0..n
enum	9.6.4	0..n
fraction-digits	9.3.4	0..1
length	9.4.4	0..1
path	9.9.2	0..1
pattern	9.4.5	0..n
range	9.2.4	0..1
require-instance	9.9.3	0..1
type	7.4	0..n

```
+------------------+---------+-------------+
```

7.5. “容器”声明

“container”语句用于定义架构树中的内部数据节点。 它有一个参数，它是一个标识符，后面跟着一个包含详细容器信息的子状态块。
容器节点没有值，但在数据树中有一个子节点的列表。 子节点在容器的子状态中定义。

7.5.1. 容器的存在

YANG 支持两种类型的容器，这些容器仅用于组织数据节点的层次结构，以及那些在数据树中的存在具有明确含义的容器。
在第一种风格中，容器没有其自身的意义，只存在于包含子节点。 特别是，没有子节点的容器节点的存在在语义上等同于不存在容器节点。 YANG 称这种风格为“不存在容器”(non-presence container)。 这是默认的类型。
例如，用于同步光网络（SONET）接口的一组加扰选项可以放置在“scrambling”容器内以增强配置分层结构的组织并将这些节点保持在一起。 “scrambling”节点本身没有任何意义，所以当节点变空时删除节点，可以减轻用户的工作量。
在第二种方式中，容器本身的存在具有一些含义，代表一个数据位。

对于配置数据，容器既充当配置按钮又充当组织相关配置节点的手段。 这些容器是显式创建和删除的。

YANG 将这种风格称为“存在容器”(presence container)，并使用“存在”语句来指示它，该存储过程以一个文本字符串作为参数，表示节点的存在。
例如，“ssh”容器可以使用 Secure Shell（SSH）打开登录服务器的能力，但也可以包含任何与 SSH 有关的配置按钮，例如连接速率或重试限制。
“presence”语句（见第 7.5.5 节）用于给出数据树中容器的存在的语义。

7.5.2. container 的子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| action       | 7.15    | 0..n        |
| anydata      | 7.10    | 0..n        |
| anyxml       | 7.11    | 0..n        |
| choice       | 7.9     | 0..n        |
| config       | 7.21.1  | 0..1        |
| container    | 7.5     | 0..n        |
| description  | 7.21.3  | 0..1        |
```

```
| grouping     | 7.12    | 0..n        |
| if-feature   | 7.20.2  | 0..n        |
| leaf         | 7.6     | 0..n        |
| leaf-list    | 7.7     | 0..n        |
| list         | 7.8     | 0..n        |
| must         | 7.5.3   | 0..n        |
| notification | 7.16    | 0..n        |
| presence     | 7.5.5   | 0..1        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| typedef      | 7.3     | 0..n        |
| uses         | 7.13    | 0..n        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.5.3. “必须”(must)声明

“must”语句是可选的，它将一个包含 XPath 表达式的字符串作为参数（见 6.4 节）。它用于正式声明对有效数据的约束。约束条件按照第 8 节的规定执行。

当一个数据存储被验证时，所有的“must”约束在可访问树中的每个节点的概念上被评估一次（见第 6.4.1 节）。

所有这些约束必须评估为“true”，以使数据有效。

除了第 6.4.1 节中的定义之外，XPath 表达式在以下上下文中概念性地评估：

- 如果“must”语句是“notification”语句的子语句，则上下文节点是表示可访问树中通知的节点。
- 如果“must”语句是“input”语句的子语句，则上下文节点是表示可访问树中操作的节点。
- 如果“must”语句是“output”语句的子语句，则上下文节点是表示可访问树中操作的节点。

否则，上下文节点是可访问树中为其定义“must”语句的节点。

使用标准的 XPath 规则将 XPath 表达式的结果转换为布尔值。

请注意，由于数据树中的所有叶子值在概念上都是以规范形式存储的（请参见第 9.1 节），所以任何 XPath 比较都是在规范值上进行的。

另请注意，XPath 表达式是在概念上评估的。这意味着实现不必在服务器中使用 XPath 评估程序。评估如何在实践中完成是一个实施决策。

7.5.4.1. “错误消息”(error-message)声明

“error-message”语句是可选的，它将字符串作为参数。 如果约束的计算结果为“false”，则字符串在 NETCONF 的<rpc-error>中作为<error-message>传递。

7.5.4.2. “错误应用标记”(error-app-tag)声明

“error-app-tag”语句是可选的，它将字符串作为参数。 如果约束条件的计算结果为“false”，则字符串在 NETCONF 的<rpc-error>中作为<error-app-tag>传递。

7.5.4.3. must 和 error-message 的使用示例

```
container interface {
  leaf ifType {
    type enumeration {
      enum ethernet;
      enum atm;
    }
  }
  leaf ifMTU {
    type uint32;
  }
  must 'ifType != "ethernet" or ifMTU = 1500' {
    error-message "An Ethernet MTU must be 1500";
  }
  must 'ifType != "atm" or'
     + ' (ifMTU <= 17966 and ifMTU >= 64)' {
    error-message "An ATM MTU must be 64 .. 17966";
  }
}
```

7.5.5. “存在”(presence)声明

“presence”语句为数据树中容器的存在赋予一个含义。 它以一个字符串作为参数，该字符串包含文本描述节点存在的含义。
如果容器具有“presence”语句，则数据树中的容器存在意义。 否则，容器被用来给数据一些结构，它本身没有意义。
有关更多信息，请参阅第 7.5.1 节。

7.5.6. 容器的子节点语句

在一个容器中，可以使用“container”，“leaf”，“list”，“leaf-list”，“uses”，“choice”，“anydata”和“anyxml”语句来定义容器的子节点。

7.5.7. XML 编码规则

一个容器节点被编码为一个 XML 元素。 元素的本地名称是容器的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。
容器的子节点被编码为容器元素的子元素。 如果容器定义了 RPC 或操作输入或输出参数，则这些子元素将按照“container”语句中定义的顺序进行编码。否则，子元素按任何顺序编码。
容器的子元素之间的任何空格都是不重要的，即一个实现可以在子元素之间插入空白字符。

如果一个不存在的容器没有任何子节点，容器可能会或可能不会出现在 XML 编码中。

7.5.8.NETCONF <edit-config>操作

通过使用容器的 XML 元素中的“operation”属性（参见[RFC6241]中的第 7.2 节），可以通过<edit-config>创建，删除，替换和修改容器。
如果一个容器没有“presence”语句并且最后一个子节点被删除，NETCONF 服务器可能会删除该容器。
当 NETCONF 服务器处理<edit-config>请求时，容器节点的过程元素如下所示：

- 如果操作是“合并(merge)”或“替换(replace)”，则创建该节点（如果该节点不存在）。
- 如果操作是“创建(create)”，则创建该节点（如果该节点不存在）。 如果节点已经存在，则返回“数据存在(data-exists)”错误。
- 如果操作是“删除(delete)”，则该节点将被删除（如果存在）。 如果节点不存在，则返回“数据缺失(data-missing)”错误。

7.5.9. 使用示例

鉴于以下容器定义：

```
container system {
  description
    "Contains various system parameters.";
  container services {
    description
      "Configure externally available services.";
    container "ssh" {
      presence "Enables SSH";
      description
        "SSH service-specific configuration.";
      // more leafs, containers, and stuff here...
    }
  }
}
```

相应的 XML 实例示例：

```
<system>
  <services>
    <ssh/>
  </services>
</system>
```

由于存在<ssh>元素，所以启用了 SSH。
使用<edit-config>删除容器：

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

```
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh nc:operation="delete"/>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.6. “叶”声明

“leaf”语句用于在模式树中定义叶节点。 它需要一个参数，它是一个标识符，后面是一个包含详细叶子信息的子状态块。
叶节点在数据树中有一个值，但没有子节点。 从概念上讲，数据树中的值始终是规范形式（参见第 9.1 节）。

数据树中存在零个或一个实例的叶节点。

“leaf”语句用于定义特定内置或派生类型的标量变量。

7.6.1. 叶子的默认值

如果叶子不存在于数据树中，叶子的默认值就是服务器使用的值。缺省值的使用取决于架构树中不是不存在容器的树叶最近的祖先节点（请参阅第 7.5.1 节）：

- 如果模式树中不存在这样的祖先，则必须使用默认值。
- 否则，如果这个祖先是个案例节点，如果数据树中存在任何案例的节点，或者案例节点是选择的默认案例，并且数据中不存在任何其他案例的节点，则必须使用默认值树。
- 否则，如果祖先节点存在于数据树中，则必须使用默认值。

在这些情况下，默认值被认为正在使用中。

请注意，如果叶子或其任何祖先具有“when”条件或“if-feature”表达式评估为“false”，则默认值未被使用。

当默认值正在使用时，服务器必须在操作上表现得好像数据树中存在叶子，其默认值是它的值。

如果一个叶子有一个“default”语句，叶子的默认值就是“default”语句的值。否则，如果叶子的类型有一个默认值，叶子不是强制性的，叶子的默认值就是该类型的默认值。在其他所有情况下，叶子没有默认值。

7.6.2. leaf 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| config       | 7.21.1  | 0..1        |
| default      | 7.6.4   | 0..1        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
| mandatory    | 7.6.5   | 0..1        |
| must         | 7.5.3   | 0..n        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| type         | 7.6.3   | 1           |
| units        | 7.3.3   | 0..1        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.6.3. 叶的“类型”声明

必须存在的“type”语句将现有内置类型或派生类型的名称作为参数。 可选的子变量指定了这种类型的限制。 细节见第 7.4 节。

7.6.4. 叶子的“默认”声明

“default”语句是可选的，它将一个包含叶子默认值的字符串作为参数。
“default”语句的值必须根据叶的“type”语句中指定的类型有效。
“default”语句不能出现在“mandatory”为“true”的节点上。
默认值的定义不能用“if-feature”语句来标记。 例如，以下是非法的：

```
leaf color {
  type enumeration {
    enum blue { if-feature blue; }
    ...
  }
  default blue; // 非法 - 枚举值是有条件的
}
```

7.6.5. 叶的“强制性”声明

“mandatory”语句是可选的，它将字符串“true”或“false”作为参数，并对有效数据进行约束。 如果未指定，则默认为“false”。
如果“mandatory”为“true”，那么约束的行为取决于架构树中不是不存在容器的叶节点最接近的祖先节点的类型（见第 7.5.1 节）：

- 如果模式树中不存在这样的祖先，叶必须存在。
- 否则，如果这个祖先是一个案例节点，如果数据树中存在任何来自案例的节点，叶子必须存在。

  否则，如果祖先节点存在于数据树中，叶子必须存在。

  这个约束是根据第 8 节中的规则执行的。

7.6.6. XML 编码规则

叶节点被编码为 XML 元素。 元素的本地名称是叶子的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。
叶节点的值根据类型编码为 XML，并作为元素中的字符数据发送。
一个例子见 7.6.8 节。

7.6.7. NETCONF <edit-config>操作

当 NETCONF 服务器处理<edit-config>请求时，叶节点的过程元素如下：

- 如果操作是“合并(merge)”或“替换(replace)”，则创建该节点（如果该节点不存在），并将其值设置为在 XML RPC 数据中找到的值。
- 如果操作是“创建(create)”，则创建该节点（如果该节点不存在）。 如果节点已经存在，则返回“数据存在(data-exists)”错误。
- 如果操作是“删除(delete)”，则该节点将被删除（如果存在）。 如果节点不存在，则返回“数据缺失(data-missing)”错误。

7.6.8. 使用示例

鉴于下面的“leaf”声明，放在先前定义的“ssh”容器中（见 7.5.9 节）：

```
leaf port {
  type inet:port-number;
  default 22;
  description
    "The port to which the SSH server listens.";
}
```

相应的 XML 实例示例：

```
<port>2022</port>
```

使用<edit-config>设置叶子的值：

```
<rpc message-id="101"
```

```
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <port>2022</port>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.7. “叶列表”声明

在使用“leaf”语句定义特定类型的简单标量变量的情况下，“leaf-list”语句用于定义特定类型的数组。 “leaf-list”语句带有一个参数，这个参数是一个标识符，后面是一个包含叶列表信息的子语句块。
在配置数据中，叶列表中的值必须是唯一的。

默认值的定义不能用“if-feature”语句来标记。
从概念上讲，数据树中的值必须是规范形式（见 9.1 节）。

7.7.1. 排序

YANG 支持两种样式来排序列表和叶列表中的条目。在许多列表中，列表条目的顺序不会影响列表配置的实现，服务器可以按任何合理的顺序自由排列列表条目。列表的“description”字符串可能会向服务器实现者建议一个排序。YANG 将这种风格称为“系统排序(system ordered)”这样的列表用“ordered-by system”的语句表示。
例如，有效用户列表通常会按字母顺序排序，因为用户在配置中出现的顺序不会影响用户帐户的创建。

在其他样式的列表中，列表条目的顺序对于列表配置的实现是重要的，并且用户负责对条目进行排序，而服务器维持该顺序。YANG 称这种风格的“用户排序(user ordered)”的名单;这样的列表用“按用户排序(ordered-by user)”的语句表示。

例如，数据包筛选器条目应用于传入流量的顺序可能会影响该流量被过滤的方式。用户需要决定是否应该在过滤条目之前或之后应用丢弃所有 TCP 流量的过滤条目，以允许来自可信接口的所有流量。排序的选择是至关重要的。YANG 在 NETCONF 的<edit-config>操作中提供了丰富的功能，可以控制用户排序列表中列表条目的顺序。列表条目可以被插入或重新排列，被定位为列表中的第一个或最后一个条目，或被定位在另一个特定条目之前或之后。
7.7.7 节介绍了“ordered-by”语句。

7.7.2. 叶列表的默认值

如果数据树中不存在叶列表，则叶列表的缺省值是服务器使用的值。缺省值的使用取决于架构树中不是无状态容器的叶节点最近的祖先节点（参见 7.5.1 节）：

- 如果模式树中不存在这样的祖先，则必须使用默认值。
- 否则，如果这个祖先是一个 case 节点，如果数据树中有任何节点存在于数据树中，或者 case 节点是选择的缺省情况，并且数据中没有任何其他情况的节点，则必须使用默认值树。
- 否则，如果祖先节点存在于数据树中，则必须使用默认值。

在这些情况下，默认值被认为正在使用中。

请注意，如果叶列表或其任何祖先具有“when”条件或“if-feature”表达式的计算结果为“false”，则默认值不会被使用。
当使用默认值时，服务器必须在操作上表现得像数据树中存在的叶表一样，其默认值是其值。

如果叶列表具有一个或多个“default”语句，叶列表的默认值是“default”语句的值，并且如果叶列表是用户排序的，则默认值按照“default”语句。否则，如果叶列表的类型有一个默认值，并且叶列表没有一个值大于或等于 1 的“min-elements”语句，那么叶列表的默认值是该类型的一个实例默认值。在所有其他情况下，叶列表没有任何默认值。

7.7.3. leaf-list 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| config       | 7.21.1  | 0..1        |
| default      | 7.7.4   | 0..n        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
| max-elements | 7.7.6   | 0..1        |
| min-elements | 7.7.5   | 0..1        |
| must         | 7.5.3   | 0..n        |
```

```
| ordered-by   | 7.7.7   | 0..1        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| type         | 7.4     | 1           |
| units        | 7.3.3   | 0..1        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.7.4. 叶列表的“default”声明

“default”语句是可选的，它将一个字符串作为参数，该字符串包含叶列表的默认值。
“default”语句的值必须根据叶列表的“type”语句中指定的类型有效。
在“min-elements”的值大于或等于 1 的节点上，“default”语句不能出现。

7.7.5. “最小元素”声明

“min-elements”语句是可选的，它将一个非负整数作为参数，对有效列表条目进行约束。 一个有效的叶列表或者列表必须至少有 min 元素条目。
如果不存在“min-elements”语句，则默认为零。
约束的行为取决于模式树中叶列表或列表最近的祖先节点的类型，它不是一个不存在的容器（见 7.5.1 节）：

- 如果模式树中不存在这样的祖先，则强制约束。
- 否则，如果这个祖先是一个 case 节点，那么如果这个 case 的任何其他节点存在，这个约束就会被执行。
- 否则，如果祖先节点存在，则强制执行。

根据第 8 节的规定，约束条件得到了进一步的执行。

7.7.6. “最大元素”声明

“max-elements”语句是可选的，它将一个正整数或字符串“unbounded”作为参数，对有效的列表条目进行约束。一个有效的叶列表或列表总是至多有最大元素条目。
如果不存在“max-elements”语句，则默认为“unbounded”。
“最大元素”约束按照第 8 节的规定执行。

7.7.7. “排序”声明

“ordered-by”语句定义列表中的条目顺序是由用户还是系统确定的。 参数是字符串“system”或“user”之一。 如果不存在，则默认为“system”。
如果列表代表状态数据，RPC 输出参数或通知内容，则该语句将被忽略。

更多信息见 7.7.1 节。

7.7.7.1. 基于系统的排序

列表中的条目按照系统确定的顺序排序。 列表的“description”字符串可能会向服务器实现者建议一个命令。如果没有，则可以按照任何顺序自由地对条目进行排序。一个实现应该使用相同的顺序来处理相同的数据，而不管数据是如何创建的。使用确定性顺序将使用简单工具（比如“diff”）进行比较。
这是默认的顺序。

7.7.7.2. 基于用户的排序

列表中的条目按照用户定义的顺序排序。在 NETCONF 中，这个顺序是通过在<edit-config>请求中使用特殊的 XML 属性来控制的。细节见 7.7.9 节。

7.7.8. XML 编码规则

叶列表节点被编码为一系列 XML 元素。 每个元素的本地名称是叶列表的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。
每个叶列表项的值根据类型被编码为 XML，并作为元素中的字符数据被发送。
如果叶列表是“ordered-by user”，则表示叶列表项的 XML 元素必须以用户指定的顺序出现；否则，顺序依赖于实现。 表示叶列表项的 XML 元素可以与叶列表的同级元素交织，除非叶列表定义了 RPC 或动作输入(action input)或输出参数(output parameters)。
参见 7.7.10 节的例子。

7.7.9. NETCONF <edit-config>操作

叶列表条目可以通过使用叶列表条目的 XML 元素中的“operation”属性，通过<edit-config>创建和删除，但不能修改。
在“ordered-by user”的叶列表中，YANG XML 命名空间（第 5.3.1 节）中的属性“insert”和“value”可以用来控制叶列表中插入条目的位置。这些可以在“create”操作期间用于插入新的叶列表条目，或者在“merge”或“replace”操作期间插入新的叶列表条目或移动现有的叶列表条目。
“insert”属性可以取值“first”，“last”，“before”和“after”。如果值是“before”或“after”，那么“value”属性也必须用于指定叶列表中的现有条目。
如果在“create”操作中没有“insert”属性，则默认为“last”。
如果在“ordered-by user”叶列表中的几个条目在同一个<edit-config>请求中被修改，则按照请求中的 XML 元素的顺序一次修改一个条目。
在具有覆盖整个叶列表的“merge”操作的<copy-config>或<edit-config>中，叶列表顺序与请求中 XML 元素的顺序相同。
当一个 NETCONF 服务器处理一个<edit-config>请求时，叶列表节点的过程元素如下：

- 如果操作是“merge”或“replace”，则叶列表条目如果不存在则创建。
- 如果操作是“create”，则创建叶列表条目（如果不存在）。如果叶列表项已经存在，则返回“data-exists”错误。
- 如果操作是“delete”，则从叶列表中删除条目（如果存在）。如果叶列表项不存在，则返回“data-missing”错误。

7.7.10. 使用示例

```
leaf-list allow-user {
  type string;
  description
    "A list of user name patterns to allow.";
}
```

相应的 XML 实例示例：

```
<allow-user>alice</allow-user>
<allow-user>bob</allow-user>
```

要在此列表中创建一个新元素，请使用默认的<edit-config>操作“merge”：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <allow-user>eric</allow-user>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

给出以下按用户排序的列表：

```
leaf-list cipher {
  type string;
  ordered-by user;
  description
    "A list of ciphers.";
}
```

以下将被用于在“3des-cbc”之后插入新的密码“blowfish-cbc”：

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <services>
          <ssh>
            <cipher nc:operation="create"
                    yang:insert="after"
                    yang:value="3des-cbc">blowfish-cbc</cipher>
          </ssh>
        </services>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.8. “列表”声明

“`list`”语句用于定义架构树中的内部数据节点。 列表节点可能存在于数据树中的多个实例中。 每个这样的实例被称为列表条目。 “`list`”语句带有一个参数，这个参数是一个标识符，后跟一个包含详细列表信息的子语句块。
如果定义了列表条目，列表条目的唯一标识就是列表的键值。

7.8.1. `list` 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| action       | 7.15    | 0..n        |
| anydata      | 7.10    | 0..n        |
| anyxml       | 7.11    | 0..n        |
| choice       | 7.9     | 0..n        |
| config       | 7.21.1  | 0..1        |
| container    | 7.5     | 0..n        |
| description  | 7.21.3  | 0..1        |
| grouping     | 7.12    | 0..n        |
```

```
| if-feature   | 7.20.2  | 0..n        |
| key          | 7.8.2   | 0..1        |
| leaf         | 7.6     | 0..n        |
| leaf-list    | 7.7     | 0..n        |
| list         | 7.8     | 0..n        |
| max-elements | 7.7.6   | 0..1        |
| min-elements | 7.7.5   | 0..1        |
| must         | 7.5.3   | 0..n        |
| notification | 7.16    | 0..n        |
| ordered-by   | 7.7.7   | 0..1        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| typedef      | 7.3     | 0..n        |
| unique       | 7.8.3   | 0..n        |
| uses         | 7.13    | 0..n        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.8.2. 列表的“key”声明

“key”语句，如果列表表示配置，并且可能存在，则必须显示它，作为参数，该语句指定了该列表中的一个或多个叶标识符的空格分隔的列表。叶子标识符不能在 key 中出现一次以上。 每个这样的叶子标识符必须引用列表的子叶子。 叶子可以直接在子列表中定义，也可以在列表中使用分组。
key 中指定的所有叶子的组合值用于唯一标识列表条目。 在创建列表条目时，所有关键叶子必须被给定值。 因此，关键叶子或其类型中的任何默认值都将被忽略。 关键叶子中的任何“强制性”陈述都将被忽略。
作为 key 一部分的叶子可以是任何内置或派生类型。
列表中的所有关键字必须与列表本身具有相同的“配置”值。

key 字符串语法由第 14 节中的“key-arg”规则正式定义。

7.8.3. 列表的“唯一”声明

“unique”语句用于对有效的列表条目进行约束。 它以一个字符串作为参数，该字符串包含空间分隔的模式节点标识符列表，必须以后代形式给出（参见第 14 节中的“descendant-schema-nodeid”规则）。 每个这样的模式节点标识符必须指向一个叶子。
如果其中一个引用的叶子代表配置数据，那么所有引用的叶子必须代表配置数据。

“unique”约束指定参数字符串中指定的所有叶子实例的组合值，包括具有默认值的叶子，在所有引用叶子存在或具有默认值的所有列表项实例中必须是唯一的。 约束条件按照第 8 节的规定执行。
唯一的字符串语法由第 14 节中的“unique-arg”规则正式定义。

7.8.3.1. 使用示例

用下面的列表：

```
list server {
  key "name";
  unique "ip port";
  leaf name {
    type string;
  }
  leaf ip {
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
}
```

以下配置无效：

```
<server>
  <name>smtp</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>

<server>
  <name>http</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>
```

以下配置是有效的，因为“`http`”和“`ftp`”列表项不具有所有引用叶子的值，因此当“唯一”约束被强制执行时不被考虑：

```
<server>
  <name>smtp</name>
  <ip>192.0.2.1</ip>
  <port>25</port>
</server>

<server>
  <name>http</name>
  <ip>192.0.2.1</ip>
</server>

<server>
```

```
  <name>ftp</name>
  <ip>192.0.2.1</ip>
</server>
```

7.8.4. 列表的子节点语句

在列表中，可以使用“container”，“leaf”，“list”，“leaf-list”，“uses”，“choice”，“anydata”和“anyxml”语句来将子节点定义到列表。

7.8.5. XML 编码规则

列表被编码为一系列 XML 元素，列表中的每个条目都有一个。每个元素的本地名称是列表的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。整个列表中没有 XML 元素。

列表的关键节点被编码为列表的标识符元素的子元素，其顺序与“key”语句中定义的顺序相同。

列表的其余部分的子节点被编码为列表元素的子元素。如果列表定义了 RPC 或操作输入或输出参数，则子元素将按照“list”语句中定义的顺序进行编码。否则，子元素按任何顺序编码。

列表条目的子元素之间的任何空格是不重要的，即实现可以在子元素之间插入空白字符。

如果列表是“ordered-by user”，则表示列表条目的 XML 元素必须按用户指定的顺序出现;否则，顺序依赖于实现。表示列表条目的 XML 元素可以与列表的同级元素交错，除非列表定义了 RPC 或动作输入或输出参数。

7.8.6. NETCONF <edit-config>操作

列表条目可以通过使用列表 XML 元素中的“operation”属性通过<edit-config>创建，删除，替换和修改。在每种情况下，所有 key 的值都用于唯一标识列表条目。如果没有为列表项指定所有键，则返回“missing-element”错误。

在“ordered-by user”列表中，YANG XML 名称空间（第 5.3.1 节）中的属性“insert”和“key”可用于控制列表中插入条目的位置。这些可以在“create”操作期间用于插入新的列表条目，或在“merge”或“replace”操作期间插入新的列表条目或移动现有条目。

“insert”属性可以取值“first”，“last”，“before”和“after”。如果值是“before”或“after”，则还必须使用“key”属性来指定列表中的现有元素。 “key”属性的值是列表条目的完整实例标识符（见第 9.13 节）的关键谓词。

如果在“create”操作中没有“insert”属性，则默认为“last”。

如果“ordered-by user”列表中的多个条目在同一个<edit-config>请求中被修改，则按请求中 XML 元素的顺序一次修改一个条目。

在具有覆盖整个列表的“replace”操作的<copy-config>或<edit-config>中，列表输入顺序与请求中 XML 元素的顺序相同。

当 NETCONF 服务器处理<edit-config>请求时，列表节点的过程元素如下所示：

- 如果操作是“merge”或“replace”，则列表条目如果不存在则创建。如果列表条目已经存在，并且存在“insert”和“key”属性，则根据“insert”和“key”属性的值

移动列表条目。如果列表条目存在并且“insert”和“key”属性不存在，则列表条目不会移动。

- 如果操作是“create”，则列表条目如果不存在则创建。如果列表项已经存在，则返回“data-exists”错误。
- 如果操作是“delete”，则条目从列表中删除（如果存在）。如果列表条目不存在，则返回“data-missing”错误。

7.8.7. 使用示例

鉴于以下列表：

```
list user {
  key "name";
  config true;
  description
    "This is a list of users in the system.";

  leaf name {
    type string;
  }
  leaf type {
    type string;
  }
  leaf full-name {
    type string;
  }
}
```

相应的 XML 实例示例：

```
<user>
  <name>fred</name>
  <type>admin</type>
  <full-name>Fred Flintstone</full-name>
</user>
```

要创建一个新的用户“barney”：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <user nc:operation="create">
```

```
          <name>barney</name>
          <type>admin</type>
          <full-name>Barney Rubble</full-name>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

要将“fred”的类型更改为“superuser”：

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <user>
          <name>fred</name>
          <type>superuser</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

给出以下按用户列表：

```
list user {
  description
    "This is a list of users in the system.";
  ordered-by user;
  config true;

  key "first-name surname";

  leaf first-name {
    type string;
  }
  leaf surname {
    type string;
  }
  leaf type {
    type string;
  }
```

```
}
```

以下将用于在用户“fred flintstone”之后插入新用户“barney rubble”：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config"
          xmlns:ex="urn:example:config">
        <user nc:operation="create"
              yang:insert="after"
              yang:key="[ex:first-name='fred']
                        [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
          <type>admin</type>
        </user>
      </system>
    </config>
  </edit-config>
</rpc>
```

以下将用于在用户“fred flintstone”之前移动用户“barney rubble”：

```
<rpc message-id="102"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:yang="urn:ietf:params:xml:ns:yang:1">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config"
          xmlns:ex="urn:example:config">
        <user nc:operation="merge"
              yang:insert="before"
              yang:key="[ex:name='fred']
                        [ex:surname='flintstone']">
          <first-name>barney</first-name>
          <surname>rubble</surname>
        </user>
```

```
      </system>
    </config>
  </edit-config>
</rpc>
```

7.9. “选择”声明

“choice”语句定义了一组替代品，其中只有一个可能出现在任何一个数据树中。 参数是一个标识符，后面跟着一个包含详细选择信息的子状态块。 该标识符用于标识架构树中的选择节点。 数据树中不存在选择节点。
一个选择由若干分支组成，每个分支都定义一个“case”子语句。 每个分支都包含一些子节点。 选择分支中的至多一个节点同时存在。
由于在数据树中，只有一个选择的情况在任何时候都是有效的，所以从一种情况创建一个节点会隐式地删除所有其他情况下的所有节点。 如果一个请求从一个案例中创建一个节点，服务器将删除在其他情况下定义的所有现有节点。

7.9.1. choice 的子语句

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
case	7.9.2	0..n
choice	7.9	0..n
config	7.21.1	0..1
container	7.5	0..n
default	7.9.3	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
mandatory	7.9.4	0..1
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.9.2. 选择的"case"声明

“case”语句用于定义选择的分支。 它将一个标识符作为参数，后面跟着一个包含详细案例信息的子状态块。
标识符用于标识架构树中的 case 节点。 数据树中不存在 case 节点。
在“case”语句中，可以使用“anydata”，“anyxml”，“choice”，“container”，“leaf”，“list”，“leaf-list”和“uses”语句来定义子节点到 case 节点。 在选择的所有情况下，所有这些子节点的标识符必须是唯一的。 例如，以下是非法的：

```
choice interface-type {     // 这个例子是非法的 YANG
  case a {
    leaf ethernet { ... }
  }
  case b {
    container ethernet { ...}
  }
}
```

作为简写，如果分支包含单个“anydata”，“anyxml”，“choice”，“container”，“leaf”，“list”或“leaf-list”语句，则可以省略“case”语句。 在这种情况下，case 节点仍然存在于模式树中，其标识符与子节点的标识符相同。 模式节点标识符（6.5 节）必须总是明确地包含节点标识符。 下面的例子：

```
choice interface-type {
  container ethernet { ... }
}
```

相当于：

```
choice interface-type {
  case ethernet {
    container ethernet { ... }
  }
}
```

case 标识符在选择中必须是唯一的。

7.9.2.1. case 的子语句

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
reference	7.21.4	0..1

```
| status       | 7.21.2  | 0..1        |
| uses         | 7.13    | 0..n        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.9.3. 选择的“default”声明

如果没有来自任何选择情况的子节点存在，则“default”语句指示是否应将情况视为默认情况。参数是默认的“case”语句的标识符。如果“default”语句丢失，则没有默认情况。
“default”语句不能出现在“mandatory”为“true”的选项上。
默认情况下，仅在考虑情况下的节点的“default”语句（即，叶子和叶列表的默认值以及嵌套选择的默认情况）时才是重要的。如果任何情况下的节点都不存在，则使用缺省情况下的默认值和嵌套的默认情况。
在默认情况下，不能有任何强制节点（第 3 节）。

子 case 下的子节点的缺省值只有在该案例下的其中一个节点存在或者 case 是默认情况下才会使用。如果 case 中没有任何节点存在，并且 case 不是默认情况，则忽略 case 的子节点的默认值。
在本例中，选项默认为“interval”，如果没有“daily”，“time-of-day”或“manual”，则使用默认值。 如果“daily”存在，将使用“time-of-day”的默认值。

```
container transfer {
  choice how {
    default interval;
    case interval {
      leaf interval {
        type uint16;
        units minutes;
        default 30;
      }
    }
    case daily {
      leaf daily {
        type empty;
      }
      leaf time-of-day {
        type string;
        units 24-hour-clock;
        default "01.00";
      }
    }
    case manual {
      leaf manual {
        type empty;
      }
    }
```

```
  }
}
```

7.9.4. 选择的“mandatory”声明

“mandatory”语句是可选的，它将字符串“true”或“false”作为参数，并对有效数据进行约束。 如果“mandatory”是“true”，则必须至少有一个选择的案例分支中的一个节点存在。
如果未指定，则默认为“false”。
约束的行为取决于模式树中不是不存在容器的选择最接近的祖先节点的类型（请参阅第 7.5.1 节）：

- 如果模式树中不存在这样的祖先，则强制约束。
- 否则，如果这个祖先是一个 case 节点，那么如果这个 case 的任何其他节点存在，这个约束就会被执行。
- 否则，如果祖先节点存在，则强制执行。

根据第 8 节的规定，约束条件得到了进一步的执行。

7.9.5. XML 编码规则

选择和案例节点在 XML 中不可见。
所选“case”语句的子节点必须按照它们在“case”语句中定义的顺序编码，如果它们是 RPC 或动作输入(action input)或输出参数(output parameter)定义的一部分。 否则，子元素按任何顺序编码。

7.9.6. 使用示例

给出以下选择：

```
container protocol {
  choice name {
    case a {
      leaf udp {
        type empty;
      }
    }
    case b {
      leaf tcp {
        type empty;
      }
    }
  }
}
```

相应的 XML 实例示例：

```
<protocol>
  <tcp/>
</protocol>
```

将协议从 TCP 更改为 UDP：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
     xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <config>
      <system xmlns="urn:example:config">
        <protocol>
          <udp nc:operation="create"/>
        </protocol>
      </system>
    </config>
  </edit-config>
</rpc>
```

7.10. “anydata”声明

“anydata”语句在模式树中定义一个内部节点。它需要一个参数，它是一个标识符，后面是一个包含详细 anydata 信息的子状态块。

“anydata”语句用于表示可以用 YANG 建模的未知节点集合，但 anyxml 除外，但是在模块设计时不知道数据模型。对于任何数据内容的数据模型，虽然不是必需的，但是可以通过协议信令或本文档范围之外的其他手段来获知。

anydata 可能有用的示例是在设计时不知道具体通知的收到通知的列表。

anydata 节点不能被扩充（见 7.17 节）。

一个 anydata 节点存在于数据树的零个或一个实例中。

一个实现可能会或可能不知道用于建模 anydata 节点的特定实例的数据模型。

由于 anydata 的使用限制了内容的操作，所以不应该使用“anydata”语句来定义配置数据。

7.10.1. anydata 的子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| config       | 7.21.1  | 0..1        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
```

```
| mandatory    | 7.6.5   | 0..1        |
| must         | 7.5.3   | 0..n        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.10.2. XML 编码规则

一个 anydata 节点被编码为一个 XML 元素。 元素的本地名称是 anydata 的标识符，其名称空间是模块的 XML 名称空间（请参阅第 7.1.3 节）。 anydata 节点的值是一组节点，它们被编码为 anydata 元素的 XML 子元素。

7.10.3. NETCONF <edit-config>操作

一个 anydata 节点被视为一个不透明的数据块。 这些数据只能被整体修改。 NETCONF 服务器将忽略 anydata 节点子元素上的任何“operation”属性。

当 NETCONF 服务器处理<edit-config>请求时， anydata 节点的过程元素如下所示：

- 如果操作是“merge”或“replace”，则创建该节点（如果该节点不存在），并将其值设置为 XML RPC 数据中找到的 anydata 节点的子元素。
- 如果操作是“create”，则创建该节点（如果该节点不存在），并将其值设置为 XML RPC 数据中找到的 anydata 节点的子元素。 如果节点已经存在，则返回“data-exists”错误。
- 如果操作是“delete”，则该节点将被删除（如果存在）。 如果节点不存在，则返回“data-missing”错误。

7.10.4. 使用示例

鉴于以下“anydata”声明：

```
list logged-notification {
  key time;
  leaf time {
    type yang:date-and-time;
  }
  anydata data;
}
```

以下是这种列表实例的有效编码：

```
<time>2014-07-29T13:43:12Z</time>
<data>
  <notification
    xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <eventTime>2014-07-29T13:43:01Z</eventTime>
    <event xmlns="urn:example:event">
      <event-class>fault</event-class>
```

```
      <reporting-entity>
        <card>Ethernet0</card>
      </reporting-entity>
      <severity>major</severity>
    </event>
  </notification>
</data>
</logged-notification>
```

7.11. “anyxml”声明

“anyxml”语句在模式树中定义一个内部节点。它需要一个参数，它是一个标识符，后面是一个包含 anyxml 信息的子状态块。

“anyxml”语句用于表示未知的 XML 块。 XML 没有限制。例如，在 RPC 回复中这可能很有用。 NETCONF 中的<get-config>操作中的<filter>参数就是一个例子。

anyxml 节点不能被扩充（见 7.17 节）。

数据树中的零个或一个实例中存在 anyxml 节点。

由于使用 anyxml 限制了对内容的操作，所以不应该使用“anyxml”语句来定义配置数据。

应该指出的是，在 YANG 版本 1 中，“anyxml”是唯一可以模拟未知数据层次的声明。在很多情况下，这个未知的数据层次实际上是在 YANG 中建模的，但具体的 YANG 数据模型在设计时并不知道。在这些情况下，建议使用“anydata”（第 7.10 节）而不是“anyxml”。

7.11.1. anyxml 子语句

substatement	section	cardinality
config	7.21.1	0..1
description	7.21.3	0..1
if-feature	7.20.2	0..n
mandatory	7.6.5	0..1
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
when	7.21.5	0..1

7.11.2. XML 编码规则

一个 anyxml 节点被编码为一个 XML 元素。 元素的本地名称是 anyxml 的标识符，其名称空间是模块的 XML 名称空间（请参阅第 7.1.3 节）。 anyxml 节点的值被编码为这个元素的 XML 内容。
请注意，编码中使用的任何 XML 前缀对于每个实例编码都是本地的。 这意味着相同的 XML 可能会被不同的实现编码。

7.11.3. NETCONF <edit-config>操作

anyxml 节点被视为不透明的数据块。 这些数据只能被整体修改。
NETCONF 服务器忽略 anyxml 节点子元素上的任何“operation”属性。
当 NETCONF 服务器处理<edit-config>请求时，anyxml 节点的过程元素如下所示：

- 如果操作是“merge”或“replace”，则创建该节点（如果该节点不存在），并将其值设置为 XML RPC 数据中找到的 anyxml 节点的 XML 内容。
- 如果操作是“create”，则创建该节点（如果该节点不存在），并将其值设置为 XML RPC 数据中找到的 anyxml 节点的 XML 内容。 如果节点已经存在，则返回“data exists”错误。
- 如果操作是“delete”，则该节点将被删除（如果存在）。 如果节点不存在，则返回“data missing”错误。

7.11.4. 使用示例

鉴于以下“anyxml”声明：

```
anyxml html-info;
```

以下是相同的 anyxml 值的两个有效的编码：

```
<html-info>
  <p xmlns="http://www.w3.org/1999/xhtml">
    This is <em>very</em> cool.
  </p>
</html-info>

<html-info>
  <x:p xmlns:x="http://www.w3.org/1999/xhtml">
    This is <x:em>very</x:em> cool.
  </x:p>
</html-info>
```

7.12. "分组"声明

“grouping”语句用于定义一个可重用的节点块，这些节点可以在模块或子模块中本地使用，也可以根据第 5.5 节中的规则使用其他模块。它需要一个参数，它是一个标识符，后面是一个包含详细分组信息的子语句块。
“grouping”语句不是一个数据定义语句，因此，它不定义模式树中的任何节点。

分组就像传统编程语言中的“结构(structure)”或“记录(record)”。
一旦定义了分组，就可以在“uses”语句中引用它(参见 7.13)。一个分组不能引用自己，也不能直接或间接地通过其他分组。
如果分组是在 YANG 模块或子模块的顶层定义的，那么分组的标识符必须在模块中是唯一的。
分组不仅仅是一种文本替换的机制;它还定义了一个节点集合。在分组中出现的标识符相对于定义分组的作用域，而不是在使用的位置。前缀映射、类型名称、分组名称和扩展使用都在“grouping”语句出现的层次结构中进行了评估。对于扩展，这意味着将直接子定义为“grouping”语句的扩展被应用到分组本身。
注意，如果一个分组在它的层次结构中定义了一个动作或一个通知节点，那么它就不能在所有的上下文中使用。例如，它不能在 rpc 定义中使用。参见第 7.15 和 7.16 节。

7.12.1. grouping 子语句

substatement	section	cardinality
action	7.15	0..n
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
notification	7.16	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.12.2 使用示例

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

grouping endpoint {
  description "A reusable endpoint group.";
  leaf ip {
```

```
    type inet:ip-address;
  }
  leaf port {
    type inet:port-number;
  }
}
```

7.13. “使用”声明

“uses”语句用于引用“grouping”定义。 它需要一个参数，这是分组的名称。
“uses”引用对分组的作用是将分组定义的节点复制到当前模式树中，然后根据“精简(according)”和“增加(augment)”语句进行更新。
在分组中定义的标识符没有绑定到名称空间，直到分组的内容通过“uses”语句添加到模式树中，该语句不出现在“grouping”语句中，此时它们被绑定到名称空间的当前模块。

7.13.1. uses 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| augment      | 7.17    | 0..n        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
| reference    | 7.21.4  | 0..1        |
| refine       | 7.13.2  | 0..n        |
| status       | 7.21.2  | 0..1        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.13.2. “refine”声明

分组中每个节点的一些属性可以通过“refine”语句来改进。参数是标识分组中的节点的字符串。这个节点被称为细化的目标节点。如果分组中的节点不存在作为“细化”语句的目标节点，则不会对其进行细化，因此可以完全按照在分组中定义的方式使用。
参数字符串是后代模式节点标识符（请参阅第 6.5 节）。

以下改进可以完成：

- 叶节点或选择节点可能会得到一个默认值，或者一个新的默认值，如果它已经有一个。
- 一个叶子列表节点可以获得一组默认值，或者一组新的默认值（如果它已经有了默认值）即精炼的默认值的集合取代了已经给出的默认值。
- 任何节点都可能获得专门的“description”字符串。

- 任何节点都可能获得专门的“reference”字符串。
- 任何节点可能会得到不同的“config”语句。
- leaf，anydata，anyxml 或 choice 节点可能会得到不同的 mandatory”语句。
- 一个容器节点可能会得到一个“presence”语句。
- 叶，叶列表，列表，容器，anydata 或 anyxml 节点可能会获得额外的“must”表达式。
- 叶子列表或列表节点可能会得到不同的“最小元素(min-elements)”或“最大元素(max-elements)”语句。
- 叶，叶列表，列表，容器，选项，大小写，anydata 或 anyxml 节点可能会获得额外的“if-feature”表达式。
- 如果扩展允许细化，任何节点都可以得到细化的扩展。详情请参阅 7.19 节。

7.13.3. XML 编码规则

即使从另一个 XML 名称空间导入另一个模块，分组中的每个节点都被编码，就好像它是内联定义的一样。

7.13.4. 使用示例

要使用 7.12.2 节中定义的“endpoint”分组来定义某个其他模块中的 HTTP 服务器，我们可以这样做：

```
import example-system {
  prefix "sys";
}

container http-server {
  leaf name {
    type string;
  }
  uses sys:endpoint;
}
```

相应的 XML 实例示例：

```
<http-server>
  <name>extern-web</name>
  <ip>192.0.2.1</ip>
  <port>80</port>
</http-server>
```

如果端口 80 应该是 HTTP 服务器的默认端口，则可以添加一个默认值：

```
container http-server {
  leaf name {
    type string;
  }
  uses sys:endpoint {
    refine port {
```

```
      default 80;
    }
  }
}
```

如果我们要定义一个服务器列表，并且每个服务器都有“ip”和“port”作为键，我们可以这样做：

```
list server {
  key "ip port";
  leaf name {
    type string;
  }
  uses sys:endpoint;
}
```

以下是一个错误：

```
container http-server {
  uses sys:endpoint;
  leaf ip {          // 非法 - 相同的标识符“ip”
    type string;
  }
}
```

7.14. “rpc”声明

“rpc”语句用于定义 RPC 操作。 它需要一个参数，它是一个标识符，后面是一个包含详细 rpc 信息的子语句块。 这个参数是 RPC 的名字。

“rpc”语句在模式树中定义一个 rpc 节点。 在 rpc 节点下，还定义了一个名为“input”的模式节点和一个名为“output”的模式节点。 节点“input”和“output”在模块的命名空间中定义。

7.14.1. rpc 子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| description  | 7.21.3  | 0..1        |
| grouping     | 7.12    | 0..n        |
| if-feature   | 7.20.2  | 0..n        |
| input        | 7.14.2  | 0..1        |
| output       | 7.14.3  | 0..1        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| typedef      | 7.3     | 0..n        |
+--------------+---------+-------------+
```

7.14.2. “输入”声明

“input”语句是可选的，用于定义操作的输入参数。它没有争论。 “input”子语句定义操作输入节点下的节点。
如果输入树中的一个叶子有一个值为“true”的“mandatory”语句，叶子务必出现在一个 RPC 调用中。
如果输入树中的一个叶子有一个默认值，那么服务器必须在 7.6.1 节描述的情况下使用这个值。在这些情况下，服务器必须在操作上表现得像叶子出现在 RPC 调用中一样，其默认值是其值。
如果输入树中的叶子列表有一个或多个默认值，那么服务器必须在 7.7.2 节中描述的那些情况下使用这些值。在这些情况下，服务器必须在操作上表现得像叶子列表出现在 RPC 调用中一样，其默认值作为其值。
由于输入树不是任何数据存储区的一部分，因此输入树中所有节点的“config”语句都将被忽略。
如果任何节点有一个“when”语句，将评估为“false”，那么这个节点绝不能出现在输入树中。

7.14.2.1. input 子语句

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.3. “输出”声明

“output”语句是可选的，用于定义 RPC 操作的输出参数。它没有争论。
“output”子状态定义操作的输出节点下的节点。
如果输出树中的一个叶子有一个值为“true”的“mandatory”语句，叶子务必出现在 RPC 回复中。
如果输出树中的一个叶子有一个默认值，那么客户端必须在 7.6.1 节描述的情况下使用这个值。在这些情况下，客户端必须在操作上表现得如同叶子出现在 RPC 回复中一样，其默认值为其值。

如果输出树中的叶子列表有一个或多个默认值，那么客户端必须在 7.7.2 节中描述的情况下使用这些值。在这些情况下，客户端必须在操作上表现得好像叶子列表出现在 RPC 回复中，默认值作为其值。
由于输出树不是任何数据存储区的一部分，因此输出树中所有节点的“config”语句都将被忽略。
如果任何节点有一个“when”语句，其结果为“false”，那么这个节点绝不能出现在输出树中。

7.14.3.1. output 子语句

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
grouping	7.12	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
typedef	7.3	0..n
uses	7.13	0..n

7.14.4. NETCONF XML 编码规则

rpc 节点被编码为<rpc>元素的子 XML 元素，由[RFC6241]中的替换组“rpcOperation”指定。 元素的本地名称是 rpc 的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。
输入参数按照在“input”语句中定义的顺序，作为子 XML 元素编码到 rpc 节点的 XML 元素。
如果 RPC 操作调用成功并且没有返回输出参数，则<rpc-reply>包含在[RFC6241]中定义的单个<ok/>元素。 如果返回输出参数，则按照在“output”语句中定义的顺序，将它们作为子元素编码到[RFC6241]中定义的<rpc-reply>元素。

7.14.5. 使用示例

以下示例定义了一个 RPC 操作：

```
module example-rock {
  yang-version 1.1;
  namespace "urn:example:rock";
  prefix "rock";
```

```
  rpc rock-the-house {
    input {
      leaf zip-code {
        type string;
      }
    }
  }
}
```

完整 rpc 和 rpc-reply 的相应 XML 实例示例：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rock-the-house xmlns="urn:example:rock">
    <zip-code>27606-0100</zip-code>
  </rock-the-house>
</rpc>

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

7.15. “动作”声明

“action”语句用于定义连接到特定容器或列表数据节点的操作。它有一个参数，它是一个标识符，后面是一个包含详细动作信息的子语句块。参数是动作的名称。

“action”语句在模式树中定义一个动作节点。在 action 节点下，还定义了名称为“input”的模式节点和名称为“output”的模式节点。节点“input”和“output”在模块的命名空间中定义。

action 不能在 rpc，另一个 action 或通知中定义，即 action 节点不能在模式树中有一个 rpc，action 或通知节点作为其祖先之一。例如，这意味着如果在通知定义中使用在其节点层次结构中某处包含某个操作的分组，则这是一个错误。

一个 action 不能有一个没有“key”语句的列表节点的祖先节点。

由于无法在模块的顶层或“case”语句中定义操作，因此如果在模块的顶层使用包含其节点层次结构顶部的操作的分组，则会出错一个案例的定义。

action 和 rpc 之间的区别在于 action 绑定到数据存储中的节点，而 rpc 不是。当一个 action 被调用时，数据存储中的节点与 action 名称和输入参数一起被指定。

7.15.1. action 子语句

```
+--------------+---------+-------------+
```

```
| substatement | section | cardinality |
+--------------+---------+-------------+
| description  | 7.21.3  | 0..1        |
| grouping     | 7.12    | 0..n        |
| if-feature   | 7.20.2  | 0..n        |
| input        | 7.14.2  | 0..1        |
| output       | 7.14.3  | 0..1        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| typedef      | 7.3     | 0..n        |
+--------------+---------+-------------+
```

7.15.2. NETCONF XML 编码规则

当一个 action 被调用时，名字空间“urn:ietf:params:xml:ns:yang:1”中的本地名称为“action”的元素（见第 5.3.1 节）被编码为子 XML 元素， [RFC6241]中定义的<rpc>元素，如[RFC6241]中的替换组“rpcOperation”所指定的。
<action>元素包含标识数据存储区中节点的节点层次结构。它必须包含从顶层到包含动作的列表或容器的直接路径中的所有容器和列表节点。对于列表，所有关键叶子也必须包括在内。最内层的容器或列表包含一个 XML 元素，其中包含已定义操作的名称。在这个元素中，输入参数被编码为子 XML 元素，其顺序与“input”语句中定义的顺序相同。
一个<rpc>中只能调用一个 action。如果<rpc>中存在多个 action，则服务器必须在<rpc-error>中回答“bad-element”<error-tag>。
如果操作调用成功并且没有返回输出参数，则<rpc-reply>包含在[RFC6241]中定义的单个<ok/>元素。如果返回输出参数，则按照在“output”语句中定义的顺序，将它们作为子元素编码到[RFC6241]中定义的<rpc-reply>元素。

7.15.3. 使用示例

以下示例定义一个操作来重置服务器集群中的一台服务器：

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

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

  list server {
    key name;
    leaf name {
      type string;
```

```
    }
    action reset {
      input {
        leaf reset-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

完整 rpc 和 rpc-reply 的相应 XML 实例示例：

```
<rpc message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <action xmlns="urn:ietf:params:xml:ns:yang:1">
    <server xmlns="urn:example:server-farm">
      <name>apache-1</name>
      <reset>
        <reset-at>2014-07-29T13:42:00Z</reset-at>
      </reset>
    </server>
  </action>
</rpc>

<rpc-reply message-id="101"
           xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <reset-finished-at xmlns="urn:example:server-farm">
    2014-07-29T13:42:12Z
  </reset-finished-at>
</rpc-reply>
```

7.16. “通知”声明

“notification”语句用于定义通知。它需要一个参数，它是一个标识符，后面是一个包含详细通知信息的子状态块。 “notification”语句在模式树中定义了一个通知节点。

通知可以在模块的顶层定义，也可以连接到模式树中的特定容器或列表数据节点。

通知不能在 `rpc`，`action` 或其他通知中定义，即通知节点不能在模式树中有一个 `rpc`，`action` 或通知节点作为其祖先之一。例如，这意味着如果在 `rpc` 定义中使用在其节点层次结构中包含通知的分组，则这是一个错误。
一个通知不能有一个没有“`key`”语句的列表节点的祖先节点。
由于无法在“`case`”语句中定义通知，因此如果在案例定义中使用在其节点层次结构顶部包含通知的分组，则会发生错误。
如果通知树中的某个叶子具有值为“`true`”的“`mandatory`”语句，叶子务必出现在通知实例中。
如果通知树中的一个叶子有一个默认值，那么客户端必须在 7.6.1 节描述的情况下使用这个值。 在这些情况下，客户端必须在操作上表现得像叶子出现在通知实例中一样，其默认值为其值。

如果通知树中的叶子列表有一个或多个默认值，则客户端必须使用与 7.7.2 节中描述的相同的情况下的这些值。 在这些情况下，客户端必须在操作上像在默认值作为其值的情况下在叶子列表中出现在通知实例中一样。

由于通知树不是任何数据存储区的一部分，因此通知树中所有节点的“`config`”语句都将被忽略。

7.16.1. `notification` 子语句

substatement	section	cardinality
anydata	7.10	0..n
anyxml	7.11	0..n
choice	7.9	0..n
container	7.5	0..n
description	7.21.3	0..1
grouping	7.12	0..n
if-feature	7.20.2	0..n
leaf	7.6	0..n
leaf-list	7.7	0..n
list	7.8	0..n
must	7.5.3	0..n
reference	7.21.4	0..1
status	7.21.2	0..1
typedef	7.3	0..n
uses	7.13	0..n

7.16.2. `NETCONF` `XML` 编码规则

在模块顶层定义的通知节点被编码为“NETCONF Event Notifications”[RFC5277]中定义的<notification>元素的子 XML 元素。 元素的本地名称是通知的标识符，其名称空间是模块的 XML 名称空间（参见第 7.1.3 节）。
当通知节点被定义为数据节点的子节点时，[RFC5277]中定义的<notification>元素包含标识数据存储区中节点的节点层次结构。 它必须包含从顶层到包含通知的列表或容器的所有容器和列表节点。 对于列表，所有关键叶子也必须包括在内。 最里面的容器或列表包含一个 XML 元素，它包含定义的通知的名称。
通知的子节点以任何顺序编码为通知节点的 XML 元素的子元素。

7.16.3. 使用示例

以下示例在模块的顶层定义通知：

```
module example-event {
  yang-version 1.1;
  namespace "urn:example:event";
  prefix "ev";

  notification event {
    leaf event-class {
      type string;
    }
    leaf reporting-entity {
      type instance-identifier;
    }
    leaf severity {
      type string;
    }
  }
}
```

完整通知的相应 XML 实例示例：

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-07-08T00:01:00Z</eventTime>
  <event xmlns="urn:example:event">
    <event-class>fault</event-class>
    <reporting-entity>
      /ex:interface[ex:name='Ethernet0']
    </reporting-entity>
    <severity>major</severity>
  </event>
</notification>
```

以下示例在数据节点中定义通知：

```
module example-interface-module {
  yang-version 1.1;
  namespace "urn:example:interface-module";
  prefix "if";

  container interfaces {
    list interface {
      key "name";
      leaf name {
        type string;
      }
      notification interface-enabled {
        leaf by-user {
          type string;
        }
      }
    }
  }
}
```

完整通知的相应 XML 实例示例：

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-07-08T00:01:00Z</eventTime>
  <interfaces xmlns="urn:example:interface-module">
    <interface>
      <name>eth1</name>
      <interface-enabled>
        <by-user>fred</by-user>
      </interface-enabled>
    </interface>
  </interfaces>
</notification>
```

7.17. “扩充”声明

“augment”语句允许模块或子模块添加到在外部模块或当前模块及其子模块中定义的模式树中，并通过“uses”语句中的分组添加到节点中。 参数是标识架构树中节点的字符串。 这个节点被称为增广的目标节点。 目标节点必须是 container，list，choice，case，input，output 或 notification 节点。 随着“augment”语句之后的子语句中定义的节点的增加。

参数字符串是一个模式节点标识符（参见第 6.5 节）。如果“augment”语句位于模块或子模块的顶层，必须使用模式节点标识符的绝对格式（由第 14 节中的规则“absolute-schema-nodeid”定义）。如果“augment”语句是“uses”语句的子语句，则必须使用后代形式（由第 14 节中的规则“descendant-schema-nodeid”定义）。

如果目标节点是容器，列表，case，输入，输出或通知节点，则“container”，“leaf”，“list”，“leaf-list”，“uses”和“choice”语句可以是在“augment”声明中使用。

如果目标节点是容器或列表节点，则可以在“augment”语句中使用“Operation”和“notification”语句。

如果目标节点是一个 choice 节点，则可以在“augment”语句中使用“case”语句或简写“case”语句（参见第 7.9.2 节）。

“augment”语句绝不能将同一个模块中具有相同名称的多个节点添加到目标节点。

如果增加在另一个模块中添加了表示配置到目标节点的强制节点（参见第 3 节），则必须使用“when”语句使该扩充有条件。定义“when”表达式时必须小心，以避免不知道扩充模块的客户端中断。

在下面的例子中，用“mandatory-leaf”扩充“interface”条目是可以的，因为增加依赖于对“some-new-iftype”的支持。旧客户端不知道这种类型，所以它不会选择这种类型，因此不会添加强制数据节点。

```
module example-augment {
  yang-version 1.1;
  namespace "urn:example:augment";
  prefix mymod;

  import ietf-interfaces {
    prefix if;
  }

  identity some-new-iftype {
     base if:interface-type;
  }
  augment "/if:interfaces/if:interface" {
     when 'derived-from-or-self(if:type, "mymod:some-new-iftype")';

     leaf mandatory-leaf {
        mandatory true;
        type string;
     }
  }
}
```

7.17.1. augment 子语句

```
+--------------+---------+-------------+
```

```
| substatement | section | cardinality |
+--------------+---------+-------------+
| action       | 7.15    | 0..n        |
| anydata      | 7.10    | 0..n        |
| anyxml       | 7.11    | 0..n        |
| case         | 7.9.2   | 0..n        |
| choice       | 7.9     | 0..n        |
| container    | 7.5     | 0..n        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
| leaf         | 7.6     | 0..n        |
| leaf-list    | 7.7     | 0..n        |
| list         | 7.8     | 0..n        |
| notification | 7.16    | 0..n        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| uses         | 7.13    | 0..n        |
| when         | 7.21.5  | 0..1        |
+--------------+---------+-------------+
```

7.17.2. XML编码规则

在“augment”语句中定义的所有数据节点都被定义为指定“augment”的模块的 XML 名称空间中的 XML 元素。
当一个节点被扩充时，扩充子节点以任何顺序被编码为扩充节点的子元素。

7.17.3. 使用示例

在命名空间 urn:example:interface-module，我们有：

```
container interfaces {
  list ifEntry {
    key "ifIndex";

    leaf ifIndex {
      type uint32;
    }
    leaf ifDescr {
      type string;
    }
    leaf ifType {
      type iana:IfType;
    }
    leaf ifMtu {
      type int32;
    }
```

```
  }
}
```

然后，在命名空间 urn:example:ds0，我们有：

```
import example-interface-module {
  prefix "if";
}
augment "/if:interfaces/if:ifEntry" {
  when "if:ifType='ds0'";
  leaf ds0ChannelNumber {
    type ChannelNumber;
  }
}
```

相应的 XML 实例示例：

```
<interfaces xmlns="urn:example:interface-module"
            xmlns:ds0="urn:example:ds0">
  <ifEntry>
    <ifIndex>1</ifIndex>
    <ifDescr>Flintstone Inc Ethernet A562</ifDescr>
    <ifType>ethernetCsmacd</ifType>
    <ifMtu>1500</ifMtu>
  </ifEntry>
  <ifEntry>
    <ifIndex>2</ifIndex>
    <ifDescr>Flintstone Inc DS0</ifDescr>
    <ifType>ds0</ifType>
    <ds0:ds0ChannelNumber>1</ds0:ds0ChannelNumber>
  </ifEntry>
</interfaces>
```

作为另一个例子，假设我们有 7.9.6 节定义的选择。 以下构造可用于扩展协议定义：

```
augment /ex:system/ex:protocol/ex:name {
  case c {
    leaf smtp {
      type empty;
    }
  }
}
```

相应的 XML 实例示例：

```
<ex:system>
  <ex:protocol>
    <ex:tcp/>
  </ex:protocol>
</ex:system>
```

或

```
<ex:system>
  <ex:protocol>
    <other:smtp/>
  </ex:protocol>
</ex:system>
```

7.18. “身份”声明

“identity”声明用于定义新的全球唯一，抽象和无类型的身份。 身份的唯一目的是表示它的名字，语义和存在。 身份可以从头开始定义，也可以从一个或多个基本身份派生。 身份的参数是一个标识符，是身份的名称。 紧接着是一个包含详细身份信息的子状态块。
内置数据类型“identityref”（见第 9.10 节）可用于引用数据模型中的身份。
7.18.1. 身份的子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| base         | 7.18.2  | 0..n        |
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
+--------------+---------+-------------+
```

7.18.2. “base”声明

可选的“base”语句将一个字符串作为参数，该字符串是从中派生新身份的现有身份的名称。 如果不存在“base”语句，则从头开始定义标识。 如果存在多个“base”语句，则从所有这些语句中派生出身份。
如果基本名称上存在前缀，则表示在前缀导入的模块中定义的标识;如果前缀与本地模块的前缀匹配，则表示前缀为本地模块。 否则，必须在当前模块或包含的子模块中定义具有匹配名称的标识。

身份不得直接或间接通过其他身份链来引用自己。

身份的推导具有以下属性：

- 这是不反射的，这意味着一个身份不是从它本身衍生出来的。
- 它是传递性的，这意味着如果 B 的身份是从 A 派生的，C 是从 B 派生的，那么 C 也是从 A 派生的。

7.18.3. 使用示例

```
module example-crypto-base {
  yang-version 1.1;
  namespace "urn:example:crypto-base";
  prefix "crypto";

  identity crypto-alg {
    description
      "Base identity from which all crypto algorithms
       are derived.";
  }

  identity symmetric-key {
    description
      "Base identity used to identify symmetric-key crypto
       algorithms.";
    }

  identity public-key {
    description
      "Base identity used to identify public-key crypto
       algorithms.";
    }
}

module example-des {
  yang-version 1.1;
  namespace "urn:example:des";
  prefix "des";

  import "example-crypto-base" {
    prefix "crypto";
  }

  identity des {
    base "crypto:crypto-alg";
    base "crypto:symmetric-key";
    description "DES crypto algorithm.";
  }
  identity des3 {
    base "crypto:crypto-alg";
    base "crypto:symmetric-key";
    description "Triple DES crypto algorithm.";
  }
}
```

7.19. “扩展”声明

“extension”语句允许定义 YANG 语言中的新语句。这个新的语句定义可以被其他模块导入和使用。
“extension”语句的参数是一个标识符，它是扩展名的新关键字，后面必须跟着一个包含详细扩展信息的子语句块。 “extension”语句的目的是定义一个关键字，使其可以被其他模块导入和使用。
扩展名可以像普通的 YANG 语句一样使用，如果语句名由“extension”语句定义，则语句名后跟一个参数，并且可以包含一个可选的子语句块。该语句的名称是通过组合定义扩展名的模块的前缀，冒号（“:”）和扩展的关键字来创建的，没有交织空白。扩展的子语句由“extension”语句定义，使用本规范范围之外的某种机制。语法上，子语句必须是 YANG 语句，包括使用“extension”语句定义的扩展名。在扩展中的 YANG 语句必须遵循第 14 节中的语法规则。
扩展可以允许细化（参见 7.13.2 节）和偏差（参见 7.20.3.2），但是定义的机制超出了本规范的范围。

7.19.1. extension 子语句

substatement	section	cardinality
argument	7.19.2	0..1
description	7.21.3	0..1
reference	7.21.4	0..1
status	7.21.2	0..1

7.19.2. “argument”声明

可选的“argument”语句将一个字符串作为参数，该字符串是关键字参数的名称。 如果不存在“argument”语句，则关键字在使用时不需要参数。
这个参数的名字在 YIN 映射中使用，它被用作 XML 属性或元素名称，具体取决于参数的“yin-element”语句。

7.19.2.1. “argument”子语句

substatement	section	cardinality
yin-element	7.19.2.2	0..1

7.19.2.2. “yin-element”声明

“yin-element”语句是可选的，它将字符串“true”或“false”作为参数。 此语句指示参数是映射到 YIN 中的 XML 元素还是映射到 XML 属性（请参阅第 13 节）。 如果没有“yin-element”语句，则默认为“false”。

7.19.3. 使用示例

要定义一个扩展：

```
module example-extensions {
  yang-version 1.1;
  ...

  extension c-define {
    description
      "Takes as an argument a name string.
       Makes the code generator use the given name
       in the #define.";
    argument "name";
  }
}
```

要使用扩展名：

```
module example-interfaces {
  yang-version 1.1;

  ...
  import example-extensions {
    prefix "myext";
  }
  ...

  container interfaces {
    ...
    myext:c-define "MY_INTERFACES";
  }
}
```

7.20. 一致性相关的陈述

本节定义了与符合性有关的声明，如 5.6 节所述。

7.20.1. “feature”声明

“feature”语句用于定义一个机制，模式的一部分被标记为有条件的。 定义了一个功能名称，稍后可以使用“if-feature”语句来引用（参见第 7.20.2 节）。除非服务器支持给定的特征表达式，否则服务器将忽略使用“if-feature”语句标记的模式节点。 这允许 YANG 模块的部分基于服务器中的条件有条件。 该模型可以表示模型中的服务器的能力，给出一个更丰富的模型，允许不同的服务器能力和角色。

“feature”语句的参数是新特征的名称，并遵循 6.2 节中有关标识符的规则。“if-feature”语句使用此名称将模式节点绑定到该功能。

在这个例子中，一个名为“本地存储(local-storage)”的特性表示服务器将 syslog 消息存储在某种本地存储上的能力。 此功能用于使“本地存储限制(local-storage-limit)”以存在某种本地存储为条件。 如果服务器不报告它支持此功能，则不支持“本地存储限制”节点。

```
module example-syslog {
  yang-version 1.1;

  ...
  feature local-storage {
    description
      "This feature means that the server supports local
       storage (memory, flash, or disk) that can be used to
       store syslog messages.";
  }

  container syslog {
    leaf local-storage-limit {
      if-feature local-storage;
      type uint64;
      units "kilobyte";
      config false;
      description
        "The amount of local storage that can be
         used to hold syslog messages.";
    }
  }
}
```

YANG 语法中的许多地方都可以使用“if-feature”语句。 当服务器不支持该功能时，将忽略带有“if-feature”标签的定义。

一个功能不能直接或间接地通过一系列其他功能来引用它自己。

为了使服务器支持依赖于任何其他特征的特征（即特征具有一个或多个“if-feature”子特征），服务器还必须支持所有从属特征。

7.20.1.1. `feature` 子语句

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
reference	7.21.4	0..1
status	7.21.2	0..1

7.20.2. “`if-feature`”声明

“`if-feature`”语句使其父语句有条件。参数是通过功能名称的布尔表达式。在此表达式中，当且仅当服务器支持该功能时，功能名称的计算结果为“`true`”。父语句由布尔表达式求值为“`true`”的服务器实现。
`if-feature` 布尔表达式语法由第 14 节中的规则“`if-feature-expr`”正式定义。圆括号用于对表达式进行分组。评估表达式时，优先顺序为（最高优先优先）：分组（括号），“`not`”，“`and`”，“`or`”。
如果布尔表达式中的特征名称上存在前缀，则前缀名称是指使用该前缀导入的模块中定义的特征，如果前缀匹配本地模块的前缀，则使用本地模块。否则，必须在当前模块或包含的子模块中定义具有匹配名称的特征。

作为列表键的叶子不能有任何“`if-feature`”语句。

7.20.2.1 使用示例

在本例中，如果服务器支持“`outbound-tls`”或“`outbound-ssh`”特性，则实现容器“`target`”。

```
container target {
  if-feature "outbound-tls or outbound-ssh";
  ...
}
```

以下示例是等效的：

```
if-feature "not foo or bar and baz";

if-feature "(not foo) or (bar and baz)";
```

7.20.3. “`deviation`”声明

“`deviation`”语句定义了服务器不能忠实执行的模块层次结构。参数是一个字符串，用于标识架构树中与模块发生偏差的节点。这个节点被称为偏差的目标节点。 “`deviation`”声明的内容给出了有关偏差的详细信息。
参数字符串是一个绝对的模式节点标识符（请参阅第 6.5 节）。

偏差定义了服务器或服务器类别偏离标准的方式。这意味着偏差必须不是公布的标准的一部分，因为它们是学习实现如何与标准不同的机制。

服务器偏差是强烈的不鼓励，只能作为最后的手段。告诉应用程序如何服务器不能遵循标准是正确实施标准的替代品。偏离模块的服务器不完全符合模块。

但是，在某些情况下，特定设备可能没有硬件或软件支持标准模块的某些部分的能力。发生这种情况时，服务器可以选择将对模块的不支持部分进行配置的尝试视为报告给毫无疑问的应用程序的错误，或者忽略这些传入的请求。这两个选择都不可以接受。

相反，YANG 允许服务器通过使用“deviation”语句来记录基础模块中不被支持，或者被支持但语法不同的部分。
在应用服务器公布的所有偏差之后，以任何顺序，产生的数据模型必须仍然有效。

7.20.3.1. deviation 子语句

substatement	section	cardinality
description	7.21.3	0..1
deviate	7.20.3.2	1..n
reference	7.21.4	0..1

7.20.3.2。 “deviate”声明

“deviate”语句定义了目标节点的服务器实现如何偏离其原始定义。参数是字符串“not-supported”，“add”，“replace”或“delete”之一。
参数“not-supported”表示目标节点没有被这个服务器实现。
参数“add”向目标节点添加属性。要添加的属性由子状态标识为“deviate”语句。如果一个属性只能出现一次，那么这个属性不能存在于目标节点中。
参数“replace”取代了目标节点的属性。要替换的属性由子状态标识为“deviate”语句。要替换的属性必须存在于目标节点中。
参数“delete”删除目标节点的属性。要删除的属性由子语句标识为“delete”语句。子状态的关键字必须匹配目标节点中相应的关键字，参数的字符串必须等于目标节点中相应的关键字的参数字符串。
deviate 的子语句

substatement	section	cardinality
config	7.21.1	0..1
default	7.6.4, 7.7.4	0..n
mandatory	7.6.5	0..1
max-elements	7.7.6	0..1

```
| min-elements | 7.7.5        | 0..1        |
| must         | 7.5.3        | 0..n        |
| type         | 7.4          | 0..1        |
| unique       | 7.8.3        | 0..n        |
| units        | 7.3.3        | 0..1        |
+--------------+--------------+-------------+
```

7.20.3.3. 使用示例

在这个例子中，服务器通知客户端应用程序它不支持 RFC 867 风格的“daytime”服务。

```
module example-deviations {
  yang-version 1.1;
  namespace "urn:example:deviations";
  prefix md;

  import example-base {
    prefix base;
  }

  deviation /base:system/base:daytime {
    deviate not-supported;
  }
  ...
}
```

一个服务器将通告模板“example-base”和“example-deviations”。
以下示例将特定于服务器的默认值设置为没有定义默认值的叶节点：

```
deviation /base:system/base:user/base:type {
  deviate add {
    default "admin"; // new users are 'admin' by default
  }
}
```

在这个例子中，服务器将名称服务器的数量限制为 3：

```
deviation /base:system/base:name-server {
  deviate replace {
    max-elements 3;
  }
}
```

如果最初的定义是：

```
container system {
  must "daytime or time";
  ...
```

```
}
```
服务器可以通过执行以下操作去除这个“must”约束：
```
deviation /base:system {
  deviate delete {
    must "daytime or time";
  }
}
```

7.21. 通用声明

本节定义了几个其他语句通用的子语句。

7.21.1. “config”声明

“config”语句将字符串“true”或“false”作为参数。 如果“config”是“true”，则定义代表配置。 表示配置的数据节点是配置数据存储的一部分。
如果“config”是“false”，则定义表示状态数据。 表示状态数据的数据节点不是配置数据存储的一部分。
如果未指定“config”，则缺省值与父架构节点的“config”值相同。 如果父节点是一个 case 节点，则该值与 case 节点的父节点选择节点相同。
如果顶层节点没有指定“config”语句，则默认值为“true”。
如果一个节点的“config”设置为“false”，那么它下面的任何节点都不能将“config”设置为“true”。

7.21.2. “status”声明

“status”语句将字符串“current”，“deprecated”或“obsolete”作为参数之一。

- “current”是指定义是当前和有效的。
- “deprecated”表示一个陈旧的定义，但它允许 new/continued 的实现，以促进与 older/existing 的实现的互操作性。
- “obsolete”意味着该定义已经过时，并且不应该被实现 and/or 可以从实现中去除。

如果未指定状态，则默认为“current”。
如果定义为“current”，则不得在同一模块中引用“deprecated”或“obsolete”的定义。
如果定义为“deprecated”，则不得在同一模块中引用“obsolete”定义。
例如，以下是非法的：

```
typedef my-type {
  status deprecated;
  type int32;
}

leaf my-leaf {
```

```
  status current;
  type my-type; // 非法的，因为 my-type 已被弃用
}
```

7.21.3. “description”声明

“description”语句将一个字符串作为参数，该字符串包含对此定义的可读文本描述。 文本以模块开发人员选择的语言（或多种语言）提供; 为了互操作性，建议选择一种在将使用该模块的网络管理员社区中广泛理解的语言。

7.21.4. “reference”声明

“reference”语句将一个字符串作为参数，这个字符串是一个可读的交叉引用的外部文档 - 定义相关管理信息的另一个模块或提供与此定义有关的附加信息的文档。
例如，“uri”数据类型的 typedef 可能如下所示：

```
typedef uri {
  type string;
  reference
    "RFC 3986: Uniform Resource Identifier (URI): Generic Syntax";
  ...
}
```

7.21.5. “when”声明

“when”语句使其父数据定义语句有条件。父数据定义语句定义的节点只有在满足“when”语句指定的条件时才有效。该语句的参数是一个 XPath 表达式（请参阅第 6.4 节），用于正式指定此条件。如果 XPath 表达式在概念上对于特定实例的计算结果为“true”，则由父数据定义语句定义的节点是有效的;否则，它不是。
作为列表键的叶子不能有“when”语句。
如果在列表中使用的分组中定义了键叶，那么“uses”语句不能有“when”语句。
有关更多信息，请参阅第 8.3.2 节。

除了第 6.4.1 节中的定义之外，XPath 表达式在以下上下文中概念性地评估：

- 如果“when”语句是“augment”语句的孩子，那么如果目标节点是数据节点，则上下文节点是数据树中扩充的目标节点。否则，上下文节点是与也是数据节点的目标节点最接近的祖先节点。如果不存在这样的节点，则上下文节点是根节点。在处理 XPath 表达式的过程中，通过删除由“augment”语句添加的节点的所有实例（如果有的话），可访问树暂时被改变。
- 如果“when”语句是“uses”，“choice”或“case”语句的子语，那么上下文节点是与节点最接近的祖先节点，“when”语句也是数据节点。如果不存在这样的节点，则上下文节点是根节点。在处理 XPath 表达式的过程中，通过删除由“uses”，“choice”或“case”语句添加的节点的所有实例（如果有的话），可访问树暂时被改变。
- 如果“when”语句是任何其他数据定义语句的孩子，则在处理 XPath 表达式时，可访问的树会被暂时更改，方法是将所有定义了“when”语句的数据节点的实例

替换为单个具有相同名称的虚拟节点，但没有值和没有子节点。如果不存在这样的实例，则暂时创建虚拟节点。上下文节点是这个虚拟节点。
使用标准的 XPath 规则将 XPath 表达式的结果转换为布尔值。
如果 XPath 表达式引用任何也有关联“when”语句的节点，则必须首先计算这些“when”表达式。在“when”表达式之间不能有循环依赖关系。
请注意，XPath 表达式是在概念上评估的。这意味着实现不必在服务器中使用 XPath 评估程序。 “when”语句可以很好地用专门编写的代码来实现。

8. 约束

8.1. 数据约束

几个 YANG 语句定义了对有效数据的约束。这些约束以不同的方式执行，具体取决于语句定义的数据类型。

- 如果约束是在配置数据上定义的，则它必须在有效的配置数据树中为 true。
- 如果约束是在状态数据上定义的，则它必须在有效的状态数据树中为 true。
- 如果在通知内容上定义了约束条件，则必须在任何通知数据树中都为 true。
- 如果约束是在 RPC 或动作输入参数(action input parameters)上定义的，那么在调用 RPC 或动作操作(action operation)时，它必须为 true。
- 如果在 RPC 或动作输出参数(action output parameters)上定义了约束，则它必须在 RPC 或动作回复(action reply)中为 true。

以下属性在所有数据树中都是正确的：

所有叶子数据值必须匹配叶子的类型约束，包括类型的“range”，“length”和“pattern”属性中定义的那些。

- 所有列表条目必须存在所有关键叶子。
- 在所有选择中，节点必须至多存在一个分支。
- 如果服务器中“if-feature”表达式的计算结果为“false”，那么必须不存在标有“if-feature”的节点。
- 如果数据树中的“when”条件评估为“false”，则必须不存在标有“when”的节点。

以下属性在有效的数据树中是正确的：

- 所有“must”约束必须评估为“true”。
- 必须满足通过“path”语句定义的所有参照完整性约束。
- 列表上的所有“unique”约束必须满足。
- 除非节点或其任何祖先具有“when”条件或“if-feature”表达式评估为“false”，否则强制执行叶子和选择的约束。

- 对于列表和叶列表，强制执行“min-elements”和“max-elements”约束，除非节点或其任何祖先具有“when”条件或“if-feature”表达式，其结果为“false”。
正在运行的配置数据存储必须始终有效。

8.2. 配置数据修改

- 如果请求创建了一个选项(choice)下的配置数据节点，服务器将删除数据树中其他任何分支的现有节点。
- 如果一个请求修改一个配置数据节点，使得任何节点的“when”表达式变为false，那么服务器删除带有“when”表达式的数据树中的节点。

8.3. NETCONF 约束执行模型

对于配置数据，在以下三个执行窗口中必须强制执行:

- 在分析 RPC 有效载荷时
- 在处理<edit-config>操作过程中
- 在验证过程中

以下各节将讨论这些方案中的每一个。

8.3.1. 有效负载分析

当内容到达 RPC 有效载荷时，它必须是格式良好的 XML，遵循由服务器实现的一组模型定义的层次结构和内容规则。

- 如果叶数据值与叶的类型约束不匹配，包括类型的“range”，“length”和“pattern”属性中定义的那些值，服务器必须在<rpc-error>中回复一个“invalid-value”的<error-tag>，以及与约束关联的 error-app-tag（7.5.4.2 节）和错误消息（7.5.4.1 节）。
- 如果列表条目中的所有键都不存在，那么服务器必须在<rpc-error>中回复一个“missing-element”的<error-tag>。
- 如果存在多个选择分支的数据，则服务器必须在<rpc-error>中回复“bad-element”的<error-tag>。
- 如果服务器中存在标记为“if-feature”的节点的数据，并且“if-feature”表达式的计算结果为“false”，则服务器必须在<rpc-error>回复“unknown-element”的<error-tag>。
- 如果标记为“when”的节点的数据存在，并且“when”条件评估为“false”，则服务器必须在<rpc-error>中回复“unknown-element”的<error-tag>。
- 对于插入处理，如果属性“before”和“after”的值对于适当的键叶子的类型无效，则服务器必须在<rpc-error>中回复“bad-attribute”的<error-tag>。
- 如果属性“before”和“after”出现在任何不是“ordered-by”属性为“user”的列表的元素中，则服务器必须在<rpc-error>中回复一个“unknown-attribute”的<error-tag> 。

8.3.2. NETCONF <edit-config>处理

传入数据解析后，NETCONF 服务器通过将数据应用于配置数据存储区执行<edit-config>操作。 在此过程中，必须检测到以下错误：

- 删除不存在数据的请求。
- 为现有数据创建请求。
- 使用“before”或“after”参数插入不存在的请求。
- 标记为“when”的节点的修改请求，以及“when”条件评估为“false”的修改请求。在这种情况下，服务器必须在<rpc-error>中回答“unknown-element”的<error-tag>。

8.3.3. 验证

数据存储处理完成后，最终内容必须遵守所有的验证约束。 根据数据存储区在不同的时间执行验证处理。 如果数据存储“running”或“startup”，则必须在<edit-config>或<copy-config>操作结束时强制执行这些约束。 如果数据存储是“candidate”，那么强制约束被延迟，直到发生<commit>或<validate>操作。

9. 内置类型

YANG 拥有一套类似于许多编程语言的内置类型，但是由于管理信息模型的特殊要求，有一些不同之处。
可以定义从这些内置类型或其他派生类型派生的其他类型。 派生类型可以使用子类型来正式限制可能值的集合。

不同的内置类型及其派生类型允许不同类型的子类型，即字符串的长度和正则表达式限制（第 9.4.4)和 9.4.5 节）和数字类型的范围限制（第 9.2.4 节）。

在 XML 模块中使用某种类型的值的词汇表示，在 YANG 模块中指定默认值和数值范围。

9.1. 规范表示

对于大多数类型，这个类型的值有一个唯一的规范表示。 有些类型允许多个相同值的词汇表示; 例如，可以将正整数“17”表示为“+17”或“17”。 实现必须支持本文档中指定的所有词汇表示。
当服务器发送 XML 编码数据时，它必须使用本节中定义的规范形式。 其他编码可能会引入其他编码。 但是请注意，数据树中的值在概念上以本节中定义的规范表示形式存储。 特别是，如果数据类型具有规范形式，则使用规范形式进行任何 XPath 表达式评估。 如果数据类型没有规范形式，则该值的格式必须与数据类型的词汇表示相匹配，但确切的格式是与实现相关的。
一些类型具有取决于编码的词汇表示，例如它们出现在其中的 XML 上下文。 这些类型没有规范的形式。

9.2. 整数内置类型

整数内置类型是 int8， int16， int32， int64， uint8， uint16， uint32 和 uint64。 它们表示不同大小的有符号和无符号整数：

- int8 表示-128 到 127 之间的整数值。
- int16 表示-32768 和 32767 之间的整数值。
- int32 表示-2147483648 到 2147483647 之间的整数值。
- int64 表示-9223372036854775808 和 9223372036854775807 之间的整数值。
- uint8 表示 0 到 255 之间的整数值。
- uint16 表示 0 和 65535 之间的整数值。
- uint32 表示 0 和 4294967295 之间的整数值。
- uint64 表示 0 和 18446744073709551615 之间的整数值。

9.2.1. 词汇表示

整数值在词汇上表示为可选符号（“+”或“-”），后面跟随一个十进制数字序列。 如果没有指定符号，则假定为“+”。
为方便起见，在为 YANG 模块中的整数指定默认值时，可以使用代表十六进制或八进制表示法中的值的替代词汇表示法。
十六进制符号由可选符号（“+”或“-”）组成，后跟字符“0x”，后跟十六进制数字，其中字母可以是大写或小写。
八进制符号由一个可选符号（“+”或“-”）组成，后跟字符“0”，后面跟着一些八进制数字。
请注意，如果 YANG 模块中的默认值具有前导零（“0”），则将其解释为八进制数。 在 XML 编码中，整数总是被解释为十进制数，并且允许前导零。
例子：

```
// 合法值
+4711                          // 合法正数值
4711                           // 合法正数值
```

```
-123                          // 合法负数值
0xf00f                        // 合法的十六进制值
-0xf                          // 合法的十六进制值
052                           // 合法的八进制值

// 非法值
- 1                           // 中间空格非法
```

9.2.2. 规范形式

正整数的规范形式不包含符号“+”。 前导零是被禁止的。 零值表示为“0”。

9.2.3. 限制

所有整数类型都可以用“range”语句限制（见第 9.2.4 节）。

9.2.4. “range”声明

“range”语句是“type”语句的可选子语句，它将范围表达式字符串作为参数。它用于限制整型和十进制内置类型，或从它们派生的类型。
一个范围由一个显式值或者一个包含下界的连续点组成，两个连续的点“..”和一个上限界限。可以给出多个值或范围，用“|”分隔。如果给出了多个值或范围，它们都必须是不相交的，并且必须按照升序排列。如果范围限制应用于已经限制范围的类型，那么新的限制必须是等同限制或更多限制，即提高下限，减少上限，删除显式值或范围或将范围拆分为多个范围与中间差距。在范围表达式中给出的每个显式值和范围边界值必须与被限制的类型匹配，或者是特殊值“min”或“max”之一。 “min”和“max”分别表示被限制类型接受的最小值和最大值。
范围表达式语法由第 14 节中的“range-arg”规则正式定义。

9.2.4.1. range 子语句

```
+---------------+---------+-------------+
| substatement  | section | cardinality |
+---------------+---------+-------------+
| description   | 7.21.3  | 0..1        |
| error-app-tag | 7.5.4.2 | 0..1        |
| error-message | 7.5.4.1 | 0..1        |
| reference     | 7.21.4  | 0..1        |
+---------------+---------+-------------+
```

9.2.5. 使用示例

```
typedef my-base-int32-type {
```

```
  type int32 {
    range "1..4 | 10..20";
  }
}

typedef my-type1 {
  type my-base-int32-type {
    // legal range restriction
    range "11..max"; // 11..20
  }
}

typedef my-type2 {
  type my-base-int32-type {
    // illegal range restriction
    range "11..100";
  }
}
```

9.3. `decimal64` 内置类型

`decimal64` 内置类型表示实数的子集，可以用十进制数字表示。 `decimal64` 的值空间是可以通过将 `64` 位有符号整数乘以十的负的幂来获得的数字的集合，即可表示为“`i x 10^-n`”，其中 `i` 是整数 `64` 并且 `n` 是整数 `1` 至 `18` 之间。

9.3.1. 词汇表示

一个 `decimal64` 值从词法上表示为一个可选的符号（“`+`”或“`-`”），后跟一串十进制数字，可选地后跟一个句号（'`.`'）作为十进制指示符和一串十进制数字。 如果没有指定符号，则假定为“`+`”。

9.3.2. 规范形式

正数 `decimal64` 值的规范形式不包含符号“`+`”。 小数点是必需的。 前导和尾随零是禁止的，但必须遵守小数点前后至少有一位的规则。 零值表示为“`0.0`”。

9.3.3. 限制

可以用“`range`”语句来限制 `decimal64` 类型（见第 9.2.4 节）。

9.3.4. “`fraction-digits`”声明

如果类型是“decimal64”，则必须存在“fraction-digits”语句，它是“type”语句的子语句。 它包含 1 到 18 之间的整数作为参数。 它通过将值空间限制为可表达为“i x 10^-n”的数字来控制 decimal64 类型的值之间的最小差异的大小，其中 n 是 fraction-digits 参数。
下表列出了每个 fraction-digit 值的最小值和最大值：

fraction-digit	min	max
1	-922337203685477580.8	922337203685477580.7
2	-92233720368547758.08	92233720368547758.07
3	-9223372036854775.808	9223372036854775.807
4	-922337203685477.5808	922337203685477.5807
5	-92233720368547.75808	92233720368547.75807
6	-9223372036854.775808	9223372036854.775807
7	-922337203685.4775808	922337203685.4775807
8	-92233720368.54775808	92233720368.54775807
9	-9223372036.854775808	9223372036.854775807
10	-922337203.6854775808	922337203.6854775807
11	-92233720.36854775808	92233720.36854775807
12	-9223372.036854775808	9223372.036854775807
13	-922337.2036854775808	922337.2036854775807
14	-92233.72036854775808	92233.72036854775807
15	-9223.372036854775808	9223.372036854775807
16	-922.3372036854775808	922.3372036854775807
17	-92.23372036854775808	92.23372036854775807
18	-9.223372036854775808	9.223372036854775807

9.3.5. 使用示例

```
typedef my-decimal {
  type decimal64 {
    fraction-digits 2;
    range "1 .. 3.14 | 10 | 20..max";
  }
}
```

9.4. 字符串内置类型

字符串内置类型表示 YANG 中可读的字符串。 合法字符是 Unicode 和 ISO/IEC 10646 [ISO.10646]字符，包括制表符，回车符和换行符，但不包括其他 C0 控制字符，代理块和非字符。 字符串语法由第 14 节中的规则“yang-string”正式定义。

9.4.1. 词汇表示

在 XML 编码中，字符串值在词法上表示为字符数据。

9.4.2. 规范形式

规范形式与词汇表示相同。 字符串值的 Unicode 规范化没有执行。

9.4.3. 限制

一个字符串可以用“length”（9.4.4 节）和“pattern”（9.4.5 节）语句进行限制。

9.4.4. “length”声明

“length”语句是“type”语句的一个可选子语句，它将长度表达式字符串作为参数。它用于限制内置类型“string”和“binary”或从它们派生的类型。
“length”语句限制字符串中的 Unicode 字符数。
长度范围由一个显式值或一个下界，两个连续的点“..”和一个上界组成。可以给出多个值或范围，用“|”分隔。长度限制值不能是负数。如果给出了多个值或范围，它们都必须是不相交的，并且必须按照升序排列。如果长度限制应用于已经受长度限制的类型，则新的限制必须是等同限制或更多限制的，即，提高下限，减少上限，移除显式长度值或范围或将范围拆分成具有中间间隙的多个范围。长度值是一个非负整数或特殊值“min”或“max”中的一个。
“min”和“max”分别表示被限制类型可接受的最小和最大长度。实现不需要支持大于 18446744073709551615 的长度值。
长度表达式语法在第 14 节中由规则“length-arg”正式定义。

9.4.4.1. length 的子语句

substatement	section	cardinality
description	7.21.3	0..1
error-app-tag	7.5.4.2	0..1
error-message	7.5.4.1	0..1
reference	7.21.4	0..1

9.4.5. “pattern”声明

“pattern”语句是“type”语句的一个可选子语句，它使用[XSD-TYPES]中定义的正则表达式字符串作为参数。 它用于将内置类型“string”或从“string”派生的类型限制为与该模式匹配的值。

如果类型具有多个“pattern”语句，则表达式将被组合在一起。即，所有这些表达式必须匹配。
如果将模式限制应用于已受模式限制的类型，则除了新模式之外，值必须与基本类型中的所有模式匹配。

9.4.5.1. pattern 的子语句

```
+---------------+---------+-------------+
| substatement  | section | cardinality |
+---------------+---------+-------------+
| description   | 7.21.3  | 0..1        |
| error-app-tag | 7.5.4.2 | 0..1        |
| error-message | 7.5.4.1 | 0..1        |
| modifier      | 9.4.6   | 0..1        |
| reference     | 7.21.4  | 0..1        |
+---------------+---------+-------------+
```

9.4.6. “modifier”声明

“modifier”语句是“pattern”语句的一个可选子语句，它以字符串“invert-match”作为参数。
如果一个模式存在“反向匹配(invert-match)”修饰符，则类型被限制为与模式不匹配的值。

9.4.7. 使用示例

用下面的 typedef：

```
typedef my-base-str-type {
  type string {
    length "1..255";
  }
}
```

以下改进是合法的：

```
type my-base-str-type {
  // legal length refinement
  length "11 | 42..max"; // 11 | 42..255
}
```

而下面的改进是非法的：

```
type my-base-str-type {
  // illegal length refinement
  length "1..999";
}
```

使用以下类型：

```
type string {
  length "0..4";
  pattern "[0-9a-fA-F]*";
}
```
以下字符串匹配：

```
AB          // legal
9A00        // legal
```
和以下字符串不匹配：

```
00ABAB      // illegal, too long
xx00        // illegal, bad characters
```
使用以下类型：

```
type string {
  length "1..max";
  pattern '[a-zA-Z_][a-zA-Z0-9\-_.]*';
  pattern '[xX][mM][lL].*' {
    modifier invert-match;
  }
}
```
以下字符串匹配：

```
enabled     // legal
```
和以下字符串不匹配：

```
10-mbit     // illegal, starts with a number
xml-element // illegal, starts with illegal sequence
```

9.5. 布尔内置类型

布尔内置类型表示一个布尔值。

9.5.1. 词汇表示

一个布尔值的词汇表示是一个值为“`true`”或“`false`”的字符串。 这些值必须是小写的。

9.5.2. 规范形式

规范形式与词汇表示相同。

9.5.3. 限制

一个布尔值不能被限制。

9.6. 枚举内置类型

枚举内置类型表示来自一组分配名称的值。

9.6.1. 词汇表示

枚举值的词汇表示是指定的名称字符串。

9.6.2. 规范形式

规范形式是分配的名称字符串。

9.6.3. 限制

一个枚举可以用一个或多个“enum”（9.6.4 节）语句来限制，这个枚举枚举了基类型的值的一个子集。

9.6.4. "enum"声明

如果类型是“enum”，则必须存在“enum”语句，它是“type”语句的子语句。 它被重复用于指定枚举类型的每个指定名称。 它将作为参数的字符串作为分配的名称。 字符串不能是零长度的，也不能有任何前导或尾随的空白字符（任何具有“White_Space”属性的 Unicode 字符）。 应该避免使用 Unicode 控制代码。该语句后面紧跟着一个包含详细枚举信息的子语句块。

枚举中所有分配的名字必须是唯一的。

当一个现存的枚举类型受到限制时，新类型中的赋值名称集必须是基类型的分配名称集的一个子集。 这个分配的名称的值不能被改变。

9.6.4.1. enum 的子语句

```
+--------------+---------+-------------+
| substatement | section | cardinality |
+--------------+---------+-------------+
| description  | 7.21.3  | 0..1        |
| if-feature   | 7.20.2  | 0..n        |
```

```
| reference    | 7.21.4  | 0..1        |
| status       | 7.21.2  | 0..1        |
| value        | 9.6.4.2 | 0..1        |
+--------------+---------+-------------+
```

9.6.4.2. “value”声明

“value”语句是可选的，用于将整数值与枚举的分配名称相关联。这个整数值必须在-2147483648到2147483647的范围内，它必须在枚举类型中是唯一的。如果没有指定值，则会自动分配一个值。如果“enum”子语句是第一个被定义的，赋值为零（0）;否则，所分配的值比当前最高枚举值（即，在父类型“语句”中当前“enum”子语句之前的隐式或显式最高枚举值）大1。
请注意，“enum”语句中存在“if-feature”语句不会影响自动分配的值。
如果当前最高值等于2147483647，那么必须为当前最高值之后的“enum”子语句指定一个枚举值。
当现有的枚举类型受到限制时，“value”语句必须具有与基类型相同的值或不存在，在这种情况下，该值与基类型中的值相同。

9.6.5 使用示例

```
leaf myenum {
  type enumeration {
    enum zero;
    enum one;
    enum seven {
      value 7;
    }
  }
}
```

叶子“myenum”的值为“seven”的词汇表示是：

```
<myenum>seven</myenum>
```

用下面的typedef：

```
typedef my-base-enumeration-type {
  type enumeration {
    enum white {
      value 1;
    }
    enum yellow {
      value 2;
    }
    enum red {
      value 3;
    }
  }
}
```

以下改进是合法的：

```
type my-base-enumeration-type {
  // legal enum refinement
  enum yellow;
  enum red {
    value 3;
  }
}
```

而下面的改进是非法的：

```
type my-base-enumeration-type {
  // illegal enum refinement
  enum yellow {
    value 4; // illegal value change
  }
  enum black; // illegal addition of new name
}
```

以下示例显示了如何使用“`if-feature`”标记“`enum`”，这使得该值只在具有相应功能的服务器上合法：

```
type enumeration {
  enum tcp;
  enum ssh {
    if-feature ssh;
  }
  enum tls {
    if-feature tls;
  }
}
```

9.7. 位(bits)内置类型

位(`bits`)内置类型表示一个位集。也就是说，一个比特值是由从 `0` 开始的小整数位置号标识的一组标志。每个比特号都有一个分配的名称。

当一个现有的位类型受到限制时，新类型中的一组指定名称必须是基类型的指定名称集合的一个子集。这个指定名称的位位置不能被改变。

9.7.1. 限制

位类型可以用“`bits`”（9.7.4 节）语句来限制。

9.7.2. 词汇表示

位类型的词汇表示是所设置的位的名称的空格分隔列表。 因此零长度的字符串表示没有位被设置的值。

9.7.3. 规范形式

在规范形式中，位值由一个单独的空格字符分隔，并按其位置排序（见第 9.7.4.2 节）。

9.7.4. “bits”声明

如果类型是“bits”，则必须存在“bit”语句，它是“type”语句的子语句。 它被重复用于指定每个分配的位类型的命名位。 它以一个字符串作为参数，该字符串是该位的分配名称。 紧接着是一个包含详细比特信息的子语句块。 分配的名称遵循与标识符相同的语法规则（请参见第 6.2 节）。
所有以位类型分配的名称必须是唯一的。

9.7.4.1. bit 子语句

substatement	section	cardinality
description	7.21.3	0..1
if-feature	7.20.2	0..n
position	9.7.4.2	0..1
reference	7.21.4	0..1
status	7.21.2	0..1

9.7.4.2. “position”声明

“position”语句是可选的，它将一个非负整数值作为参数，该值指定位在假设位域内的位置。位置值必须在 0 到 4294967295 之间，并且在位类型中它必须是唯一的。
如果没有指定位的位置，则会自动分配一个位。如果“bit”子语句是第一个被定义的，则赋值为零（0）。否则，所分配的值比当前最高位位置（即，在父“type”语句中的当前“bit”子语句之前的隐式或显式最高位位置）大 1。
请注意，在“bit”语句中存在“if-feature”语句不会影响自动分配的位置。
如果当前最高位的位置值等于 4294967295，则必须为当前最高位置值之后的“bit”子位置指定一个位置值。
当一个现有的位类型受到限制时，“position”语句必须具有与基类型相同的值或不存在，在这种情况下，该值与基类型中的值相同。

9.7.5. 使用示例

鉴于以下 typedef 和叶：

```
typedef mybits-type {
  type bits {
    bit disable-nagle {
      position 0;
    }
    bit auto-sense-speed {
      position 1;
    }
    bit ten-mb-only {
      position 2;
    }
  }
}

leaf mybits {
  type mybits-type;
  default "auto-sense-speed";
}
```

这个叶子的词法表示的值为 `disable-nagle` 和 `10-mb-only` 集合将是：

```
<mybits>disable-nagle ten-mb-only</mybits>
```

下面的例子显示了这种类型的合法改进：

```
type mybits-type {
  // legal bit refinement
  bit disable-nagle {
    position 0;
  }
  bit auto-sense-speed {
    position 1;
  }
}
```

而下面的改进是非法的：

```
type mybits-type {
  // illegal bit refinement
  bit disable-nagle {
    position 2; // illegal position change
  }
  bit hundred-mb-only; // illegal addition of new name
}
```

9.8. 二进制内置类型

二进制内建类型表示任何二进制数据，即八位字节序列。

9.8.1. 限制

二进制类型可以用“length”（9.4.4 节）语句来限制。 二进制值的长度是它包含的八位字节数。

9.8.2. 词汇表示

二进制值使用 base64 编码方案进行编码（请参见[RFC4648]中的第 4 节）。

9.8.3. 规范形式

二进制值的规范形式遵循[RFC4648]中的“Base 64 Encoding”的规则。

9.9. leafref 内置类型

leafref 内置类型被限制在模式树中的某个叶子或叶子列表节点的值空间中，并且可选地被数据树中相应的实例节点进一步限制。 “path”子状态（第 9.9.2 节）用于标识模式树中引用的叶节点或叶节点列表节点。 引用节点的值空间是引用节点的值空间。
如果“require-instance”属性（第 9.9.3 节）为“true”，那么数据树中必须存在节点，或者使用默认值的节点（见 7.6.1 和 7.7.2 节）， 所引用的模式树叶或叶列表节点的值与有效数据树中的 leafref 值相同。
如果引用节点代表配置数据，并且“require-instance”属性（第 9.9.3 节）为“true”，则引用节点也必须表示配置。
不得有任何圆形的树叶链。

如果 leafref 引用的叶是基于一个或多个特征的条件（参见 7.20.2 节），那么带有 leafref type 的叶也必须是基于至少相同的一组特征的条件。

9.9.1. 限制

leafref 可以用“require-instance”语句来限制（第 9.9.3 节）。

9.9.2. “path”声明

如果类型是“leafref”，则必须存在“path”语句，它是“type”语句的子语句。它需要一个字符串作为参数，它必须引用一个叶或叶子列表节点。
路径参数的语法是 XPath 缩写语法的子集。谓词仅用于约束列表条目的关键节点的值。每个谓词包含每个关键字只有一个相等性测试，并且如果列表具有

多个关键字，则多个相邻谓词可以存在。该语法由第 14 节中的规则“path-arg”正式定义。
只有在需要多个关键字引用来唯一标识一个叶子实例时，谓词才被使用。如果列表中有多个键或者需要除列表中的键以外的叶子的引用，则会发生这种情况。在这些情况下，通常会指定多个 leafrefs，并使用谓词将它们绑定在一起。
“path”表达式评估为由零个，一个或多个节点组成的节点集。如果“require-instance”属性为“true”，那么这个节点集必须是非空的。
除了第 6.4.1 节中的定义之外，“path”XPath 表达式在以下上下文中概念性地评估：

- 如果在“typedef”中定义了“path”语句，则上下文节点是数据树中引用 typedef 的叶节点或叶节点列表节点。
- 否则，上下文节点是定义了“path”语句的数据树中的节点。

9.9.3. “require-instance”声明

如果类型是“instance-identifier”或“leafref”，那么“require-instance”语句可以是“type”语句的子语句。 它将字符串“true”或“false”作为参数。 如果此语句不存在，则默认为“true”。
如果“require-instance”为“true”，则意味着被引用的实例必须存在以使数据有效。 这个约束是根据第 8 节中的规则执行的。
如果“require-instance”为“false”，则意味着被引用的实例可能存在于有效数据中。

9.9.4. 词汇表示

leafref 值在词法上与其引用的叶代表其值的方式相同。

9.9.5. 规范形式

leafref 的规范形式与它引用的叶子的规范形式相同。

9.9.6. 使用示例

用下面的列表：

```
list interface {
  key "name";
  leaf name {
    type string;
  }
  leaf admin-status {
    type admin-status;
  }
```

```
  list address {
    key "ip";
    leaf ip {
      type yang:ip-address;
    }
  }
}
```

以下 `leafref` 引用了一个现有的接口：

```
leaf mgmt-interface {
  type leafref {
    path "../interface/name";
  }
}
```

相应 `XML` 片段的示例：

```
<interface>
  <name>eth0</name>
</interface>
<interface>
  <name>lo</name>
</interface>

<mgmt-interface>eth0</mgmt-interface>
```

以下 `leafrefs` 指的是接口的现有地址：

```
container default-address {
  leaf ifname {
    type leafref {
      path "../../interface/name";
    }
  }
  leaf address {
    type leafref {
      path "../../interface[name = current()/../ifname]"
         + "/address/ip";
    }
  }
}
```

相应 `XML` 片段的示例：

```
<interface>
  <name>eth0</name>
  <admin-status>up</admin-status>
  <address>
    <ip>192.0.2.1</ip>
  </address>
  <address>
```

```
    <ip>192.0.2.2</ip>
  </address>
</interface>
<interface>
  <name>lo</name>
  <admin-status>up</admin-status>
  <address>
    <ip>127.0.0.1</ip>
  </address>
</interface>

<default-address>
  <ifname>eth0</ifname>
  <address>192.0.2.2</address>
</default-address>
```

下面的列表使用其中一个键的 `leafref`。 这与关系数据库中的外键类似。

```
list packet-filter {
  key "if-name filter-id";
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf filter-id {
    type uint32;
  }
  ...
}
```

相应 XML 片段的示例：

```
<interface>
  <name>eth0</name>
  <admin-status>up</admin-status>
  <address>
    <ip>192.0.2.1</ip>
  </address>
  <address>
    <ip>192.0.2.2</ip>
  </address>
</interface>

<packet-filter>
  <if-name>eth0</if-name>
  <filter-id>1</filter-id>
  ...
```

```
</packet-filter>
<packet-filter>
  <if-name>eth0</if-name>
  <filter-id>2</filter-id>
  ...
</packet-filter>
```

以下通知定义了两个引用到现有管理状态的引用：

```
notification link-failure {
  leaf if-name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf admin-status {
    type leafref {
      path "/interface[name = current()/../if-name]"
         + "/admin-status";
    }
  }
}
```

一个相应的 XML 通知的例子：

```
<notification
  xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
  <eventTime>2008-04-01T00:01:00Z</eventTime>
  <link-failure xmlns="urn:example:system">
    <if-name>eth0</if-name>
    <admin-status>up</admin-status>
  </link-failure>
</notification>
```

9.10. identityref 内置类型

identityref 内置类型用于引用现有标识（请参阅第 7.18 节）。

9.10.1. 限制

identityref 不能被限制。

9.10.2. identityref 的“base”声明

如果类型是“identityref”，则必须至少存在一次“base”语句，即“type”语句的子语句。 参数是由“identityref”声明定义的身份的名称。 如果标识名称上存在

前缀，则它指的是使用该前缀导入的模块中定义的标识。 否则，必须在当前模块或包含的子模块中定义具有匹配名称的标识。
identityref 的有效值是从所有 identityref 的基本身份派生的任何身份。 在特定的服务器上，有效值进一步限制在由服务器实现的模块中定义的一组身份。

9.10.3. 词汇表示

identityref 在词法上表示为[XML-NAMES]中定义的被引用的标识的限定名称。如果前缀不存在，则 identityref 的名称空间是对包含 identityref 值的元素有效的默认名称空间。
当使用“default”语句给 identityref 赋予默认值时，默认值中的标识名可以有一个前缀。如果身份名称中存在前缀，则表示在该模块中使用该前缀导入的身份，或者在当前模块或其子模块中定义了身份的前提下，使用当前模块的前缀。否则，必须在当前模块或其子模块中定义具有匹配名称的标识。
“must”或“when”XPath 表达式中“identityref”类型的节点的字符串值是带有前缀的引用标识的限定名。如果引用的标识是在导入的模块中定义的，则字符串值中的前缀是在相应的“import”语句中定义的前缀。否则，字符串值中的前缀是当前模块的前缀。

9.10.4. 规范形式

由于词法形式依赖于值出现的 XML 上下文，所以这种类型不具有规范形式。

9.10.5. 使用示例

使用 7.18.3 节中的身份定义和以下模块：

```
module example-my-crypto {
  yang-version 1.1;
  namespace "urn:example:my-crypto";
  prefix mc;

  import "example-crypto-base" {
    prefix "crypto";
  }

  identity aes {
    base "crypto:crypto-alg";
  }

  leaf crypto {
    type identityref {
```

```
      base "crypto:crypto-alg";
    }
  }

  container aes-parameters {
    when "../crypto = 'mc:aes'";
    ...
  }
}
```

以下是如何将叶片“crypto”编码的例子，如果该值是在“des”模块中定义的“des3”身份：

```
<crypto xmlns:des="urn:example:des">des:des3</crypto>
```

编码中使用的任何前缀对于每个实例编码都是本地的。 这意味着相同的identityref可以通过不同的实现进行不同的编码。 例如，下面的例子就像上面一样编码相同的叶子：

```
<crypto xmlns:x="urn:example:des">x:des3</crypto>
```

如果在“example-my-crypto”模块中定义的“crypto”叶的值是“aes”，则它可以被编码为：

```
<crypto xmlns:mc="urn:example:my-crypto">mc:aes</crypto>
```

或者，使用默认名称空间：

```
<crypto>aes</crypto>
```

9.11. 空的内置类型

空的内置类型代表一个没有任何价值的叶子；它通过它的存在或缺席传达信息。

一个空的类型不能有一个默认值。

9.11.1.限制

空的类型不能被限制。

9.11.2. 词汇表示

不适用。

9.11.3. 规范形式

不适用。

9.11.4. 使用示例

随着叶子：

```
leaf enable-qos {
  type empty;
}
```

如果叶子存在，以下是有效编码的示例：

```
<enable-qos/>
```

9.12. 联合内置类型

联合内置类型表示一个对应于其成员类型之一的值。

当类型是“union”时，“type”语句（7.4 节）必须存在。 它重复用于指定联合的每个成员类型。 它以一个字符串作为参数，它是一个成员类型的名称。
成员类型可以是任何内置类型或派生类型。

在生成 XML 编码时，根据值所属成员类型的规则对值进行编码。 在解释 XML 编码时，将按照每个成员类型（在“type”语句中指定的顺序）对值进行连续验证，直到找到匹配项为止。 匹配的类型将是被验证的节点的值的类型，并根据该类型的规则解释编码。
成员类型中定义的任何默认值或“units”属性均不由联合类型继承。

9.12.1. 限制

工会不能被限制。 但是，根据第 9 节中定义的规则，每个成员类型都可以被限制。

9.12.2. 词汇表示

联合的词汇表示是与任何一个成员类型的表示相对应的值。

9.12.3. 规范形式

联合值的规范形式与值的成员类型的规范形式相同。

9.12.4. 使用示例

以下是 int32 和枚举的联合：

```
type union {
  type int32;
  type enumeration {
    enum "unbounded";
  }
}
```

当成员类型是“require-instance”属性（第 9.9.3 节）为“true”的 leafref 时，必须小心。 如果这种类型的叶片引用了现有的实例，则如果目标实例被删除，叶片的值必须重新确认。 例如，具有以下定义：

```
list filter {
  key name;
  leaf name {
    type string;
  }
  ...
}

leaf outbound-filter {
  type union {
    type leafref {
      path "/filter/name";
    }
    type enumeration {
      enum default-filter;
    }
  }
}
```

假设筛选器列表中存在名称为“http”的条目，并且出站筛选器叶具有以下值：

```
<filter>
  <name>http</name>
</filter>
<outbound-filter>http</outbound-filter>
```

如果删除了筛选器条目“http”，则出站筛选器页面的值与 leafref 不匹配，并检查下一个成员类型。 当前值（“http”）与枚举不匹配，所以生成的配置无效。

如果联合中的第二个成员类型的类型是“string”而不是枚举类型，则当前值将会匹配，并且生成的配置将是有效的。

9.13. 实例标识符内置类型

实例标识内置类型用于唯一标识数据树中的特定实例节点。

实例标识符的语法是 XPath 缩写语法的子集，由第 14 节中的“instance-identifier”规则正式定义。它用于唯一标识数据树中的节点。谓词仅用于指定列表项的关键节点的值，叶列表项的值或不包含键的列表的位置索引。为了用键标识列表条目，每个谓词由每个键的一个相等性测试组成，每个键必须有一个相应的谓词。如果某个键的类型为“empty”，则表示为零长度字符串（“”）。
如果具有实例标识符类型的叶代表配置数据，并且“require-instance”属性（第 9.9.3 节）为“true”，则它引用的节点也必须表示配置。这样的一个页面对有效数据进行约束。所有这样的叶子节点必须引用现有的节点或叶节点或叶节点节点，使用它们的默认值（见 7.6.1 节和 7.7.2 节）以使数据有效。这个约束是根据第 8 节中的规则执行的。
除了第 6.4.1 节中的定义之外，“instance-identifier”XPath 表达式在以下上下文中概念性地被评估：

- 上下文节点是可访问树中的根节点。

9.13.1. 限制

实例标识符可以用“require-instance”语句来限制（见第 9.9.3 节）。

9.13.2. 词汇表示

实例标识符值从词法上表示为一个字符串。 实例标识符值中的所有节点名必须使用明确的名称空间前缀限定，这些前缀必须在实例标识符的 XML 元素中的 XML 名称空间范围内声明。
编码中使用的任何前缀对于每个实例编码都是本地的。 这意味着相同的实例标识符可以通过不同的实现被不同地编码。

9.13.3. 规范形式

由于词法形式依赖于值出现的 XML 上下文，所以这种类型不具有规范形式。

9.13.4. 使用示例

以下是实例标识符的示例：

```
/* instance-identifier for a container */
/ex:system/ex:services/ex:ssh

/* instance-identifier for a leaf */
/ex:system/ex:services/ex:ssh/ex:port
```

```
/* instance-identifier for a list entry */
/ex:system/ex:user[ex:name='fred']

/* instance-identifier for a leaf in a list entry */
/ex:system/ex:user[ex:name='fred']/ex:type

/* instance-identifier for a list entry with two keys */
/ex:system/ex:server[ex:ip='192.0.2.1'][ex:port='80']

/* instance-identifier for a list entry where the second
key ("enabled") is of type "empty" */
/ex:system/ex:service[ex:name='foo'][ex:enabled='']

/* instance-identifier for a leaf-list entry */
/ex:system/ex:services/ex:ssh/ex:cipher[.='blowfish-cbc']

/* instance-identifier for a list entry without keys */
/ex:stats/ex:port[3]
```

10. XPath 函数

本文档定义了两个通用的 `XPath` 函数和五个 YANG 类型特定的 XPath 函数。函数签名用[XPATH]中使用的语法指定。

10.1. 节点集函数

10.1.1. current()

```
node-set current()
```

`current()`函数不接受任何输入参数，并返回以初始上下文节点为唯一成员的节点集。

10.1.2. 使用示例

现在有一个列表

```
list interface {
  key "name";
  ...
  leaf enabled {
    type boolean;
  }
```

```
  ...
}
```
以下叶定义了一个“must”表达式，确保引用的接口被启用：
```
leaf outgoing-interface {
  type leafref {
    path "/interface/name";
  }
  must '/interface[name=current()]/enabled = "true"';
}
```

10.2. 字符串函数

10.2.1. re-match()

```
boolean re-match(string subject, string pattern)
```
如果“subject”字符串匹配正则表达式“pattern”，则 re-match()函数返回“true”。 否则，返回“false”。
re-match()函数检查字符串是否与给定的正则表达式匹配。 使用的正则表达式是 XML 模式正则表达式[XSD-TYPES]。 请注意，这包括正则表达式在头部和尾部的隐式锚定。

10.2.1.1. 使用示例

表达式如下：

```
re-match("1.22.333", "\d{1,3}\.\d{1,3}\.\d{1,3}")
```
返回“true”。
要计算所有名为 eth0。<number>的逻辑接口，请执行以下操作：
```
count(/interface[re-match(name, "eth0\.\d+")])
```

10.3. YANG 类型“leafref”和“instance-identifier"”的函数

10.3.1. deref()

```
node-set deref(node-set nodes)
```
deref()函数遵循第一个节点在参数“nodes”中以文档顺序定义的引用，并返回它引用的节点。
如果第一个参数节点的类型是“instance-identifier”，则该函数返回一个包含实例标识符引用的单个节点（如果存在）的节点集。 如果不存在这样的节点，则返回空的节点集合。

如果第一个参数节点的类型为“leafref”，则该函数将返回一个包含 leafref 引用的节点的节点集。 具体来说，这个集合包含由 leafref 的“path”语句（第 9.9.2 节）选择的节点，它们与第一个参数节点具有相同的值。
如果第一个参数节点是任何其他类型，则返回空节点集。

10.3.1.1. 使用示例

```
list interface {
  key "name type";
  leaf name { ... }
  leaf type { ... }
  leaf enabled {
    type boolean;
  }
  ...
}

container mgmt-interface {
  leaf name {
    type leafref {
      path "/interface/name";
    }
  }
  leaf type {
    type leafref {
      path "/interface[name=current()/../name]/type";
    }
    must 'deref(.)/../enabled = "true"' {
      error-message
        "The management interface cannot be disabled.";
    }
  }
}
```

10.4. YANG 类型的“identityref”函数

10.4.1. derived-from()

```
boolean derived-from(node-set nodes, string identity)
```

如果参数“nodes”中的任何节点是“identityref”类型的节点，并且其值是从（参见第 7.18.2 节）派生的标识，则 derived-from()函数返回“true”；否则，返回“false”。
参数“identity”是与第 14 节中的“identifier-ref”规则相匹配的字符串。如果前缀存在于标识中，则它指的是在导入了该前缀的模块中定义的标识，如果

前缀与本地模块的前缀匹配，那么该标识将在该模块中定义。 如果不存在前缀，则标识是指在当前模块或包含的子模块中定义的标识。

10.4.1.1. 使用示例

```
module example-interface {
  yang-version 1.1;

  ...
  identity interface-type;

  identity ethernet {
    base interface-type;
  }

  identity fast-ethernet {
    base ethernet;
  }

  identity gigabit-ethernet {
    base ethernet;
  }

  list interface {
    key name;
    ...
    leaf type {
      type identityref {
        base interface-type;
      }
    }
    ...
  }

  augment "/interface" {
    when 'derived-from(type, "exif:ethernet")';
    // generic Ethernet definitions here
  }
  ...
}
```

10.4.2. `derived-from-or-self()`

```
boolean derived-from-or-self(node-set nodes, string identity)
```

如果参数“nodes”中的任何节点是“identityref”类型的节点，并且其值是等于或来源于其的标识，则 derived-from-or-self()函数将返回“true”（请参阅第 7.18.2 节）身份“identity”；否则，返回“false”。
参数“identity”是与第 14 节中的“identifier-ref”规则相匹配的字符串。如果前缀存在于标识中，则它指的是在导入了该前缀的模块中定义的标识，如果前缀与本地模块的前缀匹配，那么该标识将在该模块中定义。 如果不存在前缀，则标识是指在当前模块或包含的子模块中定义的标识。

使用示例

第 10.4.1.1 节定义的模块也可能有：

```
augment "/interface" {
  when 'derived-from-or-self(type, "exif:fast-ethernet");
  // Fast-Ethernet-specific definitions here
}
```

10.5. YANG 类型“enumeration”函数

10.5.1. enum-value()

```
number enum-value(node-set nodes)
```

enum-value()函数检查参数“nodes”中文档顺序中的第一个节点是否为“enumeration”类型的节点，并返回枚举的整数值。 如果“nodes”节点集为空，或者“nodes”中的第一个节点不是“enumeration”类型，则返回 NaN（不是数字）。

10.5.1.1. 使用示例

有了这个数据模型：

```
list alarm {
  ...
  leaf severity {
    type enumeration {
      enum cleared {
        value 1;
      }
      enum indeterminate {
        value 2;
      }
      enum minor {
        value 3;
      }
      enum warning {
```

```
        value 4;
      }
      enum major {
        value 5;
      }
      enum critical {
        value 6;
      }
    }
  }
}
```

以下 XPath 表达式仅选择严重性为“major”或更高的警报：

```
/alarm[enum-value(severity) >= 5]
```

10.6. YANG 类型的“bits”函数

10.6.1. bit-is-set()

```
boolean bit-is-set(node-set nodes, string bit-name)
```

如果参数“nodes”中的文档顺序中的第一个节点是“bits”类型的节点，并且其值设置了“bit-name”位，则 bit-is-set()函数返回“true” 否则，返回“false”。

10.6.1.1. 使用示例

如果一个接口有这个叶子：

```
leaf flags {
  type bits {
    bit UP;
    bit PROMISCUOUS
    bit DISABLED;
  }
 }
```

可以使用以下 XPath 表达式来选择带有 UP 标志的所有接口：

```
/interface[bit-is-set(flags, "UP")]
```

11. 更新模块

随着模块开始使用，可能需要修改该模块。但是，如果已发布的模块发生更改，如果它们有可能导致使用原始规范的客户端与使用更新的规范的服务器之间的互操作性问题，则不允许进行更改。

对于任何已发布的变更，新的“revision”声明（第 7.1.9 节）必须包含在现有“revision”声明的前面。如果没有现有的“revision”语句，则必须添加一个以确

定新的修订版本。此外，任何必要的变更必须适用于任何元数据声明，包括“organization”和“contact”声明（7.1.7 和 7.1.8 节）。
请注意，模块中包含的定义可供任何其他模块导入，并通过模块名称在“import”语句中引用。因此，模块名称不能被改变。而且，“namespace”语句不能被改变，因为所有的 XML 元素都被命名空间限定了。
过时的定义不能从发布的模块中移除，因为它们的标识符可能仍然被其他模块引用。

已发布模块中的定义可以通过以下任何方式进行修改：

- “enumeration”类型可能会添加新的枚举，前提是旧枚举的值不会更改。请注意，在现有枚举之前插入一个新枚举或重新排序现有枚举将导致现有枚举的新值，除非它们具有分配给它们的显式值。

- “bits”类型可能会添加新的位，前提是旧位的位置不变。请注意，在现有位之前插入新位或重新排序现有位将导致现有位的新位置，除非它们具有分配给它们的明确位置。
- “range”，“length”或“pattern”语句可扩展允许的值空间。
- 可以将“default”语句添加到没有默认值（直接或间接通过类型）的叶。
- 可以添加“units”声明。
- 可以添加或更新“reference”声明。
- “must”陈述可能会被删除，或者放宽其限制。
- “when”声明可能会被删除或放宽其限制。
- “mandatory”陈述可能被删除或从“true”改为“false”。
- “min-elements”语句可能会被删除或更改为需要更少的元素。
- “max-elements”语句可能会被删除，或被更改为允许更多的元素。
- 可以在不改变定义的语义的情况下添加或更改“description”语句。
- 可以将“base”声明添加到“identity”声明中。
- 如果至少有一个“基本”语句被留下，“base”语句可以从“identityref”类型中删除。
- 可以添加新的 typedefs，分组(groupings)，rpcs，通知(notifications)，扩展(extensions)，功能(features)和身份(identities)。
- 如果新增的数据定义语句没有将强制节点（第 3 节）添加到现有节点或模块或子模块的顶层，或者如果它们有条件地依赖于新功能（即，有“if-feature“的陈述，指的是一个新的特征）。
- 可能会添加新的“case”声明。
- 代表状态数据的节点可以改变来代表配置，只要它不是强制性的（第 3 节）。

- 只要节点不是强制性的（第 3 节），就可以删除“if-feature”语句。
- 可以添加“status”语句，或从“current”更改为“deprecated”或“obsolete”，或从“deprecated”更改为“obsolete”。
- 一个“type”语句可以用另一个“type”语句替换，这个语句不会改变类型的语法或语义。例如，一个内联类型定义可能被一个 typedef 替换，但是一个 int8 类型不能被一个 int16 替换，因为语法会改变。

- 任何一组数据定义节点都可以用另一组语法和语义上等同的节点来代替。例如，一组叶子可能被一个分组的“uses”语句替换为相同的叶子。
- 一个模块可以被拆分成一组子模块，或者一个子模块可以被拆除，前提是模块中的定义除了这里所允许的以外没有任何变化。
- 如果前缀的所有本地使用也被改变，则“prefix”语句可能会改变。

12. 与 YANG 版本 1 共存

YANG 版本 1.1 模块不能包含 YANG 版本 1 子模块(submodule)，YANG 版本 1 模块不能包含 YANG 版本 1.1 子模块。
YANG 版本 1 模块或子模块不得通过修订版本导入 YANG 版本 1.1 模块。
YANG 版本 1.1 模块或子模块可以通过修改导入 YANG 版本 1 模块。
如果一个 YANG 版本 1 的模块 A 导入模块 B 没有修改，而模块 B 更新为 YANG 版本 1.1，则服务器可以同时实现这两个模块（A 和 B）。在这种情况下，NETCONF 服务器务必使用 5.6.4 节中定义的规则通告两个模块，并且应该根据[RFC6020]中定义的规则通告模块 A 以及使用 YANG 版本 1 指定的模块 B 的最新版本。
这个规则的存在是为了允许现有的 YANG 版本 1 模块和 YANG 版本 1.1 模块一起实现。如果没有这个规则，将单个模块更新到 YANG 版本 1.1 将会对导入它的模块产生级联效应，要求所有模块都更新到版本 1.1。

13. YIN

一个 YANG 模块可以被翻译成另一种基于 XML 的语法 YIN。 翻译的模块被称为 YIN 模块。 本节介绍两种格式之间的双向映射规则。
YANG 和 YIN 格式包含使用不同符号的等效信息。 YIN 表示法允许开发人员用 XML 表示 YANG 数据模型，因此可以使用丰富的基于 XML 的工具来进行数据过滤和验证，自动生成代码和文档以及执行其他任务。 可以使用像 XSLT 或 XML 验证程序这样的工具。
YANG 和 YIN 之间的映射不会修改模型的信息内容。 注释和空格不被保留。

13.1. 正式的 YIN 定义

YANG 关键字和 YIN 元素之间有一对一的对应关系。YIN 元素的本地名称与相应的 YANG 关键字相同。这尤其意味着，YIN 文档的文档元素（root）始终是<module>或<submodule>。
对应于 YANG 关键字的 YIN 元素属于关联 URI 为“urn:ietf:params:xml:ns:yang:yin:1”的命名空间。
与扩展关键字对应的 YIN 元素属于通过“extension”语句声明扩展关键字的 YANG 模块的名称空间。

所有 YIN 元素的名称必须使用[XML-NAMES]的标准机制，即“xmlns”和“xmlns:xxx”属性，用它们的命名空间（如上所述）进行适当的限定。
YANG 语句的参数在 YIN 中表示为关键字元素的 XML 属性或子元素。表 1 定义了一组 YANG 关键字的映射。对于扩展，参数映射在“extension”语句中指定（参见第 7.19 节）。以下规则适用于参数：

- 如果参数表示为属性，则此属性没有名称空间。
- 如果参数被表示为一个元素，则它的名称空间与其父关键字元素相同。
- 如果参数表示为元素，则它必须是关键字元素的第一个子元素。

YANG 语句的子语句被表示为关键字元素的（额外的）子元素，它们的相对顺序务必与 YANG 中的子语句顺序相同。
YANG MAY 中的注释可映射到 XML 注释。
表 1：YANG 语句的参数映射

keyword	argument name	yin-element
action	name	false
anydata	name	false
anyxml	name	false
argument	name	false
augment	target-node	false
base	name	false
belongs-to	module	false
bit	name	false
case	name	false
choice	name	false
config	value	false
contact	text	true
container	name	false
default	value	false
description	text	true
deviate	value	false
deviation	target-node	false
enum	name	false
error-app-tag	value	false
error-message	value	true
extension	name	false
feature	name	false
fraction-digits	value	false
grouping	name	false
identity	name	false
if-feature	name	false
import	module	false

```
| include          | module        | false       |
| input            | <no argument> | n/a         |
| key              | value         | false       |
| leaf             | name          | false       |
| leaf-list        | name          | false       |
| length           | value         | false       |
| list             | name          | false       |
| mandatory        | value         | false       |
| max-elements     | value         | false       |
| min-elements     | value         | false       |
| modifier         | value         | false       |
| module           | name          | false       |
| must             | condition     | false       |
| namespace        | uri           | false       |
| notification     | name          | false       |
| ordered-by       | value         | false       |
| organization     | text          | true        |
| output           | <no argument> | n/a         |
| path             | value         | false       |
| pattern          | value         | false       |
| position         | value         | false       |
| prefix           | value         | false       |
| presence         | value         | false       |
| range            | value         | false       |
| reference        | text          | true        |
| refine           | target-node   | false       |
| require-instance | value         | false       |
| revision         | date          | false       |
| revision-date    | date          | false       |
| rpc              | name          | false       |
| status           | value         | false       |
| submodule        | name          | false       |
| type             | name          | false       |
| typedef          | name          | false       |
| unique           | tag           | false       |
| units            | name          | false       |
| uses             | name          | false       |
| value            | value         | false       |
| when             | condition     | false       |
| yang-version     | value         | false       |
| yin-element      | value         | false       |
+------------------+---------------+-------------+
```

13.1.1. 使用示例

以下的 YANG 模块：

```
module example-foo {
  yang-version 1.1;
  namespace "urn:example:foo";
  prefix "foo";

  import example-extensions {
    prefix "myext";
  }

  list interface {
    key "name";
    leaf name {
      type string;
    }

    leaf mtu {
      type uint32;
      description "The MTU of the interface.";
      myext:c-define "MY_MTU";
    }
  }
}
```

在 7.19.3 节定义的扩展名“c-define”被翻译成以下的 YIN：

```
<module name="example-foo"
        xmlns="urn:ietf:params:xml:ns:yang:yin:1"
        xmlns:foo="urn:example:foo"
        xmlns:myext="urn:example:extensions">

  <namespace uri="urn:example:foo"/>
  <prefix value="foo"/>

  <import module="example-extensions">
    <prefix value="myext"/>
  </import>

  <list name="interface">
    <key value="name"/>
    <leaf name="name">
      <type name="string"/>
    </leaf>
    <leaf name="mtu">
```

```
      <type name="uint32"/>
      <description>
        <text>The MTU of the interface.</text>
      </description>
      <myext:c-define name="MY_MTU"/>
    </leaf>
  </list>
</module>
```

14. YANG ABNF 语法

在 YANG 中，几乎所有的陈述都是无序的。 ABNF 语法[RFC5234] [RFC7405]定义了规范的顺序。 为了提高模块可读性，建议按此顺序输入条款。
在 ABNF 语法中，无序的语句用注释标记。
这个语法假定扫描器用一个空格字符替换 YANG 的注释。
<CODE BEGINS> file "yang.abnf"

```
module-stmt         = optsep module-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          module-header-stmts
                          linkage-stmts
                          meta-stmts
                          revision-stmts
                          body-stmts
                      "}" optsep

submodule-stmt      = optsep submodule-keyword sep identifier-arg-str
optsep
                      "{" stmtsep
                          submodule-header-stmts
                          linkage-stmts
                          meta-stmts
                          revision-stmts
                          body-stmts
                      "}" optsep

module-header-stmts = ;; these stmts can appear in any order
                      yang-version-stmt
                      namespace-stmt
                      prefix-stmt

submodule-header-stmts =
                      ;; these stmts can appear in any order
                      yang-version-stmt
                      belongs-to-stmt
```

```
meta-stmts          = ;; these stmts can appear in any order
                    [organization-stmt]
                    [contact-stmt]
                    [description-stmt]
                    [reference-stmt]

linkage-stmts       = ;; these stmts can appear in any order
                    *import-stmt
                    *include-stmt

revision-stmts      = *revision-stmt

body-stmts          = *(extension-stmt /
                      feature-stmt /
                      identity-stmt /
                      typedef-stmt /
                      grouping-stmt /
                      data-def-stmt /
                      augment-stmt /
                      rpc-stmt /
                      notification-stmt /
                      deviation-stmt)

data-def-stmt       = container-stmt /
                    leaf-stmt /
                    leaf-list-stmt /
                    list-stmt /
                    choice-stmt /
                    anydata-stmt /
                    anyxml-stmt /
                    uses-stmt

yang-version-stmt   = yang-version-keyword sep yang-version-arg-str
                    stmtend

yang-version-arg-str = < a string that matches the rule >
                     < yang-version-arg >

yang-version-arg    = "1.1"

import-stmt         = import-keyword sep identifier-arg-str optsep
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        prefix-stmt
```

```
                       [revision-date-stmt]
                       [description-stmt]
                       [reference-stmt]
                     "}" stmtsep
include-stmt         = include-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [revision-date-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

namespace-stmt       = namespace-keyword sep uri-str stmtend

uri-str              = < a string that matches the rule >
                       < URI in RFC 3986 >

prefix-stmt          = prefix-keyword sep prefix-arg-str stmtend

belongs-to-stmt      = belongs-to-keyword sep identifier-arg-str
                       optsep
                       "{" stmtsep
                           prefix-stmt
                       "}" stmtsep

organization-stmt    = organization-keyword sep string stmtend

contact-stmt         = contact-keyword sep string stmtend

description-stmt     = description-keyword sep string stmtend

reference-stmt       = reference-keyword sep string stmtend

units-stmt           = units-keyword sep string stmtend

revision-stmt        = revision-keyword sep revision-date optsep
                       (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep
```

```
revision-date       = date-arg-str

revision-date-stmt  = revision-date-keyword sep revision-date stmtend

extension-stmt      = extension-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [argument-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

argument-stmt       = argument-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           [yin-element-stmt]
                       "}") stmtsep

yin-element-stmt    = yin-element-keyword sep yin-element-arg-str
                      stmtend

yin-element-arg-str = < a string that matches the rule >
                      < yin-element-arg >

yin-element-arg     = true-keyword / false-keyword

identity-stmt       = identity-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *base-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

base-stmt           = base-keyword sep identifier-ref-arg-str
                      stmtend

feature-stmt        = feature-keyword sep identifier-arg-str optsep
                      (";" /
```

```
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          *if-feature-stmt
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                      "}") stmtsep
if-feature-stmt     = if-feature-keyword sep if-feature-expr-str
                    stmtend

if-feature-expr-str = < a string that matches the rule >
                    < if-feature-expr >

if-feature-expr     = if-feature-term
                      [sep or-keyword sep if-feature-expr]

if-feature-term     = if-feature-factor
                      [sep and-keyword sep if-feature-term]

if-feature-factor   = not-keyword sep if-feature-factor /
                    "(" optsep if-feature-expr optsep ")" /
                    identifier-ref-arg

typedef-stmt        = typedef-keyword sep identifier-arg-str optsep
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        type-stmt
                        [units-stmt]
                        [default-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                     "}" stmtsep

type-stmt           = type-keyword sep identifier-ref-arg-str optsep
                    (";" /
                     "{" stmtsep
                         [type-body-stmts]
                     "}") stmtsep

type-body-stmts     = numerical-restrictions /
                    decimal64-specification /
                    string-restrictions /
                    enum-specification /
```

```
                    leafref-specification /
                    identityref-specification /
                    instance-identifier-specification /
                    bits-specification /
                    union-specification /
                    binary-specification
                    numerical-restrictions = [range-stmt]

range-stmt          = range-keyword sep range-arg-str optsep
                   (";" /
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        [error-message-stmt]
                        [error-app-tag-stmt]
                        [description-stmt]
                        [reference-stmt]
                     "}") stmtsep

decimal64-specification = ;; these stmts can appear in any order
                       fraction-digits-stmt
                       [range-stmt]

fraction-digits-stmt = fraction-digits-keyword sep
                     fraction-digits-arg-str stmtend

fraction-digits-arg-str = < a string that matches the rule >
                       < fraction-digits-arg >

fraction-digits-arg = ("1" ["0" / "1" / "2" / "3" / "4" /
                         "5" / "6" / "7" / "8"])
                    / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"

string-restrictions = ;; these stmts can appear in any order
                    [length-stmt]
                    *pattern-stmt

length-stmt         = length-keyword sep length-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [error-message-stmt]
                         [error-app-tag-stmt]
                         [description-stmt]
                         [reference-stmt]
```

```
                    "}") stmtsep

pattern-stmt        = pattern-keyword sep string optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [modifier-stmt]
                         [error-message-stmt]
                         [error-app-tag-stmt]
                         [description-stmt]
                         [reference-stmt]
                      "}") stmtsep

modifier-stmt       = modifier-keyword sep modifier-arg-str stmtend

modifier-arg-str    = < a string that matches the rule >
                    < modifier-arg >

modifier-arg        = invert-match-keyword

default-stmt        = default-keyword sep string stmtend

enum-specification  = 1*enum-stmt

enum-stmt           = enum-keyword sep string optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         *if-feature-stmt
                         [value-stmt]
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                      "}") stmtsep

leafref-specification =
                    ;; these stmts can appear in any order
                    path-stmt
                    [require-instance-stmt]

path-stmt           = path-keyword sep path-arg-str stmtend

require-instance-stmt = require-instance-keyword sep
                        require-instance-arg-str stmtend
```

```
require-instance-arg-str = < a string that matches the rule >
                       < require-instance-arg >

require-instance-arg = true-keyword / false-keyword

instance-identifier-specification =
                     [require-instance-stmt]

identityref-specification =
                     1*base-stmt

union-specification = 1*type-stmt

binary-specification = [length-stmt]

bits-specification  = 1*bit-stmt

bit-stmt            = bit-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           [position-stmt]
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                       "}") stmtsep

position-stmt       = position-keyword sep
                      position-value-arg-str stmtend

position-value-arg-str = < a string that matches the rule >
                         < position-value-arg >

position-value-arg  = non-negative-integer-value

status-stmt         = status-keyword sep status-arg-str stmtend

status-arg-str      = < a string that matches the rule >
                      < status-arg >

status-arg          = current-keyword /
                      obsolete-keyword /
```

```
                     deprecated-keyword

config-stmt         = config-keyword sep
                      config-arg-str stmtend

config-arg-str      = < a string that matches the rule >
                      < config-arg >

config-arg          = true-keyword / false-keyword

mandatory-stmt      = mandatory-keyword sep
                      mandatory-arg-str stmtend

mandatory-arg-str   = < a string that matches the rule >
                      < mandatory-arg >

mandatory-arg       = true-keyword / false-keyword

presence-stmt       = presence-keyword sep string stmtend

ordered-by-stmt     = ordered-by-keyword sep
                      ordered-by-arg-str stmtend

ordered-by-arg-str  = < a string that matches the rule >
                      < ordered-by-arg >

ordered-by-arg      = user-keyword / system-keyword

must-stmt           = must-keyword sep string optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [error-message-stmt]
                           [error-app-tag-stmt]
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

error-message-stmt  = error-message-keyword sep string stmtend

error-app-tag-stmt  = error-app-tag-keyword sep string stmtend

min-elements-stmt   = min-elements-keyword sep
                      min-value-arg-str stmtend
```

```
min-value-arg-str   = < a string that matches the rule >
                     < min-value-arg >

min-value-arg       = non-negative-integer-value

max-elements-stmt   = max-elements-keyword sep
                     max-value-arg-str stmtend

max-value-arg-str   = < a string that matches the rule >
                     < max-value-arg >

max-value-arg       = unbounded-keyword /
                     positive-integer-value

value-stmt          = value-keyword sep integer-value-str stmtend

integer-value-str   = < a string that matches the rule >
                    < integer-value >

grouping-stmt       = grouping-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                         *(typedef-stmt / grouping-stmt)
                         *data-def-stmt
                         *action-stmt
                         *notification-stmt
                     "}") stmtsep

container-stmt      = container-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         *must-stmt
                         [presence-stmt]
                         [config-stmt]
                         [status-stmt]
                         [description-stmt]
```

```
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        *data-def-stmt
                        *action-stmt
                        *notification-stmt
                     "}") stmtsep

leaf-stmt          = leaf-keyword sep identifier-arg-str optsep
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        [when-stmt]
                        *if-feature-stmt
                        type-stmt
                        [units-stmt]
                        *must-stmt
                        [default-stmt]
                        [config-stmt]
                        [mandatory-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                     "}" stmtsep

leaf-list-stmt      = leaf-list-keyword sep identifier-arg-str optsep
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        [when-stmt]
                        *if-feature-stmt
                        type-stmt stmtsep
                        [units-stmt]
                        *must-stmt
                        *default-stmt
                        [config-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                        [ordered-by-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                     "}" stmtsep

list-stmt          = list-keyword sep identifier-arg-str optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
```

```
                        [when-stmt]
                        *if-feature-stmt
                        *must-stmt
                        [key-stmt]
                        *unique-stmt
                        [config-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                        [ordered-by-stmt]
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        1*data-def-stmt
                        *action-stmt
                        *notification-stmt
                      "}" stmtsep

key-stmt           = key-keyword sep key-arg-str stmtend

key-arg-str        = < a string that matches the rule >
                     < key-arg >

key-arg            = node-identifier *(sep node-identifier)

unique-stmt        = unique-keyword sep unique-arg-str stmtend

unique-arg-str     = < a string that matches the rule >
                     < unique-arg >

unique-arg         = descendant-schema-nodeid
                     *(sep descendant-schema-nodeid)
choice-stmt        = choice-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         [default-stmt]
                         [config-stmt]
                         [mandatory-stmt]
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
```

```
                        *(short-case-stmt / case-stmt)
                     "}") stmtsep

short-case-stmt      = choice-stmt /
                     container-stmt /
                     leaf-stmt /
                     leaf-list-stmt /
                     list-stmt /
                     anydata-stmt /
                     anyxml-stmt

case-stmt            = case-keyword sep identifier-arg-str optsep
                     (";" /
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          [when-stmt]
                          *if-feature-stmt
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                          *data-def-stmt
                      "}") stmtsep

anydata-stmt         = anydata-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         *must-stmt
                         [config-stmt]
                         [mandatory-stmt]
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                      "}") stmtsep

anyxml-stmt          = anyxml-keyword sep identifier-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         *must-stmt
```

```
                       [config-stmt]
                       [mandatory-stmt]
                       [status-stmt]
                       [description-stmt]
                       [reference-stmt]
                     "}") stmtsep

uses-stmt           = uses-keyword sep identifier-ref-arg-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [when-stmt]
                         *if-feature-stmt
                         [status-stmt]
                         [description-stmt]
                         [reference-stmt]
                         *refine-stmt
                         *uses-augment-stmt
                     "}") stmtsep

refine-stmt         = refine-keyword sep refine-arg-str optsep
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *must-stmt
                           [presence-stmt]
                           *default-stmt
                           [config-stmt]
                           [mandatory-stmt]
                           [min-elements-stmt]
                           [max-elements-stmt]
                           [description-stmt]
                           [reference-stmt]
                         "}" stmtsep

refine-arg-str      = < a string that matches the rule >
                      < refine-arg >

refine-arg          = descendant-schema-nodeid

uses-augment-stmt   = augment-keyword sep uses-augment-arg-str optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          [when-stmt]
```

```
                        *if-feature-stmt
                        [status-stmt]
                        [description-stmt]
                        [reference-stmt]
                        1*(data-def-stmt / case-stmt /
                           action-stmt / notification-stmt)
                      "}" stmtsep

uses-augment-arg-str = < a string that matches the rule >
                      < uses-augment-arg >

uses-augment-arg    = descendant-schema-nodeid

augment-stmt        = augment-keyword sep augment-arg-str optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          [when-stmt]
                          *if-feature-stmt
                          [status-stmt]
                          [description-stmt]
                          [reference-stmt]
                          1*(data-def-stmt / case-stmt /
                             action-stmt / notification-stmt)
                       "}" stmtsep

augment-arg-str     = < a string that matches the rule >
                      < augment-arg >

augment-arg         = absolute-schema-nodeid

when-stmt           = when-keyword sep string optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           [description-stmt]
                           [reference-stmt]
                        "}") stmtsep

rpc-stmt            = rpc-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           [status-stmt]
```

```
                          [description-stmt]
                          [reference-stmt]
                          *(typedef-stmt / grouping-stmt)
                          [input-stmt]
                          [output-stmt]
                        "}") stmtsep

action-stmt         = action-keyword sep identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           [status-stmt]
                           [description-stmt]
                           [reference-stmt]
                           *(typedef-stmt / grouping-stmt)
                           [input-stmt]
                           [output-stmt]
                       "}") stmtsep

input-stmt          = input-keyword optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          *must-stmt
                          *(typedef-stmt / grouping-stmt)
                          1*data-def-stmt
                      "}" stmtsep

output-stmt         = output-keyword optsep
                      "{" stmtsep
                          ;; these stmts can appear in any order
                          *must-stmt
                          *(typedef-stmt / grouping-stmt)
                          1*data-def-stmt
                      "}" stmtsep

notification-stmt   = notification-keyword sep
                      identifier-arg-str optsep
                      (";" /
                       "{" stmtsep
                           ;; these stmts can appear in any order
                           *if-feature-stmt
                           *must-stmt
                           [status-stmt]
```

```
                        [description-stmt]
                        [reference-stmt]
                        *(typedef-stmt / grouping-stmt)
                        *data-def-stmt
                      "}") stmtsep

deviation-stmt      = deviation-keyword sep
                    deviation-arg-str optsep
                    "{" stmtsep
                        ;; these stmts can appear in any order
                        [description-stmt]
                        [reference-stmt]
                        (deviate-not-supported-stmt /
                          1*(deviate-add-stmt /
                             deviate-replace-stmt /
                             deviate-delete-stmt))
                    "}" stmtsep

deviation-arg-str   = < a string that matches the rule >
                    < deviation-arg >

deviation-arg       = absolute-schema-nodeid

deviate-not-supported-stmt =
                    deviate-keyword sep
                    not-supported-keyword-str stmtend

deviate-add-stmt    = deviate-keyword sep add-keyword-str optsep
                    (";" /
                     "{" stmtsep
                         ;; these stmts can appear in any order
                         [units-stmt]
                         *must-stmt
                         *unique-stmt
                         *default-stmt
                         [config-stmt]
                         [mandatory-stmt]
                         [min-elements-stmt]
                         [max-elements-stmt]
                     "}") stmtsep

deviate-delete-stmt = deviate-keyword sep delete-keyword-str optsep
                    (";" /
                     "{" stmtsep
```

```
                        ;; these stmts can appear in any order
                        [units-stmt]
                        *must-stmt
                        *unique-stmt
                        *default-stmt
                     "}") stmtsep

deviate-replace-stmt = deviate-keyword sep replace-keyword-str optsep
                    (";" /
                     "{" stmtsep
                        ;; these stmts can appear in any order
                        [type-stmt]
                        [units-stmt]
                        [default-stmt]
                        [config-stmt]
                        [mandatory-stmt]
                        [min-elements-stmt]
                        [max-elements-stmt]
                     "}") stmtsep

not-supported-keyword-str = < a string that matches the rule >
                          < not-supported-keyword >

add-keyword-str     = < a string that matches the rule >
                    < add-keyword >

delete-keyword-str  = < a string that matches the rule >
                    < delete-keyword >

replace-keyword-str = < a string that matches the rule >
                    < replace-keyword >

;; represents the usage of an extension
unknown-statement   = prefix ":" identifier [sep string] optsep
                    (";" /
                     "{" optsep
                        *((yang-stmt / unknown-statement) optsep)
                      "}") stmtsep

yang-stmt           = action-stmt /
                    anydata-stmt /
                    anyxml-stmt /
                    argument-stmt /
                    augment-stmt /
```

```
                base-stmt /
                belongs-to-stmt /
                bit-stmt /
                case-stmt /
                choice-stmt /
                config-stmt /
                contact-stmt /
                container-stmt /
                default-stmt /
                description-stmt /
                deviate-add-stmt /

                deviate-delete-stmt /
                deviate-not-supported-stmt /
                deviate-replace-stmt /
                deviation-stmt /
                enum-stmt /
                error-app-tag-stmt /
                error-message-stmt /
                extension-stmt /
                feature-stmt /
                fraction-digits-stmt /
                grouping-stmt /
                identity-stmt /
                if-feature-stmt /
                import-stmt /
                include-stmt /
                input-stmt /
                key-stmt /
                leaf-list-stmt /
                leaf-stmt /
                length-stmt /
                list-stmt /
                mandatory-stmt /
                max-elements-stmt /
                min-elements-stmt /
                modifier-stmt /
                module-stmt /
                must-stmt /
                namespace-stmt /
                notification-stmt /
                ordered-by-stmt /
                organization-stmt /
                output-stmt /
```

```
                  path-stmt /
                  pattern-stmt /
                  position-stmt /
                  prefix-stmt /
                  presence-stmt /
                  range-stmt /
                  reference-stmt /
                  refine-stmt /
                  require-instance-stmt /
                  revision-date-stmt /
                  revision-stmt /
                  rpc-stmt /
                  status-stmt /
                  submodule-stmt /
                  typedef-stmt /
                  type-stmt /
                  unique-stmt /
                  units-stmt /
                  uses-augment-stmt /
                  uses-stmt /
                  value-stmt /
                  when-stmt /
                  yang-version-stmt /
                  yin-element-stmt

;; Ranges

range-arg-str       = < a string that matches the rule >
                  < range-arg >

range-arg           = range-part *(optsep "|" optsep range-part)

range-part          = range-boundary
                  [optsep ".." optsep range-boundary]

range-boundary      = min-keyword / max-keyword /
                  integer-value / decimal-value

;; Lengths

length-arg-str      = < a string that matches the rule >
                  < length-arg >

length-arg          = length-part *(optsep "|" optsep length-part)
```

```
length-part         = length-boundary
                    [optsep ".." optsep length-boundary]

length-boundary     = min-keyword / max-keyword /
                    non-negative-integer-value

;; Date

date-arg-str        = < a string that matches the rule >
                    < date-arg >

date-arg            = 4DIGIT "-" 2DIGIT "-" 2DIGIT

;; Schema Node Identifiers

schema-nodeid       = absolute-schema-nodeid /
                    descendant-schema-nodeid

absolute-schema-nodeid = 1*("/" node-identifier)

descendant-schema-nodeid =
                      node-identifier
                      [absolute-schema-nodeid]

node-identifier     = [prefix ":"] identifier

;; Instance Identifiers

instance-identifier = 1*("/" (node-identifier
                             [1*key-predicate /
                              leaf-list-predicate /
                              pos]))

key-predicate       = "[" *WSP key-predicate-expr *WSP "]"

key-predicate-expr  = node-identifier *WSP "=" *WSP quoted-string

leaf-list-predicate = "[" *WSP leaf-list-predicate-expr *WSP "]"

leaf-list-predicate-expr = "." *WSP "=" *WSP quoted-string

pos                 = "[" *WSP positive-integer-value *WSP "]"
```

```
quoted-string       = (DQUOTE string DQUOTE) / (SQUOTE string SQUOTE)

;; leafref path

path-arg-str        = < a string that matches the rule >
                      < path-arg >

path-arg            = absolute-path / relative-path

absolute-path       = 1*("/" (node-identifier *path-predicate))

relative-path       = 1*("../") descendant-path

descendant-path     = node-identifier
                      [*path-predicate absolute-path]

path-predicate      = "[" *WSP path-equality-expr *WSP "]"

path-equality-expr  = node-identifier *WSP "=" *WSP path-key-expr

path-key-expr       = current-function-invocation *WSP "/" *WSP
rel-path-keyexpr

rel-path-keyexpr    = 1*(".." *WSP "/" *WSP)
                    *(node-identifier *WSP "/" *WSP)
                    node-identifier

;;; Keywords, using the syntax for case-sensitive strings (RFC 7405)

;; statement keywords
action-keyword          = %s"action"
anydata-keyword         = %s"anydata"
anyxml-keyword          = %s"anyxml"
argument-keyword        = %s"argument"
augment-keyword         = %s"augment"
base-keyword            = %s"base"
belongs-to-keyword      = %s"belongs-to"
bit-keyword             = %s"bit"
case-keyword            = %s"case"
choice-keyword          = %s"choice"
config-keyword          = %s"config"
contact-keyword         = %s"contact"
container-keyword       = %s"container"
default-keyword         = %s"default"
```

```
description-keyword      = %s"description"
deviate-keyword          = %s"deviate"
deviation-keyword        = %s"deviation"
enum-keyword             = %s"enum"
error-app-tag-keyword    = %s"error-app-tag"
error-message-keyword    = %s"error-message"
extension-keyword        = %s"extension"
feature-keyword          = %s"feature"
fraction-digits-keyword  = %s"fraction-digits"
grouping-keyword         = %s"grouping"
identity-keyword         = %s"identity"
if-feature-keyword       = %s"if-feature"
import-keyword           = %s"import"
include-keyword          = %s"include"
input-keyword            = %s"input"
key-keyword              = %s"key"
leaf-keyword             = %s"leaf"
leaf-list-keyword        = %s"leaf-list"
length-keyword           = %s"length"
list-keyword             = %s"list"
mandatory-keyword        = %s"mandatory"
max-elements-keyword     = %s"max-elements"
min-elements-keyword     = %s"min-elements"
modifier-keyword         = %s"modifier"
module-keyword           = %s"module"
must-keyword             = %s"must"
namespace-keyword        = %s"namespace"

notification-keyword     = %s"notification"
ordered-by-keyword       = %s"ordered-by"
organization-keyword     = %s"organization"
output-keyword           = %s"output"
path-keyword             = %s"path"
pattern-keyword          = %s"pattern"
position-keyword         = %s"position"
prefix-keyword           = %s"prefix"
presence-keyword         = %s"presence"
range-keyword            = %s"range"
reference-keyword        = %s"reference"
refine-keyword           = %s"refine"
require-instance-keyword = %s"require-instance"
revision-keyword         = %s"revision"
revision-date-keyword    = %s"revision-date"
rpc-keyword              = %s"rpc"
```

```
status-keyword           = %s"status"
submodule-keyword        = %s"submodule"
type-keyword             = %s"type"
typedef-keyword          = %s"typedef"
unique-keyword           = %s"unique"
units-keyword            = %s"units"
uses-keyword             = %s"uses"
value-keyword            = %s"value"
when-keyword             = %s"when"
yang-version-keyword     = %s"yang-version"
yin-element-keyword      = %s"yin-element"

;; other keywords

add-keyword              = %s"add"
current-keyword          = %s"current"
delete-keyword           = %s"delete"
deprecated-keyword       = %s"deprecated"
false-keyword            = %s"false"
invert-match-keyword     = %s"invert-match"
max-keyword              = %s"max"
min-keyword              = %s"min"
not-supported-keyword    = %s"not-supported"
obsolete-keyword         = %s"obsolete"
replace-keyword          = %s"replace"
system-keyword           = %s"system"
true-keyword             = %s"true"
unbounded-keyword        = %s"unbounded"
user-keyword             = %s"user"

and-keyword              = %s"and"
or-keyword               = %s"or"
not-keyword              = %s"not"

current-function-invocation = current-keyword *WSP "(" *WSP ")"

;;; Basic Rules

prefix-arg-str      = < a string that matches the rule >
                      < prefix-arg >

prefix-arg          = prefix

prefix              = identifier
```

```
identifier-arg-str  = < a string that matches the rule >
                      < identifier-arg >

identifier-arg      = identifier

identifier          = (ALPHA / "_")
                      *(ALPHA / DIGIT / "_" / "-" / ".")

identifier-ref-arg-str = < a string that matches the rule >
                         < identifier-ref-arg >

identifier-ref-arg  = identifier-ref

identifier-ref      = [prefix ":"] identifier

string              = < an unquoted string, as returned by >
                      < the scanner, that matches the rule >
                      < yang-string >

yang-string         = *yang-char

;; any Unicode or ISO/IEC 10646 character, including tab, carriage
;; return, and line feed but excluding the other C0 control
;; characters, the surrogate blocks, and the noncharacters
yang-char = %x09 / %x0A / %x0D / %x20-D7FF /
                            ; exclude surrogate blocks %xD800-DFFF
          %xE000-FDCF /     ; exclude noncharacters %xFDD0-FDEF
          %xFDF0-FFFD /     ; exclude noncharacters %xFFFE-FFFF
          %x10000-1FFFD /   ; exclude noncharacters %x1FFFE-1FFFF
          %x20000-2FFFD /   ; exclude noncharacters %x2FFFE-2FFFF
          %x30000-3FFFD /   ; exclude noncharacters %x3FFFE-3FFFF
          %x40000-4FFFD /   ; exclude noncharacters %x4FFFE-4FFFF
          %x50000-5FFFD /   ; exclude noncharacters %x5FFFE-5FFFF
          %x60000-6FFFD /   ; exclude noncharacters %x6FFFE-6FFFF
          %x70000-7FFFD /   ; exclude noncharacters %x7FFFE-7FFFF
          %x80000-8FFFD /   ; exclude noncharacters %x8FFFE-8FFFF
          %x90000-9FFFD /   ; exclude noncharacters %x9FFFE-9FFFF
          %xA0000-AFFFD /   ; exclude noncharacters %xAFFFE-AFFFF
          %xB0000-BFFFD /   ; exclude noncharacters %xBFFFE-BFFFF
          %xC0000-CFFFD /   ; exclude noncharacters %xCFFFE-CFFFF
          %xD0000-DFFFD /   ; exclude noncharacters %xDFFFE-DFFFF
          %xE0000-EFFFD /   ; exclude noncharacters %xEFFFE-EFFFF
          %xF0000-FFFFD /   ; exclude noncharacters %xFFFFE-FFFFF
```

```
          %x100000-10FFFD  ; exclude noncharacters %x10FFFE-10FFFF

integer-value       = ("-" non-negative-integer-value) /
                      non-negative-integer-value

non-negative-integer-value = "0" / positive-integer-value

positive-integer-value = (non-zero-digit *DIGIT)

zero-integer-value  = 1*DIGIT

stmtend             = optsep (";" / "{" stmtsep "}") stmtsep

sep                 = 1*(WSP / line-break)
                      ; unconditional separator

optsep              = *(WSP / line-break)

stmtsep             = *(WSP / line-break / unknown-statement)

line-break          = CRLF / LF

non-zero-digit      = %x31-39

decimal-value       = integer-value ("." zero-integer-value)

SQUOTE              = %x27
                      ; single quote

;;; core rules from RFC 5234

ALPHA               = %x41-5A / %x61-7A
                      ; A-Z / a-z

CR                  = %x0D
                      ; carriage return

CRLF                = CR LF
                      ; Internet standard newline

DIGIT               = %x30-39
                      ; 0-9

DQUOTE              = %x22
```

```
                      ; double quote

HTAB                = %x09
                      ; horizontal tab

LF                  = %x0A
                      ; line feed

SP                  = %x20
                      ; space

WSP                 = SP / HTAB
                      ; whitespace
```

<CODE ENDS>

15. 与 YANG 相关的错误的 NETCONF 错误响应

定义了一些与数据模型处理有关的错误情况的 NETCONF 错误响应。 如果相关的 YANG 语句有一个“error-app-tag”子语句，则覆盖下面指定的默认值。

15.1. 违反“unique”声明的数据的错误消息

如果一个 NETCONF 操作会导致一个“unique”约束失效的配置数据，则必须返回以下错误：

```
error-tag:      operation-failed
error-app-tag:  data-not-unique
error-info:     <non-unique>: Contains an instance identifier that points
to a leaf that invalidates the "unique" constraint.  This element is
present once for each non-unique leaf.
                The <non-unique> element is in the YANG namespace
("urn:ietf:params:xml:ns:yang:1").
```

15.2. 违反“max-elements”语句的数据的错误消息

如果一个 NETCONF 操作会导致一个列表或一个叶子列表中的条目太多的配置数据，则必须返回以下错误：

```
error-tag:      operation-failed
error-app-tag:  too-many-elements
```

即使存在多个额外的子项，该错误也会返回一次，错误路径将标识列表节点。

15.3. 违反“min-elements”声明的数据的错误信息

如果一个 NETCONF 操作会导致一个列表或一个叶子列表的条目太少的配置数据，则必须返回以下错误：

```
error-tag:      operation-failed
error-app-tag:  too-few-elements
```

这个错误被返回一次，错误路径标识列表节点，即使有多个子节点丢失。

15.4. 违反“must”声明的数据的错误消息

如果 NETCONF 操作会导致违反“must”语句的配置数据，则必须返回以下错误，除非“must”语句存在特定的“error-app-tag”子语句。

```
error-tag:      operation-failed
error-app-tag:  must-violation
```

15.5. 针对违反“require-instance”声明的数据的错误消息

如果 NETCONF 操作会导致配置数据，其中标有 require-instance“true”的“instance-identifier”或“leafref”类型的叶片指的是不存在的实例，则必须返回以下错误：

```
error-tag:      data-missing
error-app-tag:  instance-required
error-path:     Path to the instance-identifier or leafref leaf.
```

15.6. 违反强制性“choice”声明的数据的错误消息

如果一个 NETCONF 操作会导致配置数据，在强制选项中不存在节点，则必须返回以下错误：

```
error-tag:      data-missing
error-app-tag:  missing-choice
error-path:     Path to the element with the missing choice.
error-info:     <missing-choice>: Contains the name of the missing
                mandatory choice.

                The <missing-choice> element is in the YANG
                namespace ("urn:ietf:params:xml:ns:yang:1").
```

15.7. “insert”操作的错误消息

如果在列表或叶列表节点的<edit-config>中使用了“insert”和“key”或“value”属性，并且“key”或“value”引用了不存在的实例，必须返回以下错误：

```
error-tag:      bad-attribute
```

```
error-app-tag:  missing-instance
```

16. IANA 考虑

本文档从“网络配置协议（NETCONF）能力 URNs”注册表中注册一个能力标识符 URN：

Index	Capability Identifier
:yang-library	urn:ietf:params:netconf:capability:yang-library:1.0

17. 安全考虑

本文档定义了一种用于编写和阅读管理信息描述的语言。语言本身对互联网没有安全影响。

基本 NETCONF 协议的相关考虑也是相关的（参见[RFC6241]第 9 节）。
在 YANG 中建模的数据可能包含敏感信息。 YANG 中定义的 RPC 或通知可能会传输敏感信息。
安全问题与 YANG 建模数据的使用有关。这些问题应该在描述数据模型的文件中进行处理，并且用来处理数据的接口文件，例如 NETCONF 文件。
以 YANG 为模型的数据依赖于：

- 用于发送敏感信息的传输基础设施的安全性。
- 存储或释放这种敏感信息的应用程序的安全性。
- 适当的身份验证和访问控制机制来限制敏感数据的使用。

YANG 解析器对于格式错误的文档需要很强大。从未知或不可信的来源读取格式错误的文件可能导致攻击者获得运行 YANG 解析器的用户的权限。在极端的情况下，整个机器可能会受到影响。

18. 参考

18.1. 规范性参考文献

[ISO.10646] International Organization for Standardization, "Information Technology - Universal Multiple-Octet Coded Character Set (UCS)", ISO Standard 10646:2014, 2014.
[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, http://www.rfc-editor.org/info/rfc3629.
[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, http://www.rfc-editor.org/info/rfc3986.
[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, http://www.rfc-editor.org/info/rfc4648.
[RFC5234] Crocker, D., Ed., and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, http://www.rfc-editor.org/info/rfc5234.
[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, http://www.rfc-editor.org/info/rfc5277.
[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, http://www.rfc-editor.org/info/rfc6241.
[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, http://www.rfc-editor.org/info/rfc7405.
[RFC7895] Bierman, A., Bjorklund, M., and K. Watsen, "YANG Module Library", RFC 7895, DOI 10.17487/RFC7895, June 2016, http://www.rfc-editor.org/info/rfc7895.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C Recommendation REC-xml-20081126, November 2008, https://www.w3.org/TR/2008/REC-xml-20081126/.
[XML-NAMES] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, http://www.w3.org/TR/2009/REC-xml-names-20091208.
[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116.
[XSD-TYPES] Biron, P. and A. Malhotra, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

18.2. 信息参考

[CoMI] van der Stok, P. and A. Bierman, "CoAP Management Interface", Work in Progress, draft-vanderstok-core-comi-09, March 2016.
[IEEE754-2008] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2008, DOI 10.1109/IEEESTD.2008.4610935, 2008, http://standards.ieee.org/findstds/standard/754-2008.html.
[RESTCONF] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", Work in Progress, draft-ietf-netconf-restconf-16, August 2016.
[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, http://www.rfc-editor.org/info/rfc2578.
[RFC2579] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Textual Conventions for SMIv2" STD 58, RFC 2579, DOI 10.17487/RFC2579, April 1999, http://www.rfc-editor.org/info/rfc2579.
[RFC3780] Strauss, F. and J. Schoenwaelder, "SMIng - Next Generation Structure of Management Information", RFC 3780, DOI 10.17487/RFC3780, May 2004, http://www.rfc-editor.org/info/rfc3780.
[RFC4844] Daigle, L., Ed., and Internet Architecture Board, "The RFC Series and RFC Editor", RFC 4844, DOI 10.17487/RFC4844, July 2007, http://www.rfc-editor.org/info/rfc4844.
[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, http://www.rfc-editor.org/info/rfc6020.
[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, http://www.rfc-editor.org/info/rfc6643.
[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, http://www.rfc-editor.org/info/rfc6991.
[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, http://www.rfc-editor.org/info/rfc7951.
[XPATH2.0] Berglund, A., Boag, S., Chamberlin, D., Fernandez, M., Kay, M., Robie, J., and J. Simeon, "XML Path Language (XPath) 2.0 (Second Edition)", World Wide Web Consortium Recommendation REC-xpath20-20101214, December 2010, http://www.w3.org/TR/2010/REC-xpath20-20101214.
[XSLT] Clark, J., "XSL Transformations (XSLT) Version 1.0", World Wide Web Consortium Recommendation REC-xslt-19991116, November 1999, http://www.w3.org/TR/1999/REC-xslt-19991116.
[YANG-Guidelines] Bierman, A., "Guidelines for Authors and Reviewers of YANG Data Model Documents", Work in Progress, draft-ietf-netmod-rfc6087bis-07, July 2016.

致谢

编辑要感谢以下所有人

对本文件的各种草案提供了有用的意见：

Mehmet Ersue, Washam Fan, Joel Halpern, Per Hedeland, Leif Johansson, Ladislav Lhotka, Lionel Morand, Gerhard Muenz, Peyman Owladi, Tom Petch, Randy Presuhn, David Reid, Jernej Tuljak, Kent Watsen, Bert Wijnen, Robert Wilton, and Dale Worley.

贡献者

以下人员对最初的`YANG`文件作出了重大贡献：

- Andy Bierman (YumaWorks)
- Balazs Lengyel (Ericsson)
- David Partain (Ericsson)
- Juergen Schoenwaelder (Jacobs University Bremen)
- Phil Shafer (Juniper Networks)

作者的地址

Martin Bjorklund (editor) Tail-f Systems

Email: mbj@tail-f.com