# RESTCONF 协议 (2016.4.23)


## 摘要

这份文档描述了一种 RESTful 协议，此协议提供 HTTP 上的编程接口，用于访问 YANG 定义的数据，使用 NETCONF 定义的数据存储。


## 此备忘的状态

此互联网草案与 BCP 78 和 BCP 79 条款保持完全一致。

互联网草案是指 Internet Engineering Task Force (IETF)的工作文档，需要注意的是，其它组织可能也会分发工作文档为互联网草案。当前的互联网草案列表可以在http://datatracker.ietf.org/drafts/current/找到。

互联网草案是草稿文档，其最长有效期为 6 个月，期间随时可能会更新，替换或废弃。不建议使用互联网草案作为参考材料，也不要在除了“进行中”的主体中引用。

此互联网草案将在 2016 年 4 月 23 日过期。



# 目录

- 2.4.1 内容模型
- 2.4.2 编辑模型
- 2.4.3 锁定模型
- 2.4.4 持久模型
- 2.4.5 预设值模型
- 2.3.1 RESTCONF资源类型
- 2.3.2 资源发现
- 2.1 消息模型
- 2.2 通知模型
- 2.3 资源模型

# 1. 介绍

有这样一种标准机制的需求，它允许 WEB 应用以一种模块化和可扩展的方式访问网络设备中的配置数据，操作数据，数据模型特定的协议操作，及通知事件。

此文档描述了一种叫做 RESTCONF 的 RESTful 协议，其运行于 HTTP[RFC2616] 上，可用于访问 YANG[RFC6020] 中定义的数据，及使用 NETCONF[RFC6241] 中定义的数据存储。

NETCONF 协议定义了配置数据存储和一系列的创建，获取，更新，删除(CRUD)操作，可用于访问数据存储。YANG 语言定义了数据存储内容，操作数据，自定义协议操作，通知事件的语法和语义。RESFful 操作用于访问数据存储中的分层的数据。

RESTful API 可以被用来创建对 NETCONF 数据存储的 CRUD 操作，包含 YANG 定义的数据。这可以通过一个简化的方式来完成 HTTP 的 RESTful 的设计原则。 由于 NETCONF 协议操作不相关，用户不需要为了使用 RESTful API 而需要任何 NETCONF 储备知识。

配置数据和状态数据被公开做为可检索的资源并使用 GET 方法。资源代表配置数据可以通过 DELETE， PATCH，POST 和 PUT 方法被修改。数据模型的特定的协议操作被定义为 YANG 的“rpc”通过 POST 方法调用。数据模型指定通知事件，并被 YANG 定义为“通知”，并可以被访问（TBD 推送方式）。

## 1.1 NETCONF 功能的简单子集

RESTful API 使用的框架和元模型不需要从那些使用 NETCONF 协议的的框架和元模型来镜像。它只需要和 NETCONF 兼容就足够了。我们只需要一个简化的框架和协议来驱动 NETCONF 的 3 个数据存储(候选，运行中，启动)就行了，同时还能隐藏从客户端得到的各种数据存储的复杂性。

我们需要一个简化的交易模型，该模型允许在复杂的语境资源中进行简单的增删改查操作。这代表了 NETCONF 协议的事物能力的受限子集。

应用程序如果需要更复杂的事物能力也许应该考虑 NETCONF 而不是 RESTCONF。下列事物特性不直接被 RESTCONF 支持：

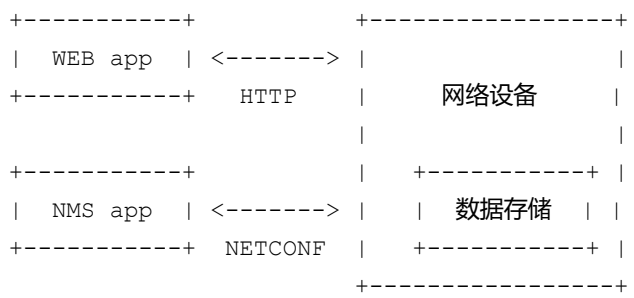
- 数据存储锁定 (全局或者部分)
- 候选数据存储
- 启动数据存储

- 验证操作
- 确认-提交的过程

一台服务器可以将 NETCONF 操作暴露为基于数据模型的操作资源，但那不在本文档的讨论范围中。

RESTful API 没有要取代 NETCONF 的意思，而是提供一种额外的简易接口，遵循 RESTful 原则并且与面向资源的设备抽象兼容。它希望那些需要 NETCONF 完整功能集（比如通知功能）的应用能继续使用 NETCONF。

下图展示了系统组件：




## 1.2 数据模型驱动的 API

RESTCONF 整合了 HTTP 上的 RESTful API 的简单性和结构驱动的可预测性与自动化潜力。

一个使用 HATEOAS 原则的 RESTful 客户端不会使用任何数据建模语言来定义 API 的应用指定内容。客户端可以通过遍历作为 Location ID 返回的 URI 来发现每一个子资源，以此发掘服务器的功用。

此方法关于控制复杂网络设备有 3 大弱点：

- 低效的性能：配置的 API 会很复杂，且需要数不清的协议消息来发现所有的结构信息。一般数据类型信息也要放进协议消息里进行传递，这也是一大浪费。
-

无数据模型丰富性：缺少数据模型，结构级别的语义及有效性约束对应用不可用。

- 

无工具自动化：API 自动化工具需要一些内容结构。这些工具能根据特定的数据模型自动变更编程和文档任务。

诸如 YANG 模块的数据模型模块，将把 "API contract"作为很荣幸的服务器。应用程序设计人员可以编码成数据模型，在重要的细节中预先知道确切地协议操作和一致性服务器实施将支持的数据存贮内容。

一旦客户想使用它，RESTCONF 将提供 YANG 模块支持的服务器性能信息。对于基于 YANG 模块基础之上的定义，客户协议操作和数据存储内容的 URIs 是可以预测的。需要注意的是：对于客户使用来说 YANG 模块和可预测的 URIs 是可以选择的。在功能上，他们完全地不能忽视任何协议的损失。

有使用 CLI 和 SNMP 的运营经验的，表明该操作者了解特定服务或者相关数据的“设置”以及不期待此信息是任意的，并且发现每次客户会打开一个服务器的一个管理会话。

## 1.3 术语

本文档中的关键字"MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL"的解释如BCP 14[RFC2119]描述。

### 1.3.1 NETCONF

以下术语定义在[RFC6241]:

- 

候选配置数据存储

- 

客户端

- 

配置数据

- 

数据存储

- 

配置数据存储

- 

协议操作

- 

运行时配置数据存储

- 

服务器

- 

启动配置数据存储

- 

状态数据

- 

用户

### 1.3.2 HTTP

以下术语定义在[RFC2616]:

- 

实体标签

- 分段
- 标题行
- 消息体
- 方法
- 路径
- 查询
- 请求URI
- 应答体

### 1.3.3 YANG

以下术语定义在[RFC6020]:

- 容器

- 数据节点
- 关键叶子节点
- 叶子节点
- 叶子节点集合
- 集合
- presence容器(或P-容器)
- RPC操作(现称为协议操作)
- non-presence容器 (或NP-容器)
- 系统排序
- 用户排序

[译注：YANG不了解，翻译可能不准确。]

### 1.3.4 术语

本文档会使用以下术语:

- API资源（API resource）：媒体类型为"application/vnd.yang.api+xml"或"application/vnd.yang.api+json"的一种资源。API资源只能被服务器编辑。
- 数据资源（data resource）：媒体类型为"application/vnd.yang.data+xml"或"application/vnd.yang.data+json"的一种资源。数据资源可以被客户和服务器编辑。只有YANG 容器和集合可以是数据资源。顶层YANG终点被当作数据资源的域。
- 数据存储资源（datastore resource）：媒体类型为"application/vnd.yang.datastore+xml"或"application/vnd.yang.datastore+json"的一种资源。数据存储资源只能被服务器编辑。
- 编辑操作（edit operation）：使用POST、PUT、PATCH或DELETE方法作用于数据资源上的一种RESTCONF操作。
- 事件资源（event resource）：媒体类型为"application/vnd.yang.event+xml"或"application/vnd.yang.event+json"的一种资源。它描述了一种由通知消息中递交的概念体系或数据模型特定的事件。
- 域（field）：资源中的一个YANG终端节点。
- 操作（operation）：消息的RESTCONF概念操作，源自HTTP方法、请求URL、报头、和消息体。
-

操作资源（operation resource）：媒体类型为"application/vnd.yang.operation+xml"或"application/vnd.yang.operation+json"的一种资源。

- 补丁（patch）：作用于目标数据存储的一般PATCH操作。消息体内容里的媒体类型指出了使用的补丁类型。
- 简单补丁（plain patch）：媒体类型为"application/vnd.yang.data+xml"或"application/vnd.yang.data+json"的PATCH操作。
- 请求参数（query parameter）： 请求URI中编码在查询部分的参数（和其值）。
- 资源（resource）：表示一个设备中可管理组件的概念对象。包括资源本身和它的所有的域。
- 检索请求（retrieval request）：使用GET或HEAD的操作。
- 目标资源（target resource）： 和特定消息关联的一种资源，由请求URI中的“path”组件定义。
- 统一数据存储（unified datastore）：运行配置的设备的概念表示。服务器将隐藏NETCONF数据存储的所有编辑操作，比如：":candidate"和":startup"能力。
- YANG补丁（YANG Patch）：媒体类型为"application/vnd.yang.patch+xml"或"application/vnd.yang.patch+json"的PATCH操作。
- YANG终端节点（YANG terminal node）：YANG节点表示 叶子节点、叶子节点集合或任何xml定义。

## 1.4 概述

这份文档定义了 RESTCONF 协议，它是一个 RESTful 风格的 API，用于访问概念数据仓库（仓库中保存着由 YANG 语言定义的数据）。RESTCONF 使用 HTTP 方法提供了一个应用框架和元模型。

本文档中定义的一系列 URI，用于访问 RESTCONF 资源。服务器所支持的 YANG 组件集决定了附加数据模型的特定操作和服务器中可用的顶级数据节点资源。

#### 1.4.1 资源 URI 示意图

RESTCONF 资源的 URI 层次结构由一个入口点容器、三个顶级资源和一个域组成。本文档将在第 5 节对每个 URI 进行详细介绍。

```
/.well-known/restconf
   /datastore
      /<top-level-data-nodes> (config=true or false)
   /modules
      /module
         /name
         /revision
         /namespace
         /feature
         /deviation
   /operations
      /<custom protocol operations>
   /version (field)
```

### 1.4.2 RESTCONF 消息示例

本文档中的例子使用标准 YANG 组件（在第 8 节定义）和非标准 YANG 组件（在第 14 节有定义非标 YANG 准组件的例子）。

本节介绍典型的 RESTCONF 消息交换过程。

#### 1.4.2.1 获取顶级 API 资源

获取资源的默认编码是 XML，在默认情况下获取资源时， 任何嵌套资源也会同时得到。客户端从顶级 API 资源开始获取，使用的入口点URI 是“/.well-known/restconf”。

```
1  GET /.well-known/restconf?format=json HTTP/1.1
2  Host: example.com
```

```
Accept: application/vnd.yang.api+json
```

服务器可能会返回如下内容:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/vnd.yang.api+json

{
  "restconf": {
    "datastore" : [ null ],
    "modules": {
      "module": [
        {
          "name" : "example-jukebox",
          "revision" : "2013-09-04",
          "namespace" : "example.com"
        }
      ]
    },
    "operations" : {
       "play" : [ null ]
    },
    "version": "1.0"
  }
}
```

为了能收到 XML 编码的内容，需要在请求头里添加“Accept”字段，例如下面这个请求:

```
GET /.well-known/restconf HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+xml
```

另一个方法是使用 "format" 请求参数，如下例所示:

```
GET /.well-known/restconf?format=xml HTTP/1.1
Host: example.com
```

无论使用哪种方法，服务器都会返回相同的结果, 可能会像下面这样 :

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/vnd.yang.api+xml
```

```
   <restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
     <datastore />
     <modules>
       <module>
         <name>example-jukebox</name>
         <revision>2013-09-04</revision>
         <namespace>example.com</namespace>
       </module>
     </modules>
     <operations>
       <play xmlns="http://example.com/ns/example-jukebox" />
     </operations>
     <version>1.0</version>
   </restconf>
```

关于 GET 方法的详细介绍，请查看第 3.3 节。

#### 1.4.2.2 创建新的数据资源

为了创建一个新的"点唱机（jukebox）"资源，客户端可能会发送这些内容:

```
POST /.well-known/restconf/datastore HTTP/1.1
   Host: example.com
   Content-Type: application/vnd.yang.data+json

   { "example-jukebox:jukebox" : [null] }
```

如果资源创建成功，服务器可能会返回:

```
    HTTP/1.1 201 Created
   Date: Mon, 23 Apr 2012 17:01:00 GMT
   Server: example-server
   Location: http://example.com/.well-known/restconf/datastore/
     example-jukebox:jukebox
   Last-Modified: Mon, 23 Apr 2012 17:01:00 GMT
   ETag: b3a3e673be2
```

为了在"媒体库 (library) "资源里面新建一个“艺术家（artist）” 资源，客户端可能会发出如下的请求。

```
    POST /.well-known/restconf/datastore/example-jukebox:jukebox/
      library HTTP/1.1
   Host: example.com
   Content-Type: application/vnd.yang.data+json

   { "artist" : {
       "name" : "Foo Fighters"
     }
```

```
    }
```

如果资源创建成功，服务器将会返回下面的结果（注意响应头的"Location" 字段里的换行只是出于显示的目的才加上的）：

```
 HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Location: http://example.com/.well-known/restconf/datastore/
     example-jukebox:jukebox/library/artist/Foo%20Fighters
Last-Modified: Mon, 23 Apr 2012 17:02:00 GMT
ETag: b3830f23a4c
```

为了在“点唱机（jukebox）”资源中给这个艺术家创建一个新的“唱片集（album）”，客户端可能会发送下面的请求（注意，请求行的 URI 出现的换行只是出于显示的目的）。

```
 POST /.well-known/restconf/datastore/example-jukebox:jukebox/
   library/artist/Foo%20Fighters  HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json

{
  "album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:Alternative",
    "year" : 2012   // note this is the wrong date
  }
}
```

如果资源创建成功，服务器可能会返回下面的内容（注意，响应头里加入了换行只是出于显示的目的）。

```
 HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:03:00 GMT
Server: example-server
Location: http://example.com/.well-known/restconf/datastore/
  example-jukebox:jukebox/library/artist/Foo%20Fighters/
  album/Wasting%20Light
Last-Modified: Mon, 23 Apr 2012 17:03:00 GMT
ETag: b8389233a4c
```

关于 POST 方法的详细介绍请查看第 3.4 节。

#### 1.4.2.3 替换已存在的数据资源

注意：替换已存在的资源是一个相当极端的操作。打补丁（PATCH ）的方法通常更合适。

在这里替换唱片集子资源只是为了举一个例子。为了替换“唱片集（album）” 资源内容，客户端可能会发送下面的内容（注意请求行里的 URI 出现了换行只是出于显示的目的）：

```
PUT /.well-known/restconf/datastore/example-jukebox:jukebox/
      library/artist/Foo%20Fighters/album/Wasting%20Light   HTTP/1.1
   Host: example.com
   If-Match: b3830f23a4c
   Content-Type: application/vnd.yang.data+json

   {
     "album" : {
       "name" : "Wasting Light",
       "genre" : "example-jukebox:Alternative",
       "year" : 2011
     }
   }
```

如果资源更新成功，服务器可能会返回如下结果：

```
    HTTP/1.1 204 No Content
   Date: Mon, 23 Apr 2012 17:04:00 GMT
   Server: example-server
   Last-Modified: Mon, 23 Apr 2012 17:04:00 GMT
   ETag: b27480aeda4c
```

关于 PUT 方法的详细介绍在第 3.5 节。

#### 1.4.2.4 给已存在的数据资源打补丁

为了仅替换"唱片集（album）"资源中的“年（year）” 域（而不是替换整个资源），客户端可能会发送一个像下面这样的补丁数据。注意请求行中的 URI 出现了换行只是出于显示的目的：

```
    PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
      library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
   Host: example.com
   If-Match: b8389233a4c
   Content-Type: application/vnd.yang.data+json

   { "year" : 2011 }
```

如果更新成功，服务器可能会返回如下结果：

```
    HTTP/1.1 204 No Content
   Date: Mon, 23 Apr 2012 17:49:30 GMT
   Server: example-server
   Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
```

```
    ETag: b2788923da4c
```

同样的请求也可以用 XML 编码的方式发出，比如像下面这样：

```
 PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/vnd.yang.data+xml
```

```
<year xmlns="http://example.com/ns/example-jukebox">2011</year>
```

关于 PATCH 方法的详细介绍，请查看第 3.6 节。

#### 1.4.2.5 删除已存在的数据资源

为了删除像"唱片集（album）"这样的资源，客户端可能会发送以下资源：

```
 DELETE /.well-known/restconf/datastore/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
```

如果资源删除成功，服务器可能会返回以下结果：

```
 HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:40 GMT
Server: example-server
```

关于 DELETE 方法的详细介绍请查看第 3.7 节。

#### 1.4.2.6 删除数据资源中的可选域

DELETE 方法不能用于删除资源中的可选域。只能使用媒体类型为 YANG Patch 的 PATCH 方法删除。

关于 YANG Patch 方法的详细介绍请查看第 7 节。

#### 1.4.2.7 调用数据模型的特定操作

为了使用操作资源（operation resource）调用数据模型特定的操作，需要使用 POST 方法。客户端可能会发送一个“备份数据仓库（backup-datastore）”请求，如下所示：

```
POST /.well-known/restconf/operations/example-ops:backup-datastore
    HTTP/1.1
```

```
Host: example.com
```

服务器可能会这样返回:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:50:00 GMT
Server: example-server
```

关于使用 POST 方法发送操作资源的详细介绍，请查看第 3.9 节。

## 2. 框架

**RESTCONF 协议定义了一个框架，用于实现对配置管理有用的公共 API。本节介绍 RESTCONF 的组成部分。**

## 2.1 消息模型

RESTCONF 协议用 HTTP 实体表示消息。一个 HTTP 消息对应一个单独的协议方法。大部分消息可以在一个单独的资源上执行一个单独的任务，比如获取资源或编辑资源。例外的是 PATCH 方法，如果使用 YANG Patch 格式，利用一个消息可以编辑多个数据仓库。

## 2.2 通知模型

```
[TBD]
```

## 2.3 资源模型

RESTCONF 协议操作的对象是有层次结构的资源，首先处理它的顶级 API 资源。每个资源代表设备中的一个可管理组件。

可以认为资源是由概念数据集合和允许操作该数据的方法集组成的。它可以包含子节点，即嵌套资源或域。子资源的类型和允许操作它们的方法是数据模型相关的。

资源有它自己的媒体类型标识符，用 HTTP 响应头中的“Content-Type”字段表示。一个资源可以包含 0 个或多个嵌套资源。一个资源的创建和删除操作可以独立于它的父资源，如果它的父资源存在的话。

这篇文档定义了所有的 RESTCONF 资源，除了数据仓库内容、协议操作和通知事件。那些资源类型的语法和语义是用 YANG 语句定义的。

### 2.3.1 RESTCONF 资源类型

RESTCONF 协议定义了一些应用相关的媒体类型，用于标识每个可用的资源类型。下表总结了资源和媒体类型的对应关系。

RESTCONF 媒体类型

| 资源 | 媒体类型 |
| --- | --- |
| API | application/vnd.yang.api |
| Datastore | application/vnd.yang.datastore |
| Data | application/vnd.yang.data |
| Event | application/vnd.yang.event |
| Operation | application/vnd.yang.operation |
| Patch | application/vnd.yang.patch |

这些资源将在第5节介绍。

### 2.3.2 资源发现

客户端应该首先获取顶级 API 资源，使用入口点 URI“/.well-known/restconf”。

RESTCONF 协议没有包含资源发现机制。相反，服务器端发布的 YANG 组件定义了这些机制，用于构建可预测操作和数据资源标识符。

当获取子资源时，使用请求参数“depth”可以控制返回子资源的层级深度。这个参数可以和 GET 方法一起使用，用于发现指定资源中的子资源。

关于“depth”参数的详细介绍，请查看第3.8.2节。

## 2.4 数据存储模型：

统一存储(unified 数据存储)用于简化客户端的资源管理。RESTCONF 数据存储是将设备支持的运行时的配置和非配置数据组合到一起。默认的配置数据返回数据存储内容中的 GET 方法。

以 NETCONF 数据存储为基础可以用于实现统一数据存储，但是服务器的设计不能在 NETCONF 中定义限制抽取数据存储程序。

代表(candidate)和启动(startup)数据存储在 RESTCONF 协议中是无效的。事务管理和配置持续数据是在服务器上处理的，而并不是在客户端。

### 2.4.1 内容模型

RESTCONF 协议工作在 YANG 的数据建模语言定义的概念上的数据存储。服务器列出它支持每个YANG 模块，在顶层 API 资源类型中的“/.well-known/restconf/modules/module”资源，使用基于 RFC 6020 中定义的 YANG 模块支持的 URI 格式的结构体。

概念数据存储内容，数据模型的具体操作和通知事件，被这组 YANG 模块资源所标示。所有 RESTCONF 内容（数据资源，操作资源，或事件资源）定义在 YANG 语言中。

作为配置或非配置的数据的 classification，从 YANG“config”语句派生而来。GET 方法获取的数据可以以几种方式过滤，包括“config”参数检索配置或非配置数据。

数据排序的行为, 从 YANG“ordered-by”语句派生而来。YANG 提供 PATCH 操作允许 list 或 leaf-list 字段插入或移动, 以 NETCONF 同样的方式。

服务器不需要保持系统有序数据，以任何特定的持久顺序。服务器应该保持相同的数据排序(系统排序的数据)，直到下一次重新启动服务器或终止服务器。服务器必须保持相同的数据排序(用户排序数据)，直到下一次重新启动服务器或终止服务器。

2.4.2 编辑模型

编辑模型的RESTCONF数据存储是简单的，直接的，其行为类似于 NETCONF之中的":writable-running"性能。

数据存储的每个 RESTCONF编辑资源在成功地完全交易之后将被激活。它是一个实现服务器如何完成RESTCONF编辑请求的特定问题。举个例子来说，仅仅接受通过一个候选人数据存储的服务器就可以在内部编辑数据存储以及自动地完成"commit"操作。

在编辑模型时需要更多控制的应用程序可以考虑使用NETCONF来代替RESTCONF。

2.4.2.1 编辑操作发现

有时候一个服务器不需要完成对每个资源的逐一操作。有时候数据模型请求会导致实现一个编辑操作子集的一个节点。比如，在父资源被创建之后，一个服务器可以不允许一个特定的配置数据节点模块化。

该 OPTIONS方法可以确定一个特定资源的服务器支持 HTTP方法能够被使用。打个比方，如果该服务器允许一个数据资源节点被创建，那么该 POST方法将返回响应。

### 2.4.2.2 编辑冲突检测

RESTCONF 中提供了两种"编辑冲突检测"的机制，分别针对数据存储资源和数据源。

- 时间戳(timestamp): 维护上次修改时间，对于一次检索请求响应头中会返回“Last-Modified”和“Date”两个值。在编辑操作请求时，如果目标资源在特定的时间戳之后已经被修改过，“If-Unmodified-Since”头属性可以被用来令服务端拒绝此次请求。

- 实体标签(entity tag): 维护一个唯一的晦涩难懂的字符串，对于一次检索请求响应头中会返回“ETag”值。在编辑操作请求时，如果目标资源实体标签与特定值不匹配，“If-Match”头属性可以被用来令服务端拒绝此次请求。

注意，服务端仅需要为某个数据存储资源维护这些属性，而不是为些个别的数据资源。

举例:

在此例中, 服务端只支持受委托的数据存储上次修改的时间戳。客户端已经提前得到了“Last-Modified”头，缓存了一些在下面请求中提供的值，此次请求是要替换键值是“11”的list成员值:

```
PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album/Wasting%20Light/year HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+json
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT
Content-Type: application/vnd.yang.data+json

{ "year" : "2011" }
```

在此例中，数据存储资源在“If-Unmodified-Since”头所指定的时间之后已经改变了。服务端响应应如下:

```
HTTP/1.1 304 Not Modified
Date: Mon, 23 Apr 2012 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:45:00 GMT
ETag: b34aed893a4c
```

### 2.4.3 锁模型

RESTCONF 不提供数据存储锁。需要对运行中的配置数据存储内容逐一的做数次改变的应用，在没有其它客户端干扰的情况下，可以考虑使用 NETCONF 协议来替代 RESTCONF 协议。

### 2.4.4 持久模型

每一次对数据存储资源的 RESTCONF 编辑都会被服务端在特定实现的事件中保存为非易失性存储。不能保证配置改变会被立刻保存，也不能保证已经保存的配置一直可以作为运行配置的镜像。

那些需要更多地控制持久模型的应用或许应该考虑使用 NETCONF 协议来替代 RESTCONF 协议。

### 2.4.5 默认模型

NETCONF 的叶子节点有相当复杂的约束处理模型。RESTCONF 试图避免这种复杂性，并限制操作应用到一个资源。

如果目标的 GET 方法（加上“select”的值）是一种数据节点，则代表一个叶子节点，有一个默认值，并且这个叶子节点还没有被赋予一个值，则服务器**必须**在所使用的服务器上返回默认值。

GET 方法只返回存在后代的节点，这是由服务器决定的。没有一种机制是适用于客户端访问服务器内置默认值资源的，这种节点是不存在的，但是还是有服务器使用一些默认值的。（在 NETCONF 中不会检出类似 "with-defaults=report-all" 这样的模式。）

应用程序需要更多的控制，可能会考虑使用 NETCONF 代替 RESTCONF 约束模型。

## 2.5 事务模型

RESTCONF 协议提供扩展的事务框架，允许简化的事务模型，使用普通的其他操作编辑一个资源（和子资源）。它还提供了 YANG 的补丁，这是一个标准的 PATCH 方法的变体。它允许对一个更大的集合进行编辑操作，并可以应用于多个资源。

RESTCONF 没有提供更多更复杂的事务模型，它允许多次编辑被存储在临时的暂存器中，到时一起提交。

需要更多控制事务模型的应用程序可能会考虑使用 NETCONF 代替 RESTCONF。

## 2.6 可扩展模型

RESTCONF 协议被设计成是可扩展的数据存储内容和数据模型的特定的操作协议。它可以添加新的协议操作，且不改变入口点，如果他们是可选的，就不需要改变任何现有的操作。

它为每个 YANG 模块使用单独的名称空间。内容被编码成 XML，并指示模块使用中的“名称空间”，并指示在 YANG 模块中的 URI 值。在 YANG 模块中指定模块使用的模块命名，内容使用 JSON 编码，但这不是必需的，除非 YANG 多个兄弟节点有相同的标识符的名字。模块名称中指定了 JSON 的编码规则[I-D.lhotka-netmod-json]。

## 2.7 版本控制模型

一个资源实例的版本使用一个实体标签来识别，就像 HTTP 定义的那样。本节列出的版本标示符应用于资源模式定义的版本。RESTCONF 协议使用两种模式版本控制信息：

- RESTCONF 协议版本
- 数据和操作资源定义版本

协议版本通过 "well-known" 统一资源标示符入口点"/.well-known/restconf"来识别。如果非向后兼容的变更曾经是需要的，那么这一点将会改变(例如 "/.well-known/restconf2")。不会破坏向后兼容的较小的版本变更将不会导致入口点改变。

客户端可以使用 API "restconf/version"资源来识别服务端实现的 RESTCONF 协议的精确的版本。此值包含完整的 RESTCONF 协议版本。每次重新发布协议规格时，必须重新更新"/.well-known/restconf/version"资源。

为数据或操作定义的资源版本号是一个日期字符串，这个日期是定义资源的YANG模型的修正日期。其他资源类型的资源版本号是一个数字字符串，由"/.well-known/restconf/version" 域定义。

### 2.8 检索过滤模型

---

在RESTCONF协议里针对数据资源的检索有三种类型的过滤器。

- 所有/全无的情况：在请求头里使用一些条件测试机制，检索只能得到两种结果，要么是满足条件然后得到一个完整的“200 OK”响应，要么是不满足条件然后得到一个“304 Not Modified” 状态行。
- 数据分类：请求配置或非配置数据。
- 过滤器：请求目标资源里的所有可能子集节点。

在5.3.2章节有数据检索过滤器的详细描述。

### 2.9 接入控制模型

RESTCONF协议在操作和数据资源之外的内容不提供粒度化的接入控制。NETCONF接入控制模型（NACM)在【RFC6536】中定义。表1显示了RESTCONF和NETCONF操作的特定的对应关系。资源路径需要在服务器内部转换以便于YANG实例标识符保持一致。服务器可以使用这个信息来将NACM接入控制协议规则应用于RESTCONF消息。

服务器绝对不能给任何没有授权的客户端开放接入权限。

## 3. 操作

RESTCONF协议使用HTTP方法来为针对特别资源的请求定义CRUD操作。以下表显示了RESTCONF操作和NETCONF协议操作的对应方式：

表1： RESTCONF中的CRUD方法

| RESTCONF | NETCONF |
|---|---|
| OPTIONS | none |
| HEAD | none |
| GET | <get-config>, <get> |
| POST | <edit-config> (operation="create") |
| PUT | <edit-config> (operation="replace") |
| PATCH | <edit-config> (operation="merge") |
| DELETE | <edit-config> (operation="delete") |

### 3.1 OPTIONS

OPTIONS方法是由客户端发送的，用来确定服务器针对特定资源支持哪种方法。它支持所有的多媒体类型。请注意这个方法的实现是HTTP的一部分，这个章节不会引入任何额外的要求。

请求必须包含一个至少包含了输入点组件的URI请求。

服务器会返回包含“204 No Conten”的“Status-Line”头，并且在响应中包含一个“Allow”头。这个头将会根据目标多媒体资源类型来填值。响应中可能也会其他头。

例子1：

客户端可能请求一种方法，支持名叫“library”的数据资源：

```
OPTIONS /.well-known/restconf/datastore/example-jukebox:jukebox/
  library/artist   HTTP/1.1
Host: example.com
```

服务器应该响应（如果config=true）：

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH,DELETE
```

例子2：

客户端可能请求一种方法，支持在“system”资源里的非配置类“counters”资源：

```
OPTIONS /.well-known/restconf/datastore/example-system:system/
   counters HTTP/1.1
Host: example.com
```

服务器应该响应：

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Allow: OPTIONS,HEAD,GET
```

例子3：

客户端可能请求一种方法，支持“play”资源：

```
OPTIONS /.well-known/restconf/operations/play HTTP/1.1
Host: example.com
```

服务器应该响应：

```
   HTTP/1.1 204 No Content
   Date: Mon, 23 Apr 2012 17:02:00 GMT
   Server: example-server
   Allow: POST
```

Head 方法是由客户发送的，一种只需要检索 headers 就可以获得可比较的 get 方法，而不需要理会响应主体的方法。它支持所有的资源类型，除了操作资源。

MUST 请求包含一个 URI 请求，这个 URI 则必须包含至少一个组件入口点。HEAD 方法所支持的 GET 方法支持相同查询参数。例如，查询参数中的“select”就可以用来指定目标资源里的嵌套资源。

访问控制的行为是强制性的，就像 GET 方法替代 HEAD 方法。除了不包含相应主体这一部分，MUST 服务器的响应就和 GET 方法取代 HEAD 方法一样。

例子：

客户端可能发出“library”资源 JSON 代表的头部响应请求：

```
 HEAD /.well-known/restconf/datastore/example-jukebox:jukebox/
       library HTTP/1.1
   Host: example.com
   Accept: application/vnd.yang.data+json
```

服务器可能回应：

```
HTTP/1.1 200 OK
   Date: Mon, 23 Apr 2012 17:02:40 GMT
   Server: example-server
   Content-Type: application/vnd.yang.data+json
   Cache-Control: no-cache
   Pragma: no-cache
   ETag: a74eefc993a2b
   Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT
```

## 3.3 GET

GET 方法由客户端发出，来从资源那里获取数据和元数据。除了操作资源之外的所有资源类型都支持该方法。请求**必须**包含一个至少包含了一个端点组件的请求URL。

如下是得到了GET方法支持的查询参数：

GET 查询参数

| 名称 | 章节 | 描述 |
| --- | --- | --- |
| config | 3.8.1 | 请求的是配置的还是非配置的数据 |
| depth | 3.8.2 | 控制获取请求的深度 |
| format | 3.8.3 | 对请求的响应是JSON还是XML内容的 |
| select | 3.8.6 | 指定目标资源内部的一个嵌套资源 |

服务端**肯定不会**返回那些用户并没有读取权限的数据资源。

如果用户没有被授权读取目标资源的任何部分，就会有一个包含了“403 Forbidden”状态行的错误响应信息被返回到客户端。

如果用户被授权读取某部分而不是全部的目标的资源，未被授权的内容就会从响应消息体中省略, 而得到授权的内容则会返回给客户端。

示例:

客户端请求JSON形式的“library”资源，消息头如下:

```
GET /.well-known/restconf/datastore/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album?format=json HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+json
```

服务端的响应会如下:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:40 GMT
Server: example-server
Content-Type: application/vnd.yang.data+json
Cache-Control: no-cache
Pragma: no-cache
ETag: a74eefc993a2b
Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT

{
  "album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:Alternative",
    "year" : 2011
  }
}
```

## 3.4 POST

POST 方法会由客户端以各种不同的原因发出。服务端使用目标资源的媒体类型来决定如何处理请求。

请求**必须**包含一个请求URI，它包含了一个目标资源，标识如下资源类型中的一种:

支持POST的资源类型

| 类型 | 描述 |
| --- | --- |
| Datastore | 创建一个顶层配置的数据资源 |
| Data | 创建一个配置的数据子资源 |
| Operation | 调用协议操作 |

### 3.4.1 创建资源模式

如果目标资源的类型是一个数据存储或数据资源，那么POST是适合这种请求来创建这样一个资源或子资源的。

下面的查询参数是用来支持POST方法来处理数据存储和数据资源。他们只用于列出用户要求的YANG数据节点。

POST查询参数

| 名称 | 节 | 描述 |
| --- | --- | --- |
| insert | 3.8.4 | 指定在何处插入资源 |
| point | 3.8.5 | 指定插入资源的点 |

如果POST方法成功，状态栏返回"204 No Content"且不会有消息体响应。

如果用户是未经授权创建的目标资源，状态栏会给客户端返回错误的响应“403 Forbidden”。其他错误的处理定义可以查看第6节。

### 3.4.2 调用操作模式

如果目标资源类型是一个操作资源，然后 POST 方法被当作一个请求来调用该操作。消息体（如果有的话）作为处理操作的输入参数。详细的操作资源请参考 5.4 节。

如果 POST 方法成功，如果状态行有返回响应消息体”200 OK”，如果没有响应消息体在状态行会返回"204 No Content"。

如果用户未被授权调用目标操作，会在状态行返回"403 Forbidden"给客户端。所有其他错误响应处理根据第 6 节中的定义。

## 3.5 PUT

PUT 方法由客户端发送替换目标资源。

请求必须包含一个请求 URI，并包含目标资源所标识要替换的资源。

如果 PUT 方法成功，会在状态行返回"200 OK"， 它不会返回消息体。

如果用户没有被认证就替换目标资源，就会返回一个包含"403 Forbidden"的回应状态行给客户端。所有其他的错误处理依据第 6 部分的程序定义。

## 3.6 PATCH

PATCH 方法使用 HTTP 的 PATCH 方法，它由 [RFC5789] 定义，它给资源的 patch 机制提供了一个可扩展的框架。每一个 patch 类型都需要一个唯一的媒体类型。大多数的 patch 类型都能得到服务器的支持。还有两种强制性的 patch 类型是服务器必须要实现的：

- plain patch type: 如果指定的媒体类型是 “application/vnd.yang.data” ，那么 PATCH 方法对目标资源来说是一个简单的合并操作。消息体是包含XML或者JSON编码的资源内容，这些内容可以与目标资源合并。

- YANG Patch type: 如果指定的媒体类型是 "application/vnd.yang.patch" ，然后 PATCH 方法是一个 YANG Patch 格式的编辑列表（查看第7部分）。在 ‘ietf-restconf’ 的 YANG 模块中，消息体包含由 XML 或 JSON 编码的指定 ‘patch’ 容器实例（查看第8部分）。

PATCH方法必须被用来创建或者删除存在的资源或子资源的可选段。

如果PATCH方法成功，状态栏会返回一个“200 OK”，没有任何消息体。

如果用户没有被认证就更新目标资源，服务器就会在状态栏返回一个错误响应包含一个“403 Forbidden”给客户端。所有其他的错误响应处理定义详见第6节。

## 3.7 DELETE

DELETE方法用于删除目标资源。

如果DELETE方法成功，就会在状态栏返回“200 OK”，不会返回消息体。

如果用户没被认证就删除目标资源，服务器就会在状态栏返回一个错误响应包含一个“403Forbidden”给客户端。所有其他的错误响应处理定义详见第6节。

## 3.8 查询参数

每一个 RESTCONF 操作都允许在请求URI中有零到多个查询参数。第三节详细描述了每个操作的定义中用到的查询参数。

查询参数的顺序任意。每个参数可以出现零或者一次。如果参数是没有的就可能会应用到一个默认值。

这一节定义了所有的 RESTCONF 查询参数。

### 3.8.1 "config" 参数

"config" 参数用来指定请求的是否是配置的或者非配置的数据。

这个参数只支持GET和HEAD方法。它也只是在目标资源是一个数据资源的时候被支持。

```
    syntax: config= true | false
    default: true
```

示例:

这个由客户端发起的请求只会获取存在于“library”资源中的非配置数据节点。

```
GET /.well-known/restconf/datastore/example-jukebox:jukebox/
  library?config=false   HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+json
```

服务端可能会像下面这样响应:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/vnd.yang.data+json
```

```
07
08    {
09      "library" : {
10         "artist-count" : 42,
11         "album-count" : 59,
12         "song-count" : 374
13      }
14    }
```

### 3.8.2 "depth" 参数

"depth" 参数备用指定GET方法的响应中内嵌层级的数量。内嵌层级包含了目标资源以及被包含在目标资源内部的子节点。

起始层级别取决于针对该操作的目标资源。

```
1       syntax: depth=<range: 1..max> | unbounded
2       default: unbounded
```

示例:

下面这个示例操作会获取存在于顶层“jukebox”资源中的两层配置数据节点。

```
1     GET /.well-known/restconf/datastore/example-jukebox:jukebox
2         ?depth=2 HTTP/1.1
3     Host: example.com
4     Accept: application/vnd.yang.data+json
```

服务端也许会像下面这样响应:

```
01    HTTP/1.1 200 OK
02    Date: Mon, 23 Apr 2012 17:11:30 GMT
03    Server: example-server
04    Cache-Control: no-cache
05    Pragma: no-cache
06    Content-Type: application/vnd.yang.data+json
07
08    {
09      "jukebox" : {
10       "library" : {
11         "artist" : {
12           "name" : "Foo Fighters"
13         }
14       },
15       "player" : {
16         "gap" : 0.5
17       }
```

```
    }
  }
```

服务端默认会在被获取的资源中包含所有的子树, 它们都是相同的资源类型。只有跟目标资源的媒体类型不同的一个层级的子资源会被返回。

例如，如果客户端要获取的是"application/vnd.yang.api"资源类型, 那么数据存储资源的节点会作为一个空节点被返回, 因为它所有的子节点都是数据资源。在这一场景中数据存储的整个内容是不会被返回的。操作资源也会作为空节点被返回(例如 "play" 操作)。

请求URL:

```
GET /.well-known/restconf HTTP/1.1
```

响应:

```
{
  "restconf": {
    "datastore" : [ null ],
    "modules": {
      "module": [
        {
          "name" : "example-jukebox",
          "revision" : "2013-09-04",
          "namespace" : "example.com"
        }
      ]
    },
    "operations" : {
       "play" : [ null ]
    },
    "version": "1.0"
  }
}
```

### 3.8.3 "format" 参数

"format" 参数是被用来指定响应返回的文本格式。请注意，这个参数可以用来代替“Accept”头来识别响应中所需的格式。

"format" 参数仅仅支持GET和HEAD方法。它支持所有的RESTCONF媒体类型。

```
syntax: format= xml | json
default: Accept header, then xml
```

如果 "format" 参数出现，然后覆盖Accept头。如果没有Accept标头或“格式”参数出现，默认就是XML。

举例来说：

```
GET /.well-known/restconf/datastore/example-routing:routing
    HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+json
```

这个示例请求只能检索配置数据节点存在于顶级“路由”资源，并在JSON编码中检索它们。

```
GET /.well-known/restconf/datastore/example-routing:routing
    ?format=json HTTP/1.1
Host: example.com
```

这个示例请求只能检索配置数据节点存在于顶级“路由”资源，并在JSON编码中检索它们。

### 3.8.4 "insert" 参数

"insert" 参数被用来指定一个资源应该如何被插入到一个用户指定了排序的列表中。

这个参数只被POST方法支持。它也只会在资源是数据资源，并且那个数据是以由用户而不是系统进行排序的YANG列表来呈现时，才会被支持。

如果"before" 或者"after"这两个值被用到的话，那么用于插入参数的"point"参数也**必须**应该有。

```
  syntax: insert= first | last | before | after
  default: last
```

示例:

```
Request from client:

  PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
    playlist/Foo-One?insert=first HTTP/1.1
  Host: example.com
  Content-Type: application/vnd.yang.data+json

  {
    "song" : {
      "index" : 1,
      "id" : "/example-jukebox:jukebox/library/artist/
          Foo%20Fighters/album/Wasting%20Light/song/Rope"
     }
  }

Response from server:

```

```
   HTTP/1.1 201 Created
   Date: Mon, 23 Apr 2012 13:01:20 GMT
   Server: example-server
   Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
   Location: http://example.com/.well-known/restconf/datastore/
      example-jukebox:jukebox/playlist/Foo-One/song/1
   ETag: eeeada438af
```

### 3.8.5 "point" 参数

"point" 参数被用来指定将要被创建或者从一个用户排序的列表中移除的数据资源的插入点。除非 "insert" 查询参数也存在， 并且其取值是"before" 或者 "after"，否则它就会被忽略掉。

这个参数包含了将要被用来作为一个POST方法的插入点的资源的实体标识。它是以章节5.3.1中定义的规则进行编码的。这个参数没有默认值。

```
   syntax: point= <instance-identifier of insertion point node>
```

示例：

在这个示例中，客户要在一个 "album" 资源中插入一首新的歌曲，在另一首歌曲的后面。为了好展示，请求的URI被分段了。

客户端的请求：

```
  POST /.well-known/restconf/datastore/example-jukebox:jukebox/
     library/artist/Foo%20Fighters/album/Wasting%20Light?insert=after
     &point=/.well-known/restconf/datastore/example-jukebox:jukebox/
     library/artist/Foo%20Fighters/album/Wasting%20Light/song/
     Bridge%20Burning   HTTP/1.1
  Host: example.com
  Content-Type: application/vnd.yang.data+json

  {
    "song" : {
       "name" : "Rope",
       "location" : "/media/rope.mp3",
       "format" : "MP3",
       "length" : 259
    }
  }
```

服务端的响应：

```
  HTTP/1.1 204 No Content
  Date: Mon, 23 Apr 2012 13:01:20 GMT
  Server: example-server
  Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
```

```
ETag: abcada438af
```

### 3.8.6 "select" 参数

"select" 查询参数被用来指定一个表达式，它能表示目标资源中所有数据节点的一个子集。它包含了一个相对路径表达式，使用目标资源作为上下文节点。

它被除了操作型资源之外的所有资源类型所支持。内容会根据章节5.3.1中定义的“api-select”规则来进行编码。这个参数只被允许用于GET和HEAD方法。

[FIXME(表示待完善): select字符串的语法仍然待定; XPath, 模式标识符（schema-identifier），正则表达式, 某些其他的东西; 也许是添加一个用于XPath的参数"xselect"，而这个参数就会被限制为一个路径表达式。]

在这个示例中客户端会从服务端获取到JSON格式的API版本信息:

```
GET /.well-known/restconf?select=version&format=json HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+json
```

服务端也许会像下面这样响应。

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/vnd.yang.api+json

{ "version": "1.0" }
```

## 3.9 协议操作

RESTCONF 协议允许使用POST方法调用一些数据模型指定的操作。媒体类型"application/vnd.yang.operation+xml" 或者"application/vnd.yang.operation+json" 必须被用在消息头的 "Content-Type" 一行中。

数据模型指定的操作时受到支持的。这些操作语法和语义跟这些操作的YANG“rpc”语句定义精确对应。

可以参考章节 5.4 详细了解关于操作资源的内容。

### 4. 消息

这一节要描述的是在RESTCONF协议中使用的消息。

## 4.1 请求URI结构

用URI表示的资源遵循 [RFC3986] 中面向通用URI的结构。

RESTCONF 操作时从 HTTP 方法和请求URI那里衍生出来的，使用的是如下所述的概念域:

```
    <OP> /.well-known/restconf/<path>?<query>#<fragment>
```

```
     ^              ^             ^        ^         ^
     |              |             |        |         |
    方法            入口          资源      查询      片段

     M              M            O        O         I

   M=必须有的（mandatory）, O=可选的（optional）, I=被忽略（ignored）
```

```
    <文本> 部分可以让客户端用真正的值替换
```

- 方法: HTTP 方法标识由客户端请求的RESTCONF操作, 作用于在请求URI指定的目标资源之上。RESTCONF 操作的细节在章节3中有描述。

- 入口: 众所周知的RESTCONF入口端口("/.well-known/restconf")。

- 资源: 标识将要被操作访问的资源的路径表达式。如果这个域没有被提供，那么目标资源就是API本身，以 "application/vnd.yang.api" 为媒体类型提供出来。

- 查询：跟RESTCONF消息相关联的参数集合。这些都有“名称=值”配对的这种熟悉的形式。规范中定义了一个特别指定参数集合，尽管服务端**可能**会选择支持一些没有在本文档中定义的额外的参数。

- 片段：这个域并不会在RESTCONF协议中被使用到。

客户端**应该不会**为一个资源假定好最终的URI的结构。而是现有的资源可以被GET方法找到。当新的资源被客户端创建时，就会有一个"Location"消息头被返回，它标识了新创建的资源的路径。客户端**必须**在资源被创建好以后使用精确的路径标识符来访问资源。

操作的"目标"是资源。而请求URI中“path”域就表明了用于操作的目标资源。

## 4.2 消息头

有几个HTTP消息头在RESTCONF 消息中使用，RESTCONF 消息使用的消息头不只局限于在这个章节罗列的HTTP消息头。

HTTP定义了特定环境下需要使用的消息头，以第三章节涉及的每个操作定义为例来说明如何使用特定的消息头。

这里有在RESTCONF中使用的请求消息头，它们通常应用于数据来源(即请求方)。下表总结了在RESTCONF 消息中常用的请求消息头：

RESTCONF 请求消息头

| 名称 | 描述 |
|---|---|
| Accept | 可接受的响应内容类型（例如：text/html） |
| Content-Type | 请求消息体的类型（例如：application/json） |
| Host | 服务主机地址 |
| If-Match | 如果实体与ETag匹配才生效 |
| If-Modified-Since | 如果在特定时间（例如：上一次最后修改时间）后的行为为修改才起作用 |
| If-Unmodified-Since | 如果在特定时间（例如：上一次最后修改时间）后的行为不为修改才起作用 |

下表总结了在RESTCONF 消息中常用的响应消息头：

RESTCONF 响应消息头

| 名称 | 描述 |
|---|---|
| Allow | 定义了在返回405错误时合法的行为（例如：GET） |
| Content-Type | 响应消息体的类型（例如：application/json） |
| Date | 发送消息的时间，包含日期和时分秒. |
| ETag | 一个标示符号用于指定资源的特定版本. |
| Last-Modified | 某一资源的最后修改时间，包含日期和时分秒. |
| Location | 一个资源标示符用于指定一个新创建的资源. |

## 4.3 报文编码

RESTCONF 报文在 HTTP 中的编码是依据 RFC 2616。"utf-8" 字符集适用于所有的报文。RESTCONF 报文文本是在 HTTP 消息体中被发送的。

内容是通过 JSON 或者 XML 格式被编码的。

XML编码规则的数据节点是在【RFC6020】中定义的。相同的编码规则适用于所有的 XML 内容。

JSON 编码规则在【I-D.lhotka-netmod-json】中被定义。普通的 JSON 不能被使用，因为有特殊的编码规则被用来处理多种模块的命名空间，还要提供一致的数据类型处理。

请求输入文本编码格式被定义成 Content-Type 头。如果存在一个报文体要发送，这个字段就必须出现。

响应输出内容编码格式是标识 Accept 头的，查询参数的“格式”，如果没有指定，就使用请求输入的编码格式。如果没有输入请求，然后默认输出的编码是 XML。文件扩展编码在请求中不被用来识别格式编码。

## 4.4 RESTCONF 元数据

RESTCONF 协议需要检出 NETCONF 协议中使用的一致的元数据。信息默认的页，最近修改的时间戳，等等。一般用于数据存储内容的注释表示。元数据没有在 YANG 模式中定义，因为它适用于存储，这在所有数据节点中是很常见的。

这些信息编码的属性是 XML 编码的，而 JSON 不是一个标准的附加 non-schema 定义的元数据资源或字段的方式。

### 4.4.1 以JSON方式编码RESTCONF元数据

YANG到JSON的映射依据[I-D.lhotka-netmod-json]编码规则，该方式不支持属性。因为YANG不支持在数据节点中定义元数据。本章将详细介绍如何将RESTCONF元数据按JSON方式编码。

只支持简单的元数据：

- 在一个特定的数据节点中一个元数据实例只能出现0次或1次
- 根据[I-D.lhotka-netmod-json]编码规则,一个元数据实例与一个资源相关的编码就好像一个YANG叶子类型的字符串，除了以 "@" (%40) 字符开头的标示符。

- 

一个元数据实例关联到一个资源内的域的编码，就好像它是一个容器的元数据的值和容器内域的值的原生编码。这种编译方式根据[I-D.lhotka-netmod-json]原则，除了以 "@" (%40)开头的元数据外。由容器的名称与容器的值组成的键值对将会在这个容器中重复出现，这个容器也包含域的名称与实际值的键值对。

示例:

```
Meta-data:

  enabled=<boolean>
  owner=<owner-name>

YANG example:

  container top {
    leaf A {
      type int32;
    }
    leaf B {
      type boolean;
    }
  }
```

客户端将检索"top"数据资源(见下图)，服务器包括元数据的数据存储。注意在RESTCONF中不提供一个请求的参数来请求或者废弃元数据。

```
GET /.well-known/restconf/datastore/example:top HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+json
```

服务器端可能的返回如下:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/vnd.yang.api+json

{
  "top": {
    "@enabled" : "true",
    "@owner" : "fred",
    "A" : {
      "@enabled" : "true",
      "A" : 42
    },
    "B" : {
      "@enabled" : "false",
```

```
          "B" : true
        }
      }
    }
```

### 4.5 返回状态

每个消息代表着一些资源访问，一个 HTTP "状态码"为每一个请求返回状态行。如果返回一个4XX 或者5XX范围的状态码作为状态行，则错误信息将会在response中以在6.1章节定义的格式返回。

### 4.6 消息缓存

由于数据存储的内容存在不可预测的时间变化，RESTCONF 服务器的响应通常不会被缓存。

对每个回应来说，服务器应该包括一个“缓存-控制”头，它指定是否应该缓存响应。 “Pragma”头可以指定“不-缓存”也可以指定不支持“缓存-控制”头。

替代使用 HTTP 缓存的，是客户端应该要追踪“ETag”和 / 或“最后-修改”头返回的服务器数据存储资源（或数据资源-如果服务器支持的话）。检索请求一个资源可以包括标题，如“如果没有匹配（If-None-Match）”或“如果修改是因为（If-Modified-Since） “这会让服务器返回一个“304”错误，即如果资源没有改变就不可修改状态栏。如果这个目标资源的元数据被维护着，客户端可以使用的方法来检索消息头，这应该包括“ETag”和“最后-修改”头。

## 5. 资源

RESTCONF协议里使用的资源通过请求URI里的路径来标识。每一个操作都在一个目标资源上执行。

### 5.1 API资源 (/.well-known/restconf)

API资源包含RESTCONF特性的状态和接入点。

它是顶层资源，并且拥有"application/vnd.yang.api+xml"或者"application/vnd.yang.api+json"的介质类型。它可以通过著名的相对URI"/.well-known/restconf"来获取。

这里需要赋值一个强制的字段“version”来标识服务端实现RESTCONF协议的特定版本:

-

每一次这个字段被检索，必须返回相同的服务端响应。

- 

它由服务启动时指派。

- 

服务端必须返回“1.0”作为RESTCONF协议这一次的版本。

- 

这个字段使用第八章提到的“version”叶子定义通过枚举数据类型规则加密。

这个资源拥有以下子资源：

RESTCONF 资源

| 子资源 | 描述 |
| --- | --- |
| datastore | 指向"datastore" 资源 |
| modules | YANG模型 能力URIs |
| operations | 数据 - 模型 特定操作 |

### 5.1.1 /.well-known/restconf/datastore

这个强制资源代表了运行中的配置数据存储和任何可用的非配置数据，它可能被直接获取或着编辑。它不能被客户端创建或着删除。这个资源类型在 5 . 2 节定义。

### 5.1.2 /.well-known/restconf/modules

这种强制性资源的标识符包含 YANG 数据模型模块的支持。

服务器必须保持这个资源的最近修改时间戳，当资源被检出 GET 或 HEAD 方法时，会返回“最近修改”头。

这个资源的服务器应保持一个实体标记，当这种资源被检出 GET 或 HEAD 方法时返回“ETag”头。

#### 5.1.2.1 /.well-known/restconf/modules/module

这个强制性的资源包含一个 URI 字符串，为每个 YANG 数据模型模块提供支持。每个 YANG 模块资源是通过一个资源或数据资源的操作为每一个模块提供实例。

“模块”的内容列表被定义在第 8 节“模块”的 YANG 列表语句中。

服务端也许会为这个资源的每个实例维护一个最后修改时间戳。当 GET or HEAD 方法检索这个资源时将跟随该资源实例一并返回。如果该资源不支持最后修改时间戳，将使用其父模块资源的最后修改时间戳代替。

服务端也许会为这个资源的每个实例维护一个实例标示，当GET or HEAD方法检索这个资源时将根据资源实例返回一个"ETag"消息头，该消息头包含实例标示，如果该资源不支持ETag，将使用其父模块资源的最后修改时间戳代替。

该资源将依据[RFC6991]中的"ietf-iana-types" 模块从"URI"中派生出来的类型进行编码。

此资源有额外的编码要求。这个URI必须遵守在[RFC6020]的5.6.4节中定义的YANG模块的格式要求。

#### 5.1.2.2 检索示例

在这个示例中，客户端将从服务端检索“modules”资源并以JSON格式化该资源。GET请求如下：

```
GET /.well-known/restconf/modules&format=json HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+json
```

服务端将返回如下信息:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/vnd.yang.api+json

{
  "modules": {
    "module": [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ]
      },
```

```
        {
          "name" : "foo-types",
          "revision" : "2012-01-05",
          "namespace" : "http://example.com/ns/foo-types"
        },
        {
          "name" : "bar",
          "revision" : "2012-11-05",
          "namespace" : "http://example.com/ns/bar",
          "feature" : [ "bar-ext" ]
        }
      ]
    }
  }
```

### 5.1.3 /.well-known/restconf/operations

这个可选资源提供服务端支持的数据模型的特定协议操作。如果服务端没有被公布的数据模型的特定操作,服务端将缺省这个资源。

任何数据模型的特定操作定义在YANG模块中，将服务端公布作为这个资源的可用子节点。

## 5.2 数据存储资源

数据存储资源代表数据资源概念树的根。

服务器必须保持这个资源为最新修改的时间戳，并且当这个资源被检索出 GET 或者 HEAD 方法时，返回“最近修改”的时间戳。更改配置数据资源仅对这个时间戳的数据存储有影响。

服务器应该保持这个资源的实体标记，当这个资源被检出 GET 或者 HEAD 方法时返回 "ETag" 头。 如果更改任何配置数据存储中的数据资源，资源实体标记在更改之前，要变更到一个新的未使用的值。

数据存储资源可以使用 GET 方法检索，获取配置数据资源或无配置数据资源存储中的数据资源。 “配置”查询参数被用来在他们之间选择。更多细节详见3.8.1节。

通过“深度”查询参数，返回子树的深度检索操作可以被控制。大量的嵌套层，开始于目标资源，也可以被指定，返回的数量没有限制。更多细节详见3.8.2节。

【修正：不清楚是否顶层的 YANG 数据节点必须被放在列表容器中。

一个数据存储资源通过 PATCH 方法直接改写。存储资源内的配置数据资源可以通过所有方法被直接编辑。】

## 5.3 数据资源

一个数据资源代表一个 YANG 数据节点，它是一个数据存储资源的子孙节点。YANG 容器和列表数据节点类型被认为代表数据资源。其他 YANG 数据节点被认为是在父域的资源。

为了配置数据资源，服务器维持一个最近被修改过的时间戳，并在检出 GET 或者 HEAD 方法时返回“最近被修改”的头。

同样是为了配置数据资源，服务器会为资源维持一个资源实体标签，并在检出作为目标资源的方法是 GET 或 HEAD 方法时返回 "ETag" 头。如果资源被变更或者任何配置资源内的资源被变更， 资源的实体标记将被变更成一个新的以前未使用过的值。

一个数据资源可以使用 GET 方法检索来获取配置数据资源或无配置目标资源中的数据资源。“配置”查询参数用于在二者之间做出选择。更多细节请参考 3.8.1 节。

返回的子树深度检索操作可以控制查询参数的“深度”。大量的嵌套层级，开始于目标资源，并可以被指定，也可以无限制数量地返回。更多细节请参考 3.8.2 节。

根据目标资源和特定的操作，客户机配置的数据资源可以被编辑操作所更改。更多细节请参考第三节编辑操作。

### 5.3.1 在请求 URI 中 YANG 编码的实例标识符

在 YANG 中，从根节点到目标资源，数据节点被命名为一个绝对的 XPath 表达式。在 RESTCONF 中，使用友好的 URL 表达式来代替。

YANG 的 “实例标识符（instance-identifier）” （i-i）数据类型反映在 RESTCONF 上，是在节中定义的路径表达式。

RESTCONF 的实例标识符（instance-identifier）类型转换

| 名称 | 注释 |
| --- | --- |
| 点 | 插入点总是一个满的i-i |
| 路径 | 请求的URI路径是完全或部分的i-i |

“路径” 组件请求 URI 包含绝对路径表达式来标识目标资源。在检索目标资源 GET 方法中，“选择（select）” 查询参数用于选择性地识别请求中的数据节点。

一个可预测的位置的数据资源是很重要的，因为应用程序代码的 YANG 数据模型模块使用静态命名并对所有数据节点定义了一个绝对路径位置。

一个 RESTCONF 的数据资源标识符不是一个 XPath 表达式。它的编码是从左到右的，并从顶层开始的，“api-path”的规则在 5.3.1.1 节有介绍。每一个祖先节点的名字目标资源节点都是有序编码开始的，并结束于节点名称的目标资源。

如果“选择（select）”是当前的，它被编码的时候，就开始于一个目标资源的子节点，这个"api-select"规则在 5.3.1.1 节中定义。

如果一个数据节点的路径表达式是一个 YANG 列表节点，键值列表(如果有的话)的编码是根据“键-值”规则。如果目标资源是列表节点，从操作上来说，这个键值可以被省略。举个例子，POST 方法创建一个新的数据资源列表节点，不允许键值出现在请求 URI 中。

YANG列表中所展示的数据源的叶子键的值必须像下面这样进行编码：

- ”key“语句中确认的每个叶子的值都是按顺序进行编码的。
- “key”语句中的所有组件都必须进行编码。局部的实体标识是不支持的。
- 每个值都使用 5.3.1.1节的“key-value”规则进行编码，要依据对应于叶子键的数据类型的编码规则来进行。
- 空字符串可以是一个有效的键值(例如： "/top/list/key1//key3").
- "/" 字符必须进行URL编码(例如: "%2F").
- 所有的空格都必须进行URL编码。
- "null" 值是不被接受的，因为"empty"数据类型不能被用于作为叶子键。
- XML 编码是在 [RFC6020] 中被定义的。
- JSON 编码是在 [I-D.lhotka-netmod-json] 中被定义的。

- 依据[RFC3896]中定义的规则，整个键值对"key-value"都必须进行常规的URL编码。

- 即使在资源被创建时并没有冲突的本地名称，资源的URI值也会在Location消息头中返回，用于模块名称的数据源 SHOULD 的确认。 这样就确保了即使服务端加载了一个老客户端并不知道的新的模块，正确的资源也会得到确认。

示例:

```
 [ lines wrapped for display purposes only ]

/.well-known/restconf/datastore/example-jukebox:jukebox/library/
   artist/Beatles&select=album

/.well-known/restconf/datastore/example-list:newlist/
   17&select=nextlist/22/44/acme-list-ext:ext-leaf

/.well-known/restconf/datastore/example-list:somelist/
   fred%20and%20wilma

/.well-known/restconf/datastore/example-list:somelist/
   fred%20and%20wilma/address
```

#### 5.3.1.1 数据源标识的 ABNF

如下的 ABNF 语法被用来构造 RESTCONF 路径标识:

```
  api-path = "/" api-identifier
                 0*("/" (api-identifier | key-value ))

  [FIXME: the syntax for the select string is still TBD]
  api-select = api-identifier
                 0*("/" (api-identifier | key-value ))

  api-identifier = [module-name ":"] identifier

  module-name = identifier

  key-value = string

  ;; An identifier MUST NOT start with
  ;; (('X'|'x') ('M'|'m') ('L'|'l'))
  identifier  = (ALPHA / "_")
                *(ALPHA / DIGIT / "_" / "-" / ".")
```

```
18
19      string = <an unquoted string>
```

#### 5.3.2 检索数据资源

检索数据资源有三种过滤方式。本章将介绍每种模式的定义。

##### 5.3.2.1 条件检索

HTTP的Headers(例如："If-Modified-Since" 和"If-Match")能够用于GET方式的请求中，用来校验一个服务端条件的状态。例如被请求的数据资源的最后修改时间，或者该目标资源的实体标示。

如果满足在header中定义的条件，将返回一个 "200 OK" 的状态行并在response中返回请求的数据资源。 如果不满足在header中定义的条件，将返回一个 "304 Not Modified" 状态行代替返回信息。

##### 5.3.2.2 数据分类检索

"config" 请求参数能够用于GET方式的请求中，用来指定配置或者非配置数据 。更多关于“config”请求参数的详细信息请参照3.8.1章节。

##### 5.3.2.3 过滤检索

"select" 请求参数用来指定一个过滤器， 应用于获取目标资源的一个子集，该子集包含的所有满足过滤器条件的目标资源的子节点。

"select"参数字符串的格式请参照3.8.6章节。 筛选器表达式用于选择一组节点，它的应用范围为被目标资源确定的每个上下文节点。

### 5.4 资源操作

一个资源操作通过 YANG 来定义 "rpc" 的操作。

所有的资源操作共享相同的模块命名空间，且模块命名空间作为任何顶级数据的资源，因此，在同一个模块中定义的资源，其操作资源的名称不能与顶级的名字冲突。

如果 2 个不同的 YANG 模块定义成一样的“rpc”标识符，那么模块的名称必须在请求 URI 中使用。举例来说，如果“模块 A”和“模块 B”都被定义成“重置”操作，那么调用“模块 A”应该写成下面的样子：

```
1   POST /.well-known/restconf/operations/module-A:reset HTTP/1.1
2   Server example.com
```

任何同一模块同名资源操作的用法，都是调用一致的“rpc”定义语句。这种行为可以被用来设计协议操作，并在不同的资源类型内执行相同的通用函数。

如果一个“rpc”请求中包含一个“input”部分，则从客户端发出的请求中允许包含一个消息体，否则从客户端发出的请求中禁止包含一个消息体。如果一个“rpc”请求中包含一个“output”部分，则从服务端返回的响应中允许包含一个消息体，否则从服务端返回的响应中禁止包含一个消息体，并且必须发送一个“204 No Content”状态行代替消息体.

### 5.4.1 输入参数编码操作

如果一个“rpc”请求中包含一个“input”部分, 则消息体会提供一个“input”节点,对应YANG 数据定义中声明的“input”部分.

示例:

下面以YANG的定义的数据将用于本章的示例说明.

```
 rpc reboot {
   input {
     leaf delay {
       units seconds;
       type uint32;
       default 0;
     }
     leaf message { type string; }
     leaf language { type string; }
   }
 }
```

客户端应该发送如下的POST请求消息:

```
POST /.well-known/restconf/datastore/operations/
   example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json

{
  "input" : {
    "delay" : 600,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

服务端的响应应该如下:

```
HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2012 11:01:00 GMT
Server: example-server
```

### 5.4.2 输出参数编码操作

如果一个“rpc”请求中包含一个“output”部分, 则消息体会提供一个“output”节点，对应YANG数据定义中声明的“output”部分.

例子:

下面以YANG的定义的数据将用于本章的示例说明.

```
 rpc get-reboot-info {
   output {
     leaf reboot-time {
       units seconds;
       type uint32;
     }
     leaf message { type string; }
     leaf language { type string; }
   }
 }
```

客户端应该会发送如下请求:

```
POST /.well-known/restconf/datastore/operations/
   example-ops:get-reboot-info HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json
```

服务端的响应应该如下:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server

{
  "output" : {
    "reboot-time" : 30,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

## 5.5 资源事件

```
1 待定
```

## 6.错误报告

HTTP Status-Lines用来反应RESTCONF操作是成功还是失败。这里的<rpc-error>元素标记了NETCONF返回后记录的一些常用信息。这些错误信息适用于RESTCONF，并返回"4xx"状态码。

下列表格整理了RESTCONF操作返回状态码：

RESTCONF用到的HTTP状态码

| 状态码 | 描述 |
| --- | --- |
| 100 Continue(继续) | 请求接受后，继续201 |
| 200 OK(成功) | 响应成功 |
| 201 Created(创建) | 创建资源成功 |
| 202 Accepted(接受) | 接受资源 |
| 204 No Content(无内容) | 没有返回成功 |
| 304 Not Modified(未修改) | 操作没有完成 |
| 400 Bad Request(错误请求) | 请求的消息无效 |
| 403 Forbidden(禁止) | 拒绝访问资源 |
| 404 Not Found(未找到) | 未找到目标资源或资源节点 |
| 405 Method Not Allowed(方法禁用) | 目标资源不允许调用的方法 |
| 409 Conflict(冲突) | 资源或被锁定 |
| 413 Request Entity Too Large(请求实体过大) | 请求的实体过大错误 |
| 414 Request-URI Too Large(请求的URI过长) | 请求的URI过长错误 |
| 415 Unsupported Media Type(不支持的媒体类型) | 不是RESTCONF支持的媒体类型 |
| 500 Internal Server Error(服务器内部错误) | 操作失败 |
| 501 Not Implemented(尚未实施) | 未知的操作 |
| 503 Service Unavailable(服务不可用) | 服务错误 |

由于资源定义为YANG "rpc"语句，NETCONF <error-tag>值和HTTP状态代码之间需要有必要的映射。特定的错误条件和响应代码使用的data-model具体可以包含在"rpc"语句声明的YANG "description"语句中。

error-tag与状态码的映射

| <error-tag>值 | 状态码 |
|---|---|
| in-use | 409 |
| invalid-value | 400 |
| too-big | 413 |
| missing-attribute | 400 |
| bad-attribute | 400 |
| unknown-attribute | 400 |
| bad-element | 400 |
| unknown-element | 400 |
| unknown-namespace | 400 |
| access-denied | 403 |
| lock-denied | 409 |
| resource-denied | 409 |
| rollback-failed | 500 |
| data-exists | 409 |
| data-missing | 409 |
| operation-not-supported | 501 |
| operation-failed | 500 |
| partial-operation | 500 |
| malformed-message | 400 |

## 6.1 错误消息的响应

当一个data resource或operation resource请求发生错误时,或者出现"4xx"状态码（除"403"状态码外）时，那服务器应该发送一个"errors"包含了YANG module Section 8定义的响应主体。

例：

以下示例显示一个datastore resource的"lock-denied"错误。

```
POST /.well-known/restconf/operations/lock-datastore HTTP/1.1
Host: example.com
```

服务器响应可能为：

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/vnd.yang.api+json

{
  "errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "lock-denied",
      "error-message": "Lock failed, lock is already held",
    }
  }
}
```

## 7. YANG Patch

YANG Patch 操作支持在 RESTCONF 内执行复杂的编辑操作。"plain patch"操作仅仅提供对目标数据存储的简单合并编辑操作。

"YANG Patch" 是对目标数据存储服务器的一组有序的编辑应用。在 YANG 模块第 8 节，特定字段被定义为'yang-patch'容器。

每个块都是由客户提供的字符串，被叫做“patch-id”。

客户端可以控制错误处理的类型，并且适合编辑列表。

## 7.1 为什么不用JSON 补丁?

RESTCONF 补丁方法需要补丁内容指定的媒体类型，因此它可以使用任何的补丁机制，包括JSON补丁[RFC6902].

RESTCONF协议是旨在利用YANG数据模型语言来指定内容模式的。JSON补丁和RESTCONF之间互不相容是有如下原因：

- 一个补丁机制要与XML或JSON编码合作是必要的。
- YANG配置节点可以由一个或多个关键叶子等复杂键来命名。JSON数组的包装和所有YANG键都会被收起到单个整型索引中。
- YANG配置节点用关键叶子被命名为稳定，永久标识。JSON数组是被包装的，如果添加或删除自身条目，那所有条目都会自加，数组就要重新进行编号。
- 编辑的业务操作设置需要与NETCONF协议一致，但JSON补丁并没有提供一致的编辑业务操作的设置。
- 数据存储校验过程必须是具体的，并与YANG校验过程是一致的。
- 错误的报告需要与NETCONF协议一致，但JSON补丁并没有提供一致的错误报告机制。

## 7.2 YANG 补丁的目的是数据节点

目标数据节点在请求每个“编辑”项的“目标”叶节点时，每次编辑操作都通过目标资源的值决定的。

如果目标资源规定请求的URI标识一个数据存储资源，当“目标”的路径字符串叶是一个绝对路径表达式。第一个节点中指定的“目标”叶是一个顶级的数据节点，其被定义在YANG模块中。

如果目标资源请求中指定URI标识一个数据资源，然后在“目标”的路径字符串叶是一个相对路径表达式。第一个节点中指定的“目标”叶是一个数据节点的子节点与目标相关联的资源。

## 7.3 YANG Patch 编辑操作

每一个 YANG patch 都可以在目标数据节点上编辑指定的编辑操作。操作集合匹配 NETCONF 的编辑操作，不过它还包括一些新的操作。

YANG Patch 编辑操作

| 操作 | 描述 |
| --- | --- |
| create | 创建一个新的资源，假设资源不是已经存在的或错误的 |
| delete | 删除一个数据资源，假设资源存在或错误 |
| insert | 插入一个新的用户请求数据资源 |
| merge | 合并目标资源的的编辑值；如果不存在就创建资源 |
| move | 调整目标资源 |
| replace | 用编辑的值替换目标数据资源 |
| remove | 移除数据资源，假设资源已经存在或不是错误的 |

## 7.4 YANG Patch 错误处理

有三种错误的处理模式是服务必须提供的。这些模式指定了服务器在错误发生时所处理的行为，且是可编辑的。服务器必须确保一种良好的消息接收格式，提供的消息要符合 YANG 格式的定义，并放在"patch" 容器中，YANG 模块被定义在第8节。

如果一个结构良好，模式有效的 YANG Patch 消息被接收，那么服务器将会处理提供的编辑并以升序排序。以下错误模式适用于这个编辑列表的处理：

- all-or-none： 所有指定的编辑必须被应用，或者目标数据存储的内容应该在 PATCH 方法开始前被返回到原来的状态。服务器可能会在目标数据上完全确定地存储失败。也有可能回滚失败或“撤销”操作失败。
- stop-on-error: 如果出现错误，每个编辑将尝试排序，服务器将会停止处理编辑列表并且返回一个错误报告标识，这个标识会指示是由哪些编辑列表条目导致的错误。
- continue-on-error: 如果出现错误，每个编辑将尝试排序，服务器将会记录一个错误标识，这个标识指示引发错误的编辑列表条目，并继续下一个编辑条目。

如果它会发生改变，那么在编辑之后，还没被执行之前，服务器就会保存这些运行数据到持久性存储器上。

## 7.5 YANG Patch 响应

为了报告每次独立编辑的状态信息，YANG Patch 操作将返回一个特殊的响应。该响应也可以报告一般错误信息。在第8章关于YANG 概念容器定义中的“yang-patch-status”部分中我们定义了这种特殊响应的语法。

## 7.6 YANG Patch 示例

### 7.6.1 关于错误后继续的示例

下面的示例中将展示 一些歌曲添加到一个已经存在的唱片中。

- 每个“edit”中包含一首歌
- 第一首歌已经存在，所以针对第一首歌进行的添加操作将会报告一个错误
- 遇到错误的处理方式为“遇到错误后继续”，所以针对其余部分的添加操作将不会遇到错误并且添加成功.、

```
Request from client:

  PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
     library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
  Host: example.com
  Content-Type: application/vnd.yang.patch+json

  {
    "yang-patch" : {
      "patch-id" : "add-songs-patch",
      "error-action" : "continue-on-error",
      "edit" : [
        {
          "edit-id" : 1,
          "operation" : "create",
          "target" : "/song",
          "value" : {
            "song" : {
```

```
              "name" : "Bridge Burning",
              "location" : "/media/bridge_burning.mp3",
              "format" : "MP3",
              "length" : 288
            }
          }
        },
        {
          "edit-id" : 2,
          "operation" : "create",
          "target" : "/song",
          "value" : {
            "song" : {
              "name" : "Rope",
              "location" : "/media/rope.mp3",
              "format" : "MP3",
              "length" : 259
            }
          }
        },
        {
          "edit-id" : 3,
          "operation" : "create",
          "target" : "/song",
          "value" : {
            "song" : {
              "name" : "Dear Rosemary",
              "location" : "/media/dear_rosemary.mp3",
              "format" : "MP3",
              "length" : 269
            }
          }
        }
      ]
    }
  }

Response from server:

  HTTP/1.1 409 Conflict
  Date: Mon, 23 Apr 2012 13:01:20 GMT
  Server: example-server
  Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
  Content-Type: application/vnd.yang.api+json

  {
    "yang-patch-status" : {
      "patch-id" : "add-songs-patch",
      "edit-status" : {
        "edit" : [
          {
```

```
            "edit-id" : 1,
            "errors" : {
              "error" : {
              "error-type": "application",
              "error-tag": "data-exists",
              "error-path": "/example-jukebox:jukebox/library/artist/
                 Foo%20Fighters/album/Wasting%20Light/song/
                 Burning%20Light",
              "error-message": "Data already exists, cannot be
               created"
              }
            },
            {
              "edit-id" : 2,
              "location" : "http://example.com/.well-known/restconf/
                 datastore/example-jukebox:jukebox/library/artist/
                 Foo%20Fighters/album/Wasting%20Light/song/Rope"
            },
            {
              "edit-id" : 3,
              "location" : "http://example.com/.well-known/restconf/
                 datastore/example-jukebox:jukebox/library/artist/
                 Foo%20Fighters/album/Wasting%20Light/song/
                 Dear%20Rosemary"
            }
          }
        ]
      }
    }
  }
```

### 7.6.2 示例关于移动列表中的数据

下面的示例将展示从一个已经存在的播放列表中移动一首歌。我们将要把播放列表中的第一首歌“Foo-One”移动到同一播放列表中第三首歌的后边，本操作会成功，所以下面将展示返回一个没有错误的响应示例。

```
Request from client:

  PATCH /.well-known/restconf/datastore/example-jukebox:jukebox/
    playlist/Foo-One   HTTP/1.1
  Host: example.com
  Content-Type: application/vnd.yang.patch+json

  {
    "yang-patch" : {
      "patch-id" : "move-song-patch",
      "error-action" : "all-or-none",
      "edit" : [
```

```
        {
          "edit-id" : 1,
          "operation" : "move",
          "target" : "/song/1",
          "point" : "/song3",
          "where" : "after"
        }
      ]
    }
  }

Response from server:

  HTTP/1.1 400 OK
  Date: Mon, 23 Apr 2012 13:01:20 GMT
  Server: example-server
  Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
  Content-Type: application/vnd.yang.api+json

  {
    "yang-patch-status" : {
      "patch-id" : "move-song-patch",
      "edit-status" : {
        "edit" : [
          {
            "edit-id" : 1,
            "ok" : [ null ]
          }
        ]
      }
    }
  }
```

# 8. RESTCONF 模块

RFC Ed.:在RFC发布后更新这个日期并且移除这条注释。

<代码开始> 文件"ietf-restconf@2013-09-04.yang"

```
module ietf-restconf {
  // RFC Ed.: replace XXXX with 'ietf' and remove this note
  namespace "urn:XXXX:params:xml:ns:yang:ietf-restconf";
  prefix "restconf";

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

```

```
  organization
    "IETF NETCONF (Network Configuration) Working Group";

  contact
    "Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

     Editor:   Kent Watsen
               <mailto:kwatsen@juniper.net>

     Editor:   Rex Fernando
               <mailto:rex@cisco.com>";

  description
    "This module contains conceptual YANG specifications
     for the YANG Patch and error content that is used in
     RESTCONF protocol messages. A conceptual container
     representing the RESTCONF API nodes (type vnd.yang.api).

     Note that the YANG definitions within this module do not
     represent configuration data of any kind.
     The YANG grouping statements provide a normative syntax
     for XML and JSON message encoding purposes.

     Copyright (c) 2013 IETF Trust and the persons identified as
     authors of the code.  All rights reserved.

     Redistribution and use in source and binary forms, with or
     without modification, is permitted pursuant to, and subject
     to the license terms contained in, the Simplified BSD License
     set forth in Section 4.c of the IETF Trust's Legal Provisions
     Relating to IETF Documents
     (http://trustee.ietf.org/license-info).

     This version of this YANG module is part of RFC XXXX; see
     the RFC itself for full legal notices.";

  // RFC Ed.: replace XXXX with actual RFC number and remove this
  // note.

  // RFC Ed.: remove this note
  // Note: extracted from draft-bierman-netconf-restconf-00.txt

  // RFC Ed.: update the date below with the date of RFC publication
  // and remove this note.
  revision 2013-09-04 {
    description
      "Initial revision.";
```

```
    reference
      "RFC XXXX: RESTCONF Protocol.";
  }

  typedef data-resource-identifier {
    type string {
      length "1 .. max";
    }
    description
      "Contains a Data Resource Identifier formatted string
       to identify a specific data node.";
    reference
      "RFC XXXX: [sec. 5.3.1.1 ABNF For Data Resource Identifiers]";
  }

  // this typedef is TBD; not currently used
  typedef datastore-identifier {
    type union {
      type enumeration {
        enum candidate {
          description
            "Identifies the NETCONF shared candidate datastore.";
          reference
            "RFC 6241, section 8.3";
        }
        enum running {
          description
            "Identifies the NETCONF running datastore.";
          reference
            "RFC 6241, section 5.1";
        }
        enum startup {
          description
            "Identifies the NETCONF startup datastore.";
          reference
            "RFC 6241, section 8.7";
        }
      }
      type string;
    }
    description
      "Contains a string to identify a specific datastore.
       The enumerated datastore identifier values are
       reserved for standard datastore names.";
  }

  grouping yang-patch {

    description
      "A grouping that contains a YANG container
       representing the syntax and semantics of a
```

```
       YANG Patch edit request message.";

    container yang-patch {
      description
        "Represents a conceptual sequence of datastore edits,
         called a patch. Each patch is given a client-assigned
         patch identifier. A patch is applied with client-specified
         error handling to control how the ordered list of edits
         is applied if an error is encountered.

         A patch MUST be validated by the server to be a
         well-formed message before any of the patch edits
         are validated or attempted.

         The validation model for patches closely follows
         the constraint enforcement model in YANG, except it
         is conceptually enforced on an ordered list of edits.

         The server MUST conceptually perform field validation
         for each edit in ascending order, as defined in RFC 6020,
         section 8.3.1 and 8.3.2.  This is most relevant if the edit
         error-action is 'stop-on-error', since the identification
         of the first error determines where edit processing is
         terminated.

         If YANG datastore validation (defined in RFC 6020, section
         8.3.3) is required, then it performed after all edits have
         been individually validated.

         It is possible for a datastore constraint violation to occur
         due to any node in the datastore, including nodes not
         included in the edit list. Any validation errors SHOULD
         be reported in the reply message.

         If datastore validation is required and fails, the server
         SHOULD NOT allow the datastore to remain invalid. It is an
         implementation-specific matter how the server fixes the
         invalid datastore.  For example, the server might prune
         invalid nodes causing the datastore validation error,
         or undo the entire patch.";

      reference
        "RFC 6020, section 8.3.";

      leaf patch-id {
        type string;
        description
          "An arbitrary string provided by the client to identify
           the entire patch.  This value SHOULD be present in any
           audit logging records generated by the server for the
           patch. Error messages returned by the server pertaining
```

```
          to this patch will be identified by this patch-id value.";
      }

      leaf error-action {
        type enumeration {
          enum all-or-none {
            description
              "The server will apply all edits in the patch only
               if no errors occur.  If any errors occur then
               none of the edits will be applied and the
               contents of the target datastore MUST be unchanged.";
          }
          enum stop-on-error {
            description
              "The server will apply edits in the specified order
               and will stop processing edits if any error occurs.
               Any previous edits which were successfully applied
               will remain applied.  No further edits will be
               attempted after the first error is encountered.";
          }
          enum continue-on-error {
            description
              "The server will apply edits in the specified order
               and will continue processing edits if any error
               occurs.";
          }
        }
        default all-or-none;
        description
          "The error handling behavior for the ordered list of
           edits.";
      }

      list edit {
        key edit-id;

        description
          "Represents one edit within the YANG Patch
           request message.";

        leaf edit-id {
          type uint32;
          description
            "Arbitrary integer index for the edit.
             The server MUST process edits in ascending order.
             Error messages returned by the server pertaining
             to a specific edit will be identified by this
             identifier value.";
        }

        leaf operation {
```

```
          type enumeration {
            enum create {
              description
                "The target data node is created using the
                 supplied value, only if it does not already
                 exist.";
            }
            enum delete {
              description
                "Delete the target node, only if the data resource
                 currently exists, otherwise return an error.";
            }
            enum insert {
              description
                "Insert the supplied value into a user-ordered
                 list or leaf-list entry. The target node must
                 represent a new data resource.";
            }
            enum merge {
              description
                "The supplied value is merged with the target data
                 node.";
            }
            enum move {
              description
                "Move the target node. Reorder a user-ordered
                 list or leaf-list. The target node must represent
                 an existing data resource.";
            }
            enum replace {
              description
                "The supplied value is used to replace the target
                 data node.";
            }
            enum remove {
              description
                "Delete the target node if it currently exists.";
            }
          }
          mandatory true;
          description
            "The datastore operation requested for the associated
             edit entry";
        }

        leaf target {
          type data-resource-identifier;
          mandatory true;
          description
            "Identifies the target data resource for the edit
             operation.";
```

```
        }

        leaf point {
          when "(../operation = 'insert' or " +
            "../operation = 'move') and " +
            "(../where = 'before' or ../where = 'after')" {
            description
              "Point leaf only applies for insert or move
               operations, before or after an existing entry.";
          }
          type data-resource-identifier;
          description
            "The absolute URL path for the data node that is being
             used as the insertion point or move point for the
             target of this edit entry.";
        }

        leaf where {
          when "../operation = 'insert' or ../operation = 'move'" {
            description
              "Where leaf only applies for insert or move
               operations.";
          }
          type enumeration {
            enum before {
              description
                "Insert or move a data node before the data resource
                 identified by the 'point' parameter.";
            }
            enum after {
              description
                "Insert or move a data node after the data resource
                 identified by the 'point' parameter.";
            }
            enum first {
              description
                "Insert or move a data node so it becomes ordered
                 as the first entry.";
            }
            enum last {
              description
                "Insert or move a data node so it becomes ordered
                 as the last entry.";
            }

          }
          default last;
          description
            "Identifies where a data resource will be inserted or
             moved. YANG only allows these operations for
             list and leaf-list data nodes that are ordered-by
```

```
            user.";
        }

        anyxml value {
          when "(../operation = 'create' or " +
            "../operation = 'merge' " +
            "or ../operation = 'replace' or " +
            "../operation = 'insert')" {
            description
              "Value node only used for create, merge,
               replace, and insert operations";
          }
          description
            "Value used for this edit operation.";
        }
      }
    }

  } // grouping yang-patch


  grouping yang-patch-status {

    description
      "A grouping that contains a YANG container
       representing the syntax and semantics of
       YANG Patch status response message.";

    container yang-patch-status {
      description
        "A container representing the response message
         sent by the server after a YANG Patch edit
         request message has been processed.";

      leaf patch-id {
        type string;
        description
          "The patch-id value used in the request";
      }

      container global-errors {
        uses errors;
        description
          "This container will be present if global
           errors unrelated to a specific edit occurred.";
      }

      container edit-status {
        description
          "This container will be present if there are
           edit-specific status responses to report.";
```

```
366
367         list edit {
368           key edit-id;
369
370           description
371             "Represents a list of status responses,
372              corresponding to edits in the YANG Patch
373              request message.";
374
375           leaf edit-id {
376             type uint32;
377              description
378                "Response status is for the edit list entry
379                 with this edit-id value.";
380           }
381           choice edit-status-choice {
382             description
383               "A choice between different types of status
384                responses for each edit entry.";
385
386             leaf ok {
387               type empty;
388               description
389                 "This edit entry was invoked without any
390                  errors detected by the server associated
391                  with this edit.";
392             }
393             leaf location {
394               type inet:uri;
395               description
396                 "Contains the Location header value that would be
397                  returned if this edit causes a new resource to be
398                  created. If the edit identified by the same edit-id
399                  value was successfully invoked and a new resource
400                  was created, then this field will be returned
401                  instead of 'ok'.";
402             }
403             leaf skipped {
404               type empty;
405               description
406                 "This edit entry was skipped or not reached
407                  by the server.";
408             }
409             case errors {
410               uses errors;
411               description
412                 "The server detected errors associated with the
413                   edit identified by the same edit-id value.";
414             }
415           }
416         }
```

```
      }
    }
  }  // grouping yang-patch-status

  grouping errors {

    description
      "A grouping that contains a YANG container
       representing the syntax and semantics of a
       YANG Patch errors report within a response message.";

    container errors {
      config false;  // needed so list error does not need a key
      description
        "Represents an error report returned by the server if
         a request results in an error.";

      list error {
        description
          "An entry containing information about one
           specific error that occurred while processing
           a RESTCONF request.";
        reference "RFC 6241, Section 4.3";

        leaf error-type {
          type enumeration {
            enum transport {
              description "The transport layer";
            }
            enum rpc {
              description "The rpc or notification layer";
            }
            enum protocol {
              description "The protocol operation layer";
            }
            enum application {
              description "The server application layer";
            }
          }
          mandatory true;
          description
            "The protocol layer where the error occurred.";
        }

        leaf error-tag {
          type string;
          mandatory true;
          description
            "The enumerated error tag.";
        }

```

```
        leaf error-app-tag {
          type string;
          description
            "The application-specific error tag.";
        }

        leaf error-path {
          type data-resource-identifier;
          description
            "The target data resource identifier associated
             with the error, if any.";
        }

        leaf error-message {
          type string;
          description
            "A message describing the error.";
        }

        container error-info {
           description
             "A container allowing additional information
              to be included in the error report.";
           // arbitrary anyxml content here
        }
      }
    }
  } // grouping errors


  grouping restconf {

    description
      "A grouping that contains a YANG container
       representing the syntax and semantics of
       the RESTCONF API resource.";

    container restconf {
      description
        "Conceptual container representing the vnd.yang.api
         resource type.";

      container datastore {
        description
          "Container representing the vnd.yang.datastore resource
           type. Represents the conceptual root of the unified
           datastore containing YANG data nodes.  The child nodes
           of this container can be data resources (vnd.yang.data)
           defined as top-level YANG data nodes from the modules
           advertised by the server in /restconf/modules.";
      }
```

```
519       container modules {
520         description
521           "Contains a list of module description entries.
522            These modules are currently loaded into the server.";
523
524         list module {
525           key "name revision";
526           description
527             "Each entry represents one module currently
528              supported by the server.";
529
530           leaf name {
531             type string;
532             description "The YANG module name.";
533           }
534           leaf revision {
535             type union {
536               type yang:date-and-time;
537               type string { length 0; }
538             }
539             description
540               "The YANG module revision date. An empty string is
541                used if no revision statement is present in the
542                YANG module.";
543           }
544           leaf namespace {
545             type inet:uri;
546             mandatory true;
547             description
548               "The XML namespace identifier for this module.";
549           }
550           leaf-list feature {
551             type string;
552             description
553               "List of YANG feature names from this module that are
554                supported by the server.";
555           }
556           leaf-list deviation {
557             type string;
558             description
559               "List of YANG deviation module names used by this
560                server to modify the conformance of the module
561                associated with this entry.";
562           }
563         }
564       }
565       container operations {
566         description
567           "Container for all operation resources
568            (vnd.yang.operation),
569
```

```
            Each resource is represented as an empty leaf with the
            name of the RPC operation from the YANG rpc statement.

            E.g.;

               POST /restconf/operations/show-log-errors

               leaf show-log-errors {
                 type empty;
               }
          ";
        }
        leaf version {
          type enumeration {
            enum "1.0" {
              description
                "Version 1.0 of the RESTCONF protocol.";
            }
          }
          config false;
          description
            "Contains the RESTCONF protocol version.";
        }
      }
    }  // grouping restconf

}
```

<代码结束>

# 9. IANA 注意事项

## 9.1 著名的URI

这份备忘录登记了被URI注册表(RFC5785)所定义的众所周知的著名的URI。

```
   URI suffix: restconf

   Change controller: IETF

   Specification document(s): RFC XXXX

   Related information: None
```

## 9.2 YANG 模块注册信息

本文档登记了在IETF XML注册表[RFC3688]中注册的一个URI，根据RFC3688中定义的格式，下列注册信息必须被登记。

```
    // RFC Ed.: replace XXXX with 'ietf' and remove this note
    URI: urn:XXXX:params:xml:ns:yang:ietf-restconf
    Registrant Contact: The NETMOD WG of the IETF.
    XML: N/A, the requested URI is an XML namespace.
```

本文档在YANG 模块名称注册表[RFC6020]中注册了一个YANG模块。

```
name:         ietf-restconf
namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf
prefix:       restconf
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX
```

# 10. 安全注意事项

待定

# 11. 更新日志

```
    -- RFC Ed.: 在发布之前请删除这章
```

## 11.1 YANG-API-01 到 RESTCONF-00

- **协议重命名。**将YANG-API 改为 RESTCONF
- **阐明属性模式。**容器和列表是子资源，所有其他YANG数据节点类型是父资源的属性
- 去除了YANG 针对'optional-key'扩展部分
- 如果目标资源的数据节点缺失服务器将返回默认值，服务器返回的默认值使用其leaf默认值

- 'depth' 参数的默认值从"1"修改为'unbounded'，depth被限定为整形，所以客户端只能为这个参数设置一个整形值
- 'format'参数的默认值从 'json' 改变为 'xml'
- 扩展介绍说明
- 去除事务
- 去除部分功能
- 去除了Range 和 IfRange消息头的使用
- 简化编译模型
- 从 ietf-restconf.yang中去除全局协议操作
- 改变RPC操作术语为协议操作
- 更新JSON参考引用

- 更新未解决问题章节
- 更新IANA章节
- 添加YANG Patch部分
- 在 ietf-restconf.yang中添加YANG定义部分
- 添加合著者Kent Watsen 和Rex Fernando
- **更新YANG模块。**更新后通过 pyang 进行ietf checking（校验）
- **更新示例。**更新后URIs资源名称对应所使用模块名称，并以此方式来识别数据资源
- **更新 depth 行为。**更新后服务端将不为"GET /.well-known/restconf"返回全部内容；服务器将停止在新资源类型；例如 yang.api -->yang.dataastore 返回一个dataastore命名的空节点；yang.api --> yang.operation 返回一个operation命名的空节点；

# 12. 关闭问题

- WG 应该做这项工作吗？NETCONF？NETMOD？目前还不清楚 RESTCONF 的概念和标准文件由两个工作小组所拥有的文档建立的。

A: NETCONF WG 将会做这项工作。

- 会话应该使用吗？ “可重用会话”应该被使用吗？对审核能更好吗？如何锁定 .well-known/restconf/datastore 资源的多编辑工作，如果是一个会话操作呢？服务器什么时候释放锁？是由它是否已经被抛弃或客户端断开连接决定的？

A: RESTCONF 是一个低会话协议。它可以实现利用持久 HTTP 连接， 但是这不是必需的或者必被设计到成协议之中。

- "/.well-known/restconf/modules" 资源在 API 资源内是一个单独的资源，应该有自己的时间戳吗？当前 API 的时间戳与任何变更都会被成对加载进模块列表。那么 API 资源应该被静态化并进行缓存吗？

A: 所有子容器被认为是子资源。服务器可能支持数据节点的时间戳和实体ID。

- 如何处理没有REMOVE操作的问题，就靠DELETE？效果是本地去请求；一个 NETCONF 编辑的配置是糟糕的，因为 netconf 请求有可能是 create/delete/modify 很多节点。

A: YANG Patch 操作有 remove 或者 delete 语义。

- 应该是每一个 YANG 数据节点都是一个数据源，那么每一个 YANG RPC 都要声明一个操作资源？YANG 扩展需要允许数据模块控制资源的边界吗？

A: 嵌套的容器和列表被认为是子资源。终端节点（叶子，叶子列表，任何xml）被认为是父资源的属性。

- 在 YANG 资源创造秩序和其他资源之间的依赖关系是不确定的。YANG 有 leafrefs 和 instance-identifiers（实例标识符），可用于识别一些顺序依赖关系。是否在 RESTCONF 中需要的任何新机制需要确定资源创造秩序和其他依赖需求？

A:  YANG补丁允许客户端控制创建指令时可以编辑多个资源。编辑操作允许服务器命令所有后代资源由客户端提供的单数据存储编辑目标节点。

-

编码叶参数，需要一些额外的元数据吗？叶参数节点需要响应（RFC 5988）识别或YANG模块要提供这个元数据吗？

A：不需要特别对所需的叶参数进行编码。无论服务器使用何种协议或编码方式，都必须理解YANG模式。

- 默认的算法定义数据资源应该注意些什么？默认从另一个名称空间增加应该开始一个新的资源吗？顶级数据节点作为一个资源好吗？

A: 增强节点与其他嵌套YANG结构不遵循差异的规则。容器和列表可以从新的子资源开始。

## 13. 待解决问题

- 在RESTCONF 消息中没有消息Id属性。是否需要一个消息唯一标识？如果需要，应该从RFC 2392中的"Message-ID" 或"Content-ID" 消息头选择一个作为消息唯一标识吗？
- 用于"select"请求参数应该使用什么语法？目前的选择有"xpath"和"path-expr"。是否应该添加一个参数用于确定一个select请求参数字符串格式以便以后扩展？
- 是否需要所有 RESTCONF 用于支持例如：FastCGI和WSGI等普通应用框架的消息头？如果不需要，是否query参数应该被替换，尤其是QUERY_STRING以被广泛应用于WEB应用？
- <errors>元素是否应该在错误响应中分离出来，成为独立的媒体类型？
- 如何添加额外的数据存储支持，用于未来可能添加进来的NETCONF/NETMOD 框架？
- 客户端如何知道服务器支持的除application/vnd.yang.data和application/vnd.yang.patch 以外的PATCH（补丁）媒体类型？

- 是否 /.well-known/restconf/version属性应该被定义为元数据？它是否应该被返回作为XRD(扩展资源介绍)？是否需要除去或替代原有的版本属性？是否应该在1.0版本时使用 ietf-restconf YANG中的修订时间来替代版这个版本号字符串？

- 通知消息传递部分待定

- NETCONF和RESTCONF通知的对齐方式预计将非常接近RFC 5277的设计。发布/订阅特征待定。

- 一些章节将被重写用于支持通知和事件资源

- 自从数据资源只能为YANG 容器或者列表，如果顶层YANG数据节点不是YANG 容器或者列表应该如何处理？在RESTCONF中这种情况是否被允许？

- 一个choice 可以作为一个资源吗？ 目前RESTCONF 中不支持YANG 的choice部分。

## 14. YANG 模块示例

```
001 module example-jukebox {
002
003   namespace "http://example.com/ns/example-jukebox";
004   prefix "jbox";
005
006   organization "Example, Inc.";
007   contact "support at example.com";
008   description "Example Jukebox Data Model Module";
009   revision "2013-09-04" {
010     description "Initial version.";
011     reference "example.com document 1-4673";
012   }
013
014   identity genre {
015     description "Base for all genre types";
```

```
016     }
017
018     // abbreviated list of genre classifications
019     identity Alternative {
020       base genre;
021       description "Alternative music";
022     }
023     identity Blues {
024       base genre;
025       description "Blues music";
026     }
027     identity Country {
028       base genre;
029       description "Country music";
030     }
031     identity Jazz {
032       base genre;
033       description "Jazz music";
034     }
035     identity Pop {
036       base genre;
037       description "Pop music";
038     }
039     identity Rock {
040       base genre;
041       description "Rock music";
042     }
043
044     container jukebox {
045       presence
046         "An empty container indicates that the jukebox
047          service is available";
048
049       description
050         "Represents a jukebox resource, with a library, playlists,
051          and a plaay operation.";
052
053       container library {
054
055         description "Represents the jukebox library resource.";
056
057         list artist {
058           key name;
059
060           description
061             "Represents one artist resource within the
062              jukebox library resource.";
063
064           leaf name {
065             type string {
066               length "1 .. max";
```

```
    }
    description "The name of the artist.";
  }

  list album {
    key name;

    description
      "Represents one album resource within one
       artist resource, within the jukebox library.";

    leaf name {
      type string {
        length "1 .. max";
      }
      description "The name of the album.";
    }

    leaf genre {
      type identityref { base genre; }
      description
        "The genre identifying the type of music on
         the album.";
    }

    leaf year {
      type uint16 {
        range "1900 .. max";
      }
      description "The year the album was released";
    }

    list song {
      key name;

      description
        "Represents one song resource within one
         album resource, within the jukebox library.";

      leaf name {
        type string {
           length "1 .. max";
        }
        description "The name of the song";
      }
      leaf location {
        type string;
        mandatory true;
        description
         "The file location string of the
          media file for the song";
```

```
            }
            leaf format {
              type string;
              description
                "An identifier string for the media type
                 for the file associated with the
                 'location' leaf for this entry.";
            }
            leaf length {
              type uint32;
              units "seconds";
              description
                "The duration of this song in seconds.";
            }
          }   // end list 'song'
        }   // end list 'album'
      }  // end list 'artist'

      leaf artist-count {
         type uint32;
         units "songs";
         config false;
         description "Number of artists in the library";
      }
      leaf album-count {
         type uint32;
         units "albums";
         config false;
         description "Number of albums in the library";
      }
      leaf song-count {
         type uint32;
         units "songs";
         description "Number of songs in the library";
      }
    }  // end library

    list playlist {
      key name;

      description
        "Example configuration data resource";

      leaf name {
        type string;
        description
          "The name of the playlist.";
      }
      leaf description {
        type string;
        description
```

```
          "A comment describing the playlist.";
      }
      list song {
        key index;
        ordered-by user;

        description
          "Example nested configuration data resource";

        leaf index {    // not really needed
          type uint32;
          description
            "An arbitrary integer index for this
             playlist song.";
        }
        leaf id {
          type instance-identifier;
          mandatory true;
          description
            "Song identifier. Must identify an instance of
             /jukebox/library/artist/album/song/name.";
        }
      }
    }

    container player {
      description
        "Represents the jukebox player resource.";

      leaf gap {
        type decimal64 {
          fraction-digits 1;
          range "0.0 .. 2.0";
        }
        units "tenths of seconds";
        description "Time gap between each song";
      }
    }
  }

  rpc play {
    description "Control function for the jukebox player";
    input {
      leaf playlist {
        type string;
        mandatory true;
        description "playlist name";
      }
      leaf song-number {
        type uint32;
        mandatory true;
```

```
          description "Song number in playlist to play";
        }
      }
    }
  }
```

# 15. 参考

## 15.1 参考标准


- [I-D.lhotka-netmod-json]

- Lhotka, L., “Modeling JSON Text with YANG”, Internet-Draft draft-lhotka-netmod-yang-json-01 (work in progress), April 2013.

- [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>.

- [RFC2616]

- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

- [RFC3688]

- Mealling, M., “The IETF XML Registry”, BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <http://www.rfc-editor.org/info/rfc3688>.

- [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>.

- [RFC5785]

- Nottingham, M. and E. Hammer-Lahav, “Defining Well-Known Uniform Resource Identifiers (URIs)”, RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

- [RFC5789]

- Dusseault, L. and J. Snell, “PATCH Method for HTTP”, RFC 5789, DOI 10.17487/RFC5789, March 2010, <http://www.rfc-editor.org/info/rfc5789>.

- [RFC6020]

- Bjorklund, M., “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>.

- [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>.

- [RFC6536]

- Bierman, A. and M. Bjorklund, “Network Configuration Protocol (NETCONF) Access Control Model”, RFC 6536, DOI 10.17487/RFC6536, March 2012, <http://www.rfc-editor.org/info/rfc6536>.

- [RFC6991]

- Schoenwaelder, J., “Common YANG Data Types”, RFC 6991, DOI 10.17487/RFC6991, July 2013, <http://www.rfc-editor.org/info/rfc6991>.


## 15.2 参考文献


- [RFC6902]

- Bryan, P. and M. Nottingham, “JavaScript Object Notation (JSON) Patch”, RFC 6902, DOI 10.17487/RFC6902, April 2013, <http://www.rfc-editor.org/info/rfc6902>.

本文地址：https://www.oschina.net/translate/draft-ietf-netconf-restconf-08
原文地址：https://tools.ietf.org/pdf/draft-ietf-netconf-restconf-08.pdf

---