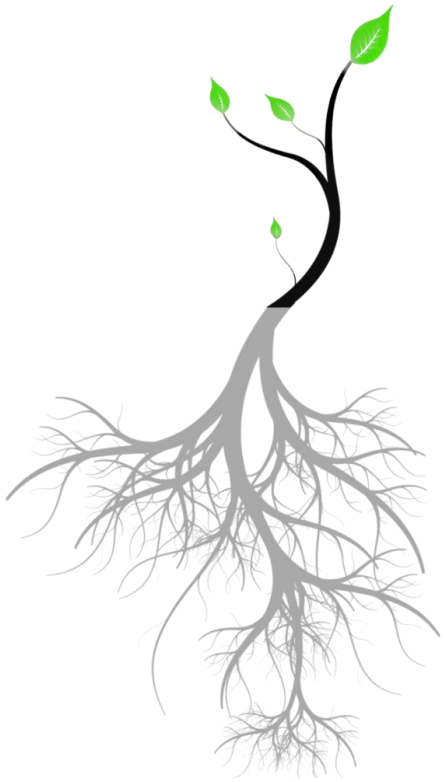
江苏省未来网络创新研究院
Jiangsu Future Network Innovation Institute

REST原理与OpenDaylight应用分享交流



SDN项目组　　Design By FLY-YE

本次培训的主要内容

1 **REST**

介绍REST的由来、架构风格、优缺点及现有技术对比

2 **OpenDaylight与REST**

结合OpenDaylight的应用场景对REST的原理进行分析

3 **REST API的思考**

总结OpenDaylight所提供的REST API，并对其合理性进行分析

R E S T

REST 不是一种具体的技术，也不是一种具体的规范，是一种内涵非常丰富的架构风格

发展

web开发技术的发展

- 静态内容阶段
- CGI程序阶段
- 脚本语言阶段

  ASP、PHP、JSP、ColdFusion
- 瘦客户端应用阶段

  Web MVC
- RIA应用阶段

  DHTML+Ajax。Ajax技术支持在不刷新页面的情况下动态更新页面中的局部内容
  DHTML开发库，例如Prototype、Dojo、ExtJS、jQuery/jQuery UI等等
- 移动Web应用阶段

  Android、iOS、Windows Phone等操作系统平台原生的开发技术之外，基于HTML5的开发技术也变得非常流行

为什么 为什么会产生REST

■从Web开发技术的发展过程看

Web从最初其设计者所构思的主要支持静态文档的阶段，逐渐变得越来越动态化。Web应用的交互模式，变得越来越复杂：从静态文档发展到以内容为主的门户网站、电子商务网站、搜索引擎、社交网站，再到以娱乐为主的大型多人在线游戏、手机游戏。

■在互联网行业，实践总是走在理论的前面

Web发展到了1995年，在CGI、ASP等技术出现之后，沿用了多年、主要面向静态文档的HTTP/1.0协议已经无法满足Web应用的开发需求，因此需要设计新版本的HTTP协议。身为HTTP/1.1协议专家组的负责人、Apache HTTP服务器的核心开发者Roy Fielding，还是Apache软件基金会的合作创始人。

■系统阐述

Roy Fielding和他的同事们在HTTP/1.1协议的设计工作中，对于Web在技术架构方面的因素做了一番深入的总结。Fiel并将这些总结纳入到了一套理论框架之中，然后使用这套理论框架中的指导原则，来指导HTTP/1.1协议的设计方向。Fielding在完成HTTP/1.1协议的设计工作之后，在他的博士学位论文中更为系统、严谨地阐述了这套理论框架，并且使用这套理论框架推导出了一种新的架构风格——REST

由来

源自一篇国外的论文

REST是一种设计风格。它不是一种标准，也不是一种软件，而是一种思想。

Roy Fielding在2000年他的博士论文中提出来的一种软件架构风格。

REST（**Re**presentational **S**tate **T**ransfer）
表象化状态转变 或者 表述性状态转移

REST 基于 HTTP，URI，以及 XML 这些现有的广泛流行的协议和标准，伴随着 REST，HTTP 协议得到了更加正确的使用。

Architectural Styles and the Design of Network-based Software Architectures

架构风格与基于网络的软件架构设计

Roy Thomas Fielding
2000

引子

软件架构和软件结构

?

软件架构

Run-time Abstraction

运行时の抽象

一个软件系统在其运行过程中某个阶段的运行时元素的抽象，关注的是软件在运行时的特性；
软件架构是一种思想，一个系统蓝图，对软件结构组成的规划和职责设定。
例如：J2EE

软件结构

静态

静态源代码的属性,源代码的模块结构;

一种层次表况，由软件组成成分构造软件的过程、方法和表示。

核心 以什么为核心进行设计

资源

Resource

REST架构风格以资源为核心进行设计

从资源的角度来观察整个网络

网络上的所有事物都是资源，分布在各处的每个资源由唯一的URI确定，而客户端的应用通过URI来获取资源的表述。

URL结构简单、可预测且易于理解

关键词

要深入理解REST，需要理解REST的五个关键词

- 资源（Resource）
- 资源的表述（Representation）
- 状态转移（State Transfer）
- 统一接口（Uniform Interface）
- 超文本驱动（Hypertext Driven）

资源

资源是一种看待服务器的方式，即，将服务器看作是由很多离散的资源组成。每个资源是服务器上一个可命名的抽象概念。因为资源是一个抽象的概念，所以它不仅仅能代表服务器文件系统中的一个文件、数据库中的一张表等等具体的东西，可以将资源设计的要多抽象有多抽象，只要想象力允许而且客户端应用开发者能够理解。与面向对象设计类似，资源是以名词为核心来组织的，首先关注的是名词。一个资源可以由一个或多个URI来标识。URI既是资源的名称，也是资源在Web上的地址。对某个资源感兴趣的客户端应用，可以通过资源的URI与其进行交互。

资源的表述

资源的表述是一段对于资源在某个特定时刻的状态的描述。可以在客户端-服务器端之间转移（交换）。资源的表述可以有多种格式，例如HTML/XML/JSON/纯文本/图片/视频/音频等。资源的表述格式可以通过协商机制来确定。请求-响应方向的表述通常使用不同的格式。

状态转移

状态转移（state transfer）
状态机中的状态迁移（state transition）的含义是不同的。
状态转移说的是：在客户端和服务器端之间转移（transfer）
代表资源状态的表述。通过转移和操作资源的表述，来间接
实现操作资源的目的

统一接口

REST要求，必须通过统一的接口来对资源执行各种操作。
对于每个资源只能执行一组有限的操作。以HTTP/1.1协议为例
HTTP/1.1协议定义了一个操作资源的统一接口，主要包括以下
内容：

- ◆7个HTTP方法：GET/POST/PUT/DELETE/PATCH/HEAD/OPTIONS
- ◆HTTP头信息（可自定义）
- ◆HTTP响应状态代码（可自定义）
- ◆一套标准的内容协商机制
- ◆一套标准的缓存机制
- ◆一套标准的客户端身份认证机制

REST还要求，对于资源执行的操作，其操作语义必须由HTTP消息体之前
的部分完全表达，不能将操作语义封装在HTTP消息体内部。这样做是为了
提高交互的可见性，以便于通信链的中间组件实现缓存、安全审计等等功能。

超文本驱动

“超文本驱动”又名“将超媒体作为应用状态的引擎”（Hypermedia As The Engine Of Application State，来自Fielding博士论文中的一句话，缩写为HATEOAS）。将Web应用看作是一个由很多状态（应用状态）组成的有限状态机。资源之间通过超链接相互关联，超链接既代表资源之间的关系，也代表可执行的状态迁移。在超媒体之中不仅仅包含数据，还包含了状态迁移的语义。以超媒体作为引擎，驱动Web应用的状态迁移。通过超媒体暴露出服务器所提供的资源，服务器提供了哪些资源是在运行时通过解析超媒体发现的，而不是事先定义的。从面向服务的角度看，超媒体定义了服务器所提供服务的协议。客户端应该依赖的是超媒体的状态迁移语义，而不应该对于是否存在某个URI或URI的某种特殊构造方式作出假设。一切都有可能变化，只有超媒体的状态迁移语义能够长期保持稳定。

约束

所谓的架构约束究竟是什么

客户端和服务器结构

连接协议具有无状态性

能够利用Cache机制

统一接口

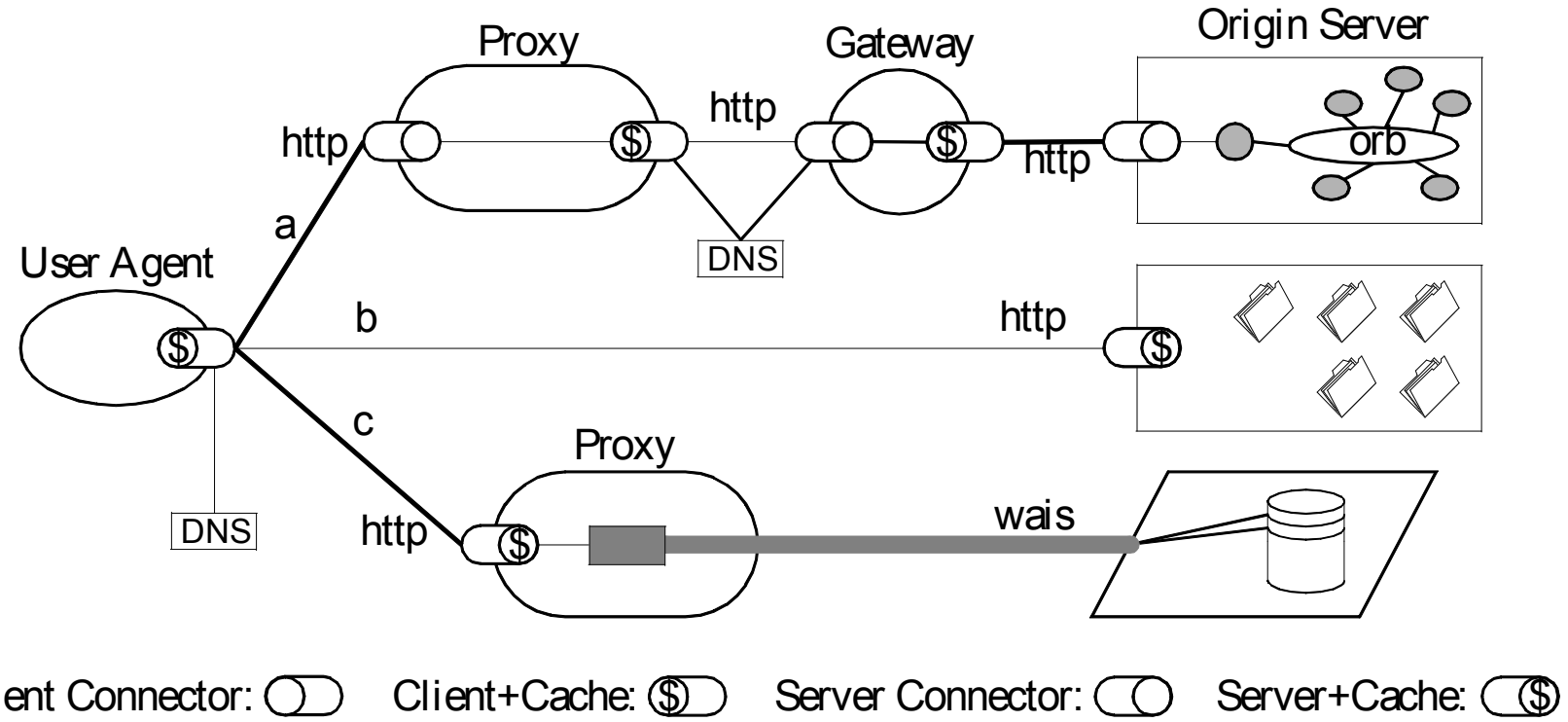
层次化的系统

按需代码Code On Demand

满足这些约束条件和原则的应用程序或设计就是 **RESTful**

过程

一个基于REST的架构的过程视图



环境差别

互联网环境与企业内网环境

1 可伸缩性需求无法控制

并发访问量可能会暴涨，也可能会暴跌。

2 安全性需求无法控制

无法控制客户端发来的请求的格式，很可能会是恶意的请求。

常见的分布式应用架构风格

①分布式对象

Distributed Objects，简称DO

架构实例有CORBA/RMI/EJB/DCOM/.NET Remoting等等

②远程过程调用

Remote Procedure Call，简称RPC

架构实例有SOAP/XML-RPC/Hessian/Flash AMF/DWR等等

③表述性状态转移

Representational State Transfer，简称REST

架构实例有HTTP/WebDAV

比较

REST与DO、RPC

比较项	REST	DO	RPC
支持抽象的工具	资源	对象	过程
统一接口	有	无	无
超文本的使用	有	无	无
数据流和管道	支持	不支持	不支持
耦合度	松耦合	紧耦合	紧耦合

比较 REST与DO

■支持抽象的工具

REST支持抽象（即建模）的工具是资源，DO支持抽象的工具是对象。在不同的编程语言中，对象的定义有很大差别，所以DO风格的架构通常都是与某种编程语言绑定的。跨语言交互即使能实现，实现起来也会非常复杂。而REST中的资源，则完全中立于开发平台和编程语言，可以使用任何编程语言来实现。

■统一接口

DO中没有统一接口的概念。不同的API，接口设计风格可以完全不同。DO也不支持操作语义对于中间组件的可见性。

■超文本的使用

DO中没有使用超文本，响应的内容中只包含对象本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比DO更高。

■数据流和管道

REST支持数据流和管道，DO不支持数据流和管道。

■耦合度

DO风格通常会带来客户端与服务器端的紧耦合。在三种架构风格之中，DO风格的耦合度是最大的，而REST的风格耦合度是最小的。REST松耦合的源泉来自于统一接口+超文本驱动。

比较 REST与RPC

■支持抽象的工具

REST支持抽象的工具是资源，RPC支持抽象的工具是过程。REST风格的架构建模是以名词为核心的，RPC风格的架构建模是以动词为核心的。简单类比一下，REST是面向对象编程，RPC则是面向过程编程。

■统一接口

RPC中没有统一接口的概念。不同的API，接口设计风格可以完全不同。RPC也不支持操作语义对于中间组件的可见性。

■超文本的使用

RPC中没有使用超文本，响应的内容中只包含消息本身。REST使用了超文本，可以实现更大粒度的交互，交互的效率比RPC更高。

■数据流和管道

REST支持数据流和管道，RPC不支持数据流和管道

■耦合度

因为使用了平台中立的消息，RPC风格的耦合度比DO风格要小一些，但是RPC风格也常常会带来客户端与服务器端的紧耦合。支持统一接口+超文本驱动的REST风格，可以达到最小的耦合度。

优势

REST有哪些优势

简

采用REST架构风格，对于开发、测试、运维人员来说，都会更简单。可以充分利用大量HTTP服务器端和客户端开发库、Web功能测试/性能测试工具、HTTP缓存、HTTP代理服务器、防火墙。这些开发库和基础设施早已成为了日常用品，不需要什么火箭科技（例如神奇昂贵的应用服务器、中间件）就能解决大多数可伸缩性方面的问题。

充分利用好通信链各个位置的HTTP缓存组件，可以带来更好的可伸缩性。其实很多时候，在Web前端做性能优化，产生的效果不亚于仅仅在服务器端做性能优化，但是HTTP协议层面的缓存常常被一些资深的架构师完全忽略掉。

活

离

统一接口+超文本驱动，带来了最大限度的松耦合。允许服务器端和客户端程序在很大范围内，相对独立地进化。对于设计面向企业内网的API来说，松耦合并不是一个很重要的设计关注点。但是对于设计面向互联网的API来说，松耦合变成了一个必选项，不仅在设计时应该关注，而且应该放在最优先位置。

OpenDaylight与REST

结合OpenDaylight的应用场景对REST的原理进行分析

REST的应用场景举例

1. REST定义了应该如何正确地使用（这和大多数人的实际使用方式有很大不同）Web标准，例如HTTP和URI。如果你在设计应用程序时能坚持REST原则，那就预示着你将会得到一个使用了优质Web架构的系统。

——Stefan Tilkov

2. 使用REST的最佳的场景是对外提供公开的服务，也就是所谓的OpenAPI，也有的人认为RESt更适合资源导向的网站，像youtube这样的网站。

3. REST 的真正价值在于 Web Services，而不是通过浏览器操作的应用程序。
Google的Gdata、豆瓣的Open API、Amazon的S3

JAX-RS 简化REST应用的开发

1 RESTful Web Service

又称 RESTful Web API，是一个使用 HTTP 并符合 REST 原则的 Web 服务

2 JAX-RS

Java API for RESTful Web Services 旨在定义一个统一的规范，使得 Java 程序员可以使用一套固定的接口来开发 REST 应用，避免了依赖于第三方框架

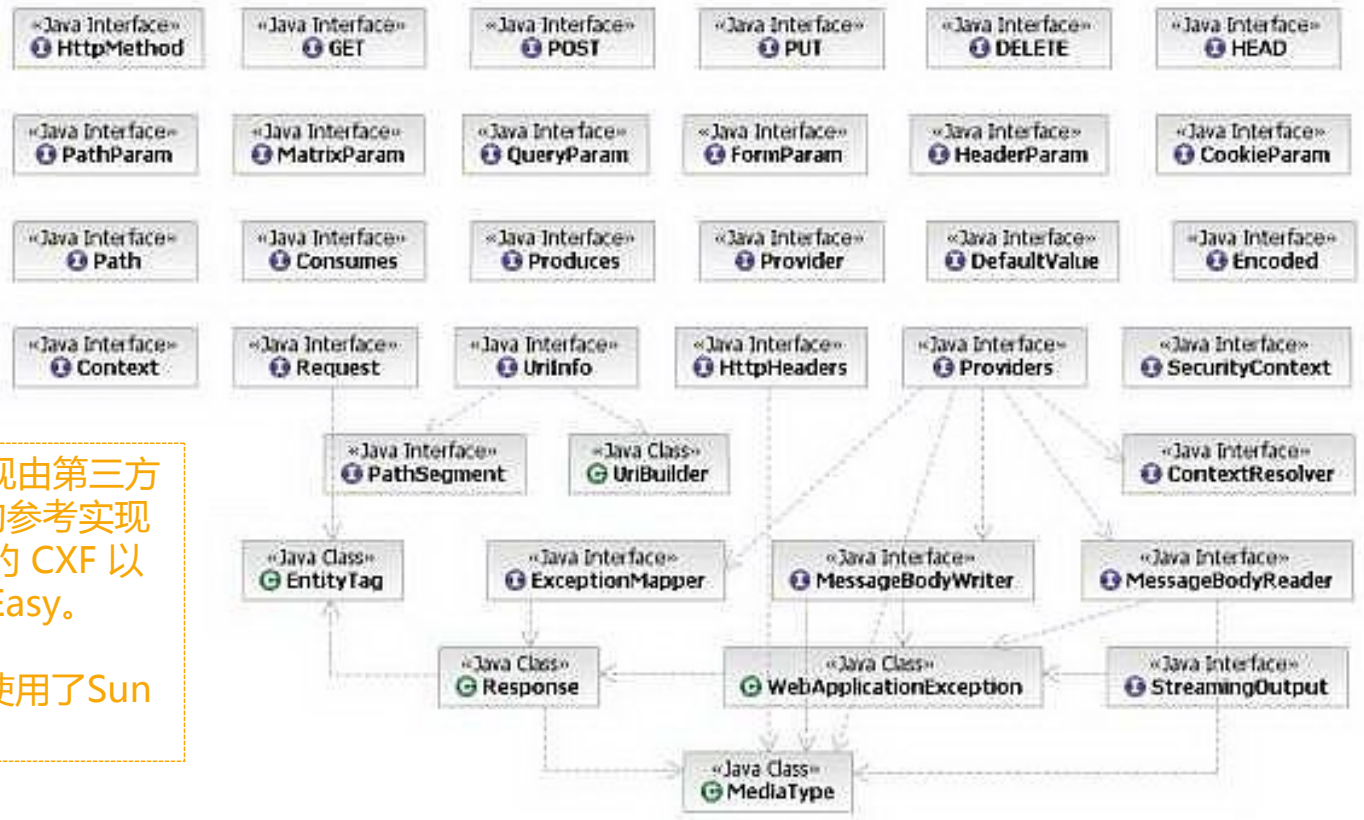
For Example

OpenDaylight使用了JAX-RS来简化REST API的开发

API

JAX-RS 定义的 API 位于 javax.ws.rs 包中

一些主要的接口、标注和抽象类如下图所示



JAX-RS 的具体实现由第三方提供，例如 Sun 的参考实现 Jersey、Apache 的 CXF 以及 JBoss 的 RESTEasy。

OpenDaylight中使用了Sun的Jersey

例子

一个可以阐述REST的简单例子

- 通过URL (http://www.opendaylight.org)， Client 向 http://www.opendaylight.org 发出请求
- http://www.opendaylight.org收到请求，回应首页给 Client、
- Client 又点击软件介绍（假设是 http://www.opendaylight.org/software 向 http://www.opendaylight.org发出software此篇文章的请求
- http://www.opendaylight.org收到请求，响应软件介绍文章内容给 Client
- Client 的通过 URI 来获取资源的具体象征 (Representational)。
- Client 取得这些具体象征使这些应用程序转变其状态 (以 浏览器而言，取得 HTML、CSS、JavaScript … 来生成界面)，随着不断取得资源的具体象征， Client 端不断地改变其状态，这样不断的反复 (iterations) 过程就是所谓的 Representational State Transfer。

Resource

Resource与 HTTP 方法的对应

资源	资源说明	GET	PUT	POST	DELETE
http://www.opendaylight.org/software/downloads	downloads是**一组**资源集合	**列出** 该组资源集合中每个资源的详细信息	**更新** 当前整组资源	**新增** 或附加一个新资源。该操作传回新资源的	**删除** 整组资源
http://www.opendaylight.org/software/downloads/hydrogen-base-10	downloads/hydrogen-base-10是**单个**资源	**取得** 指定的资源的详细信息	**更新** 或新增指定的资源	**新增** 或附加一个新元素	**删除** 指定的元素

Resource

Resource类和方法

■Web 资源作为一个 Resource 类来实现，对资源的请求由 Resource 方法来处理。Resource 类或 Resource 方法被打上了 Path 标注，Path 标注的值是一个相对的 URI 路径，用于对资源进行定位，路径中可以包含任意的正则表达式以匹配资源。和大多数 JAX-RS 标注一样，Path 标注是可继承的，子类或实现类可以继承超类或接口中的 Path 标注。

■Resource 类是 POJO，使用 JAX-RS 标注来实现相应的 Web 资源。

■Resource 类分为根 Resource 类和子 Resource 类，区别在于子 Resource 类没有打在类上的 Path 标注。Resource 类的实例方法打上了 Path 标注，则为 Resource 方法或子 Resource 定位器，区别在于子 Resource 定位器上没有任何 @GET、@POST、@PUT、@DELETE 或者自定义的 @HttpMethod。

Resource

Resource类和方法参数标注

JAX-RS 中涉及 Resource 方法参数的标注包括：

①@Path，标注资源类或方法的相对路径

②@GET，@PUT，@POST，@DELETE，标注方法是用的HTTP请求的类型

③@Produces，标注返回的MIME媒体类型

④@Consumes，标注可接受请求的MIME媒体类型

@PathParam，它用于将 @Path 中的模板变量映射到方法参数，模板变量支持使用正则表达式，变量名与正则表达式之间用分号分隔。标注方法的参数来自于HTTP请求的不同位置，来自于URL的路径

JAX-RS 规定 Resource 方法中只允许有一个参数没有打上任何的参数标注，该参数称为实体参数，用于映射请求体。

北向接口

OpenDaylight提供的北向接口

Topology REST APIs
Host Tracker REST APIs
Flow Programmer REST APIs
Static Routing REST APIs
Statistics REST APIs
Subnets REST APIs
Switch Manager REST APIs
User Manager REST APIs
Connection Manager REST APIs
Bridge Domain REST APIs

Topology REST APIs

可提供的服务

请求内容	请求方式	服务器响应
检测拓扑信息	GET	topology
获取UserLink信息	GET	list
设置UserLink	PUT	topologyUserLinkConfig

Topology REST APIs

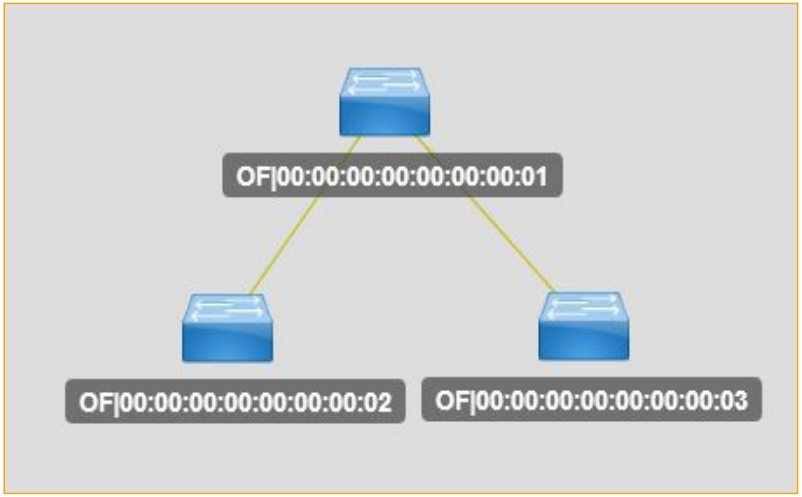
Topology的基本单元是：节点node，node的标识包括id和类型，头节点和尾节点连接器的连线组成边edge，edge和多个property的组合构成了具有实际属性的edgeProperties即有了一个具有实际特性的连接，若干edgeProperties的组合构成了Topology。

另外该APIs提供了TopologyUserLinkConfig组成的list，存储的是用户定义连接的配置信息。

Data Elements	Data Types	XML Elements	
		名称（类型）	Min/Max
edge	edge	tailNodeConnector	0/1
		headNodeConnector	0/1
edgeProperties	edgeProperties	edge (edge)	0/1
		properties/property	0/无限
list	topologyUserLinks	userLinks (topologyUserLinkConfig)	0/无限
node	node	type (string)	0/1
		id (string)	0/1
nodeConnector	nodeConnector	type (string)	0/1
		id (string)	0/1
		node ()	0/1
property	property		
topology	topology	edgeProperties (edgeProperties)	0/无限
topologyUserLinkConfig	topologyUserLinkConfig	status (string)	0/1
		srcNodeConnector (string)	0/1
		dstNodeConnector (string)	0/1
		name (string)	0/1

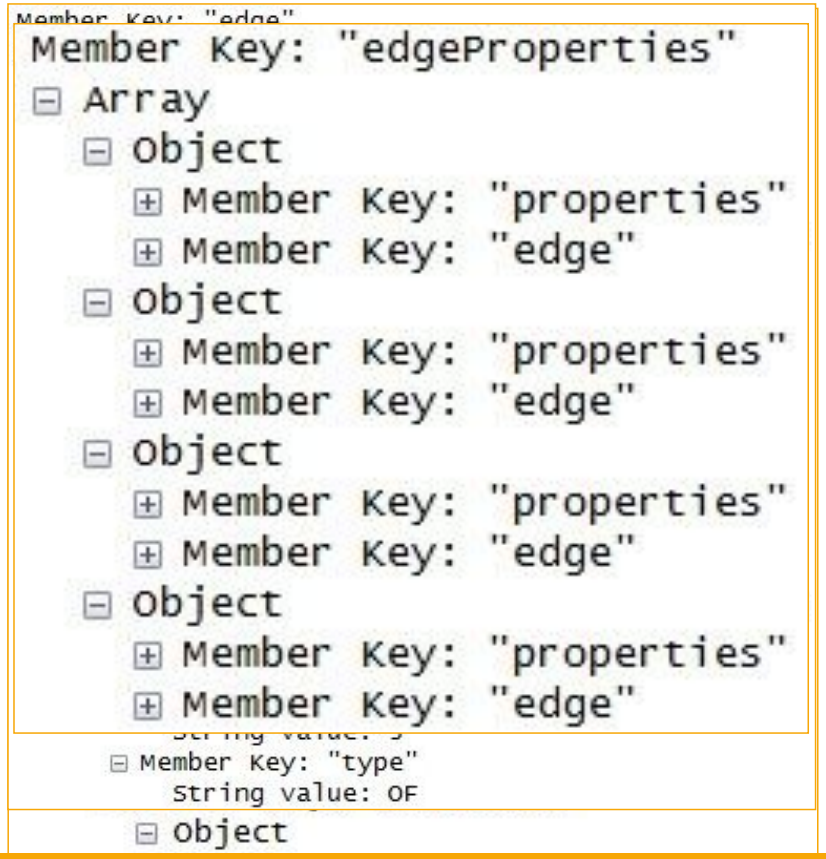
实验 Topology REST APIs实验解读

```
mn
--topo
tree,depth=2,fanout=2
--controller=remote,
ip=xxx.xxx.xxx.xxx,
port=6633
```



实验 Topology REST APIs实验解读（请求拓扑信息）

实验 Topology REST APIs实验解读（返回拓扑信息）



拓扑信息由连接组成，该拓扑的连接组成有：
(OF|2@OF|00:00:00:00:00:00:00:01
->OF|3@OF|00:00:00:00:00:00:00:03)

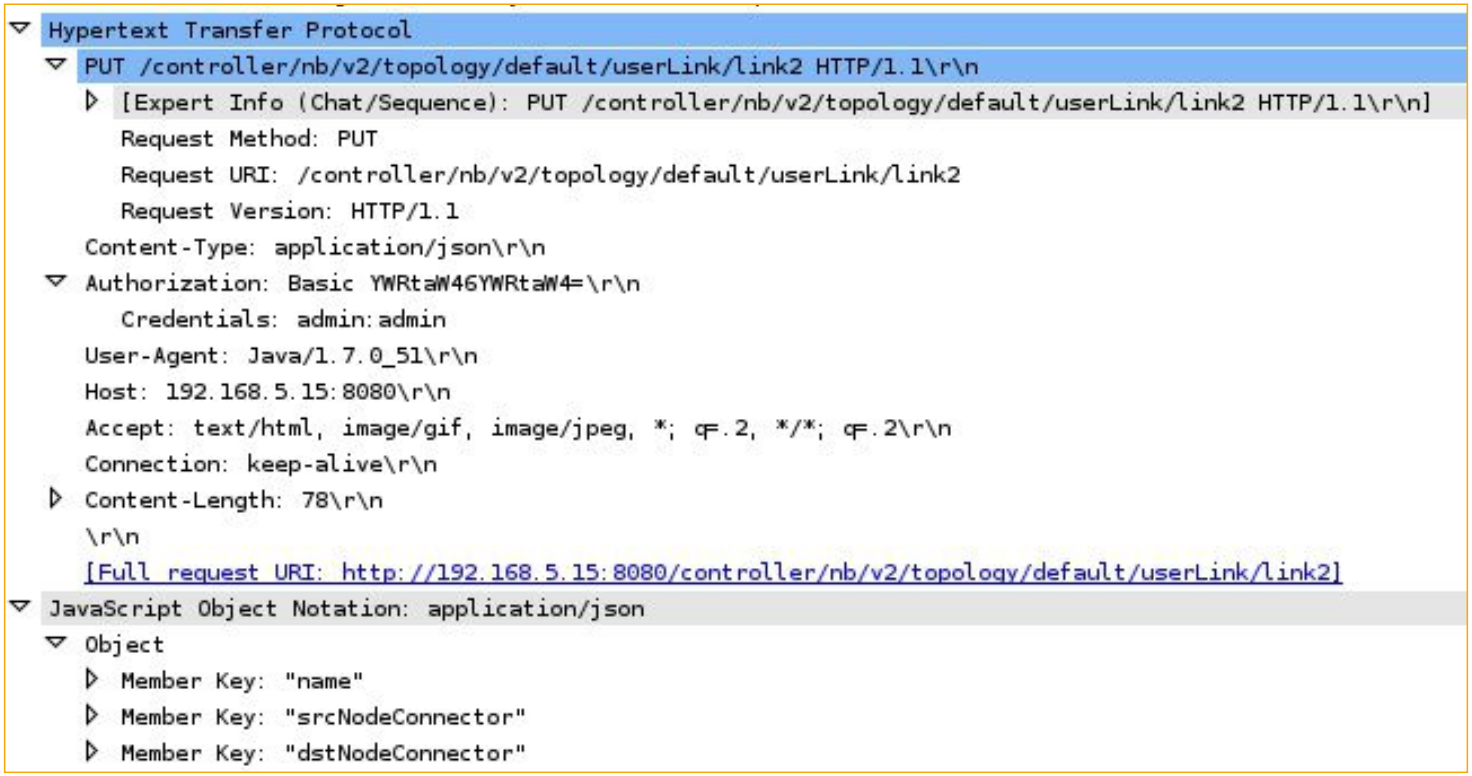
(OF|3@OF|00:00:00:00:00:00:00:03
->OF|2@OF|00:00:00:00:00:00:00:01)

(OF|3@OF|00:00:00:00:00:00:00:02
->OF|1@OF|00:00:00:00:00:00:00:01)

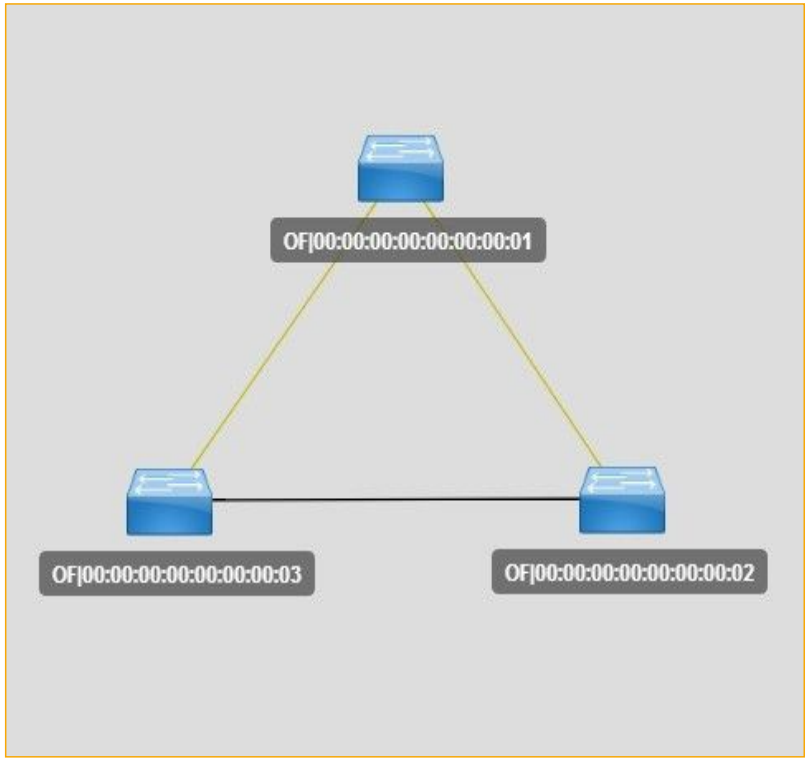
(OF|1@OF|00:00:00:00:00:00:00:01
->OF|3@OF|00:00:00:00:00:00:00:02)

其中箭头表示连接方向，@前面的数字表示交换机的端口，后面表示连接类型，后面的64位字符串是交换机的DPID。

实验 Topology REST APIs实验解读（设置UserLink信息）



实验 Topology REST APIs实验解读（获取拓扑信息）



(OF|2@OF|00:00:00:00:00:00:00:01
->OF|3@OF|00:00:00:00:00:00:00:03)

(OF|1@OF|00:00:00:00:00:00:00:02
->OF|2@OF|00:00:00:00:00:00:00:03)

(OF|3@OF|00:00:00:00:00:00:00:03
->OF|2@OF|00:00:00:00:00:00:00:01)

(OF|2@OF|00:00:00:00:00:00:00:03
->OF|1@OF|00:00:00:00:00:00:00:02)

(OF|3@OF|00:00:00:00:00:00:00:02
->OF|1@OF|00:00:00:00:00:00:00:01)

(OF|1@OF|00:00:00:00:00:00:00:01
->OF|3@OF|00:00:00:00:00:00:00:02)

REST API的思考

?

总结OpenDaylight所提供的REST API，并对其合理性进行分析

谢谢观赏

作者：冀烨

2014年3月20日

REST　OpenDaylight

未来网络创新研究院　SDN项目组