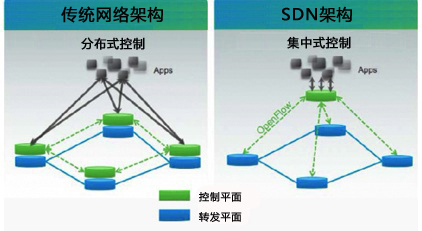
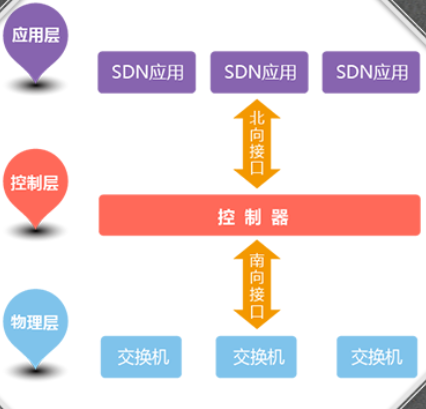
# SDN

## SDN意义

SDN（Software Defined Network，软件定义网络），是一种新型的网络架构，设计理念是将网络的控制平面（Controller）与数据转发平面（vSwitch）分离开来,从而实现网络流量的灵活控制。

[](http://s8.51cto.com/wyfs01/M02/44/86/wKioOVK3-LCjBSTgAAFb0WA01tY040.jpg)

## SDN架构



1. **物理层**：硬件设备层，专注于数据转发，主要部署SDN交换机，Open vSwitch
2. **南向接口**：物理层与控制器信号的传输通道，OpenFlow协议（设备状态、数据流表项、控制指令）
3. **控制器**：向下通过Openflow集中管理网络中所有设备，向上通过开放接口向应用层提供对网络资源的控制。主要包括开源的NOX，POX，Floodlight，Beancon，OpenDaylight，商用的Helios、SNAC
4. **北向接口**：控制器向上层业务提供的开放接口。
5. **应用层：**通过控制层提供的API对底层设备进行编程。

Hub：基于物理端口转发

L2Switch：基于MAC地址转发

Router：基于路由表转发

## OpenFlow意义

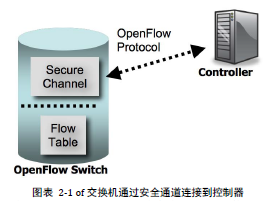
控制器：控制器知道所有网络信息，负责指挥设备如何工作。

交换机：交换机不知道任何网络信息，只会按照控制器的指挥工作。

SDN的网络设备之间不运行任何协议，网络设备的转发表由控制器配置生成。

控制器与网络设备之间需要OpenFlow协议来互相通信。

SDN是概念，OpenFlow是SDN的技术实现。

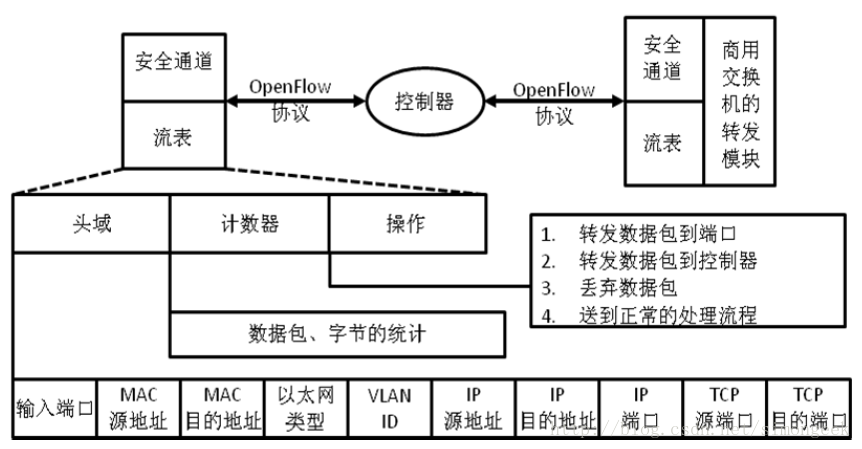


## OpenFlow结构

OpenFlow由流表（Flow Table）、安全信道（Secure Channel）和OpenFlow协议（OpenFlow Protocol）组成。

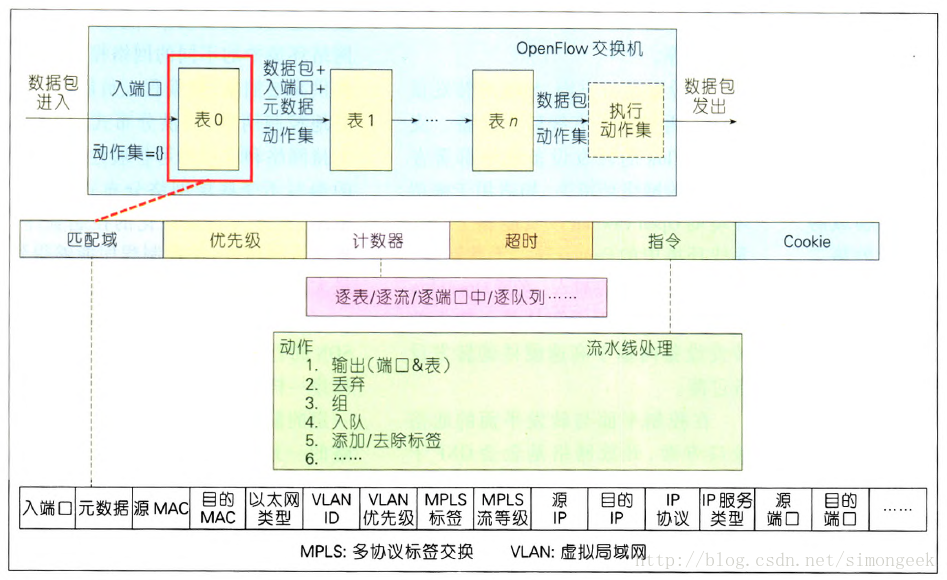
OpenFlow协议支持三种消息：controller-to-switch（控制器向交换机），asynchronous（交换机向控制器），symmetric。

Openflow交换机中的转发表称为流表（Flow table），表中包含多个流表项（Flow Entry）。流表项由头域、计数器、操作组成。头域是流表项的匹配标识，计数器是与该流表项匹配的统计数据，操作是与该流表项匹配后执行的操作。



|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ingress Port | Ether Source | Ether Dst | Ether Type | VLAN id | VLAN priority | IP src | IP dst | IP proto | IP ToS bits | TCP/UDP src port | TCP/UDP dst port |
| 交换机入端口 | 源MAC地址 | 目的MAC地址 | 以太网类型 | VLAN id | VLAN优先级 | 源IP | 目的IP | IP协议字段 | IP服务类型 | TCP/UDP源端口号 | TCP/UDP目的端口号 |

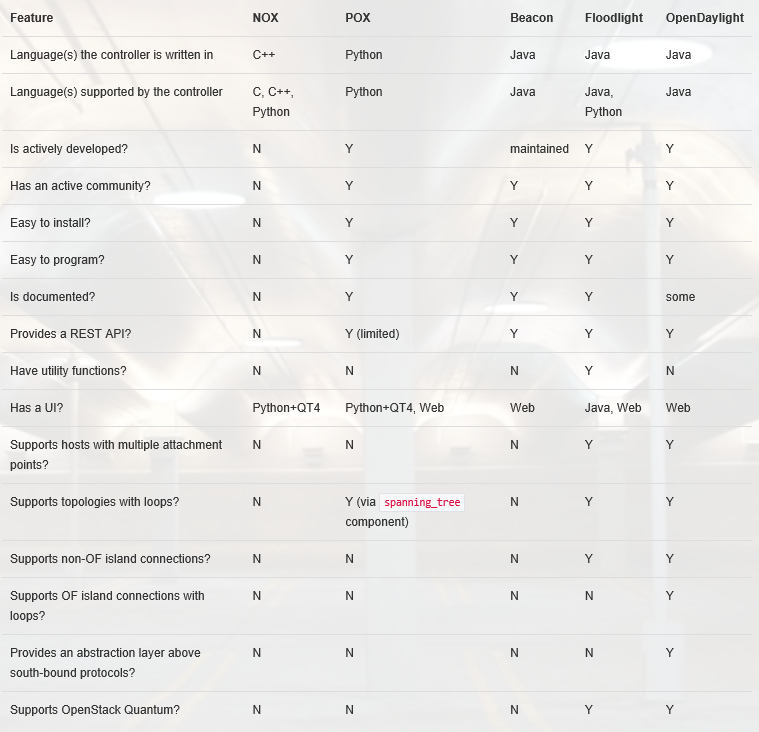
OpenFlow执行过程：





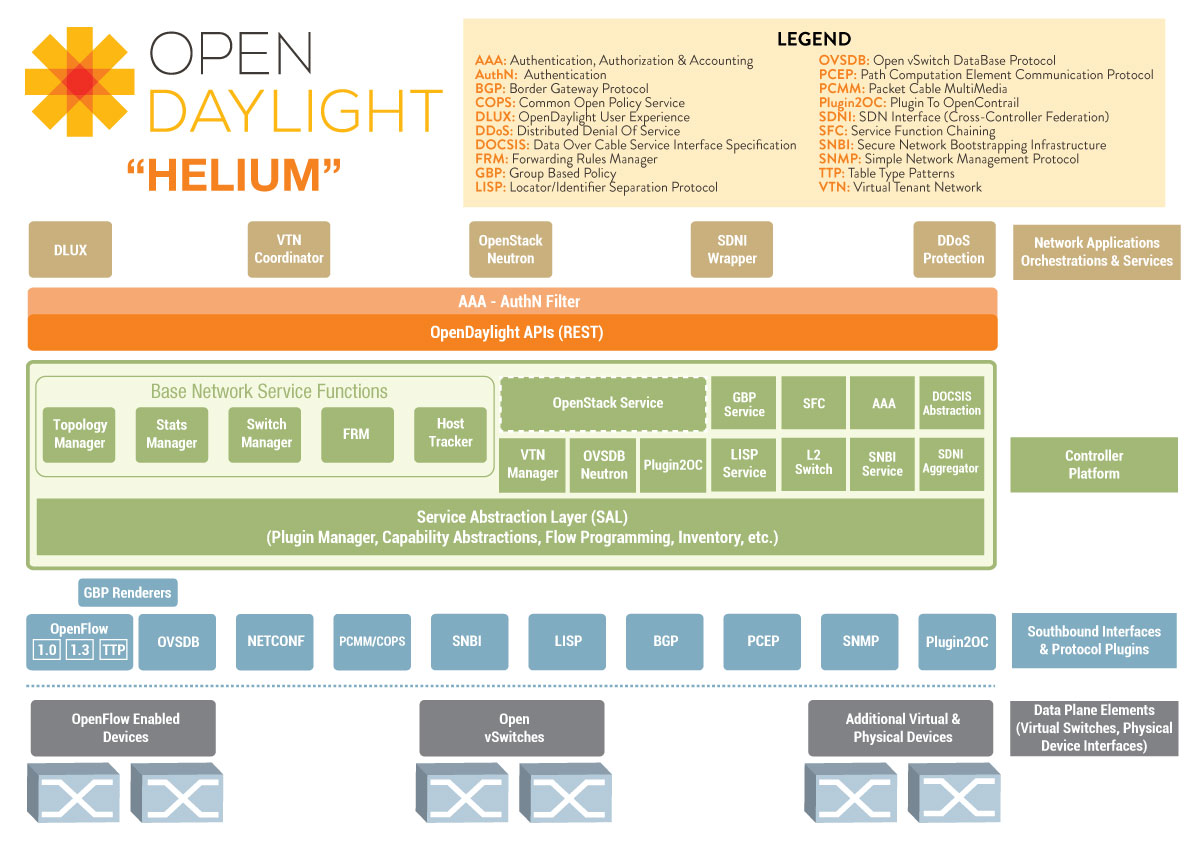
## 控制器对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 控制器 | 语言 | 平台 | 团队 | 特点 |
| NOX | Python/C++ | Linux | Nicira | 最早实现的控制器  无文档和UI  单线程操作  2010后已停止更新 |
| POX | Python | Linux | Nicira | 作为NOX的接班人，由同一组织开发  提供更友好的web-based GUI |
| Beacon | Java | Win/Mac/Linux | Standford | 易于开发和部署  模块化，代码风格好  OSGi导致UI运行困难  只支持星状拓扑  已停止更新 |
| FloodLight | Java | Win/Mac/Linux | Big switch | 作为Beancon的接班人  跨平台  支持循环拓扑  提供web-based UI和Java-based GUI两个UI，论坛活跃  文档丰富  目前最成熟的OpenFlow F/OSS Controller |
| Opendaylight | Java | Linux | Cisco | Linux 基金会推出的工业级项目  代码风格比floodlight好  主要是Cisco维护  在早期发展中 |



# OpenDaylight

SDN控制器，旨在加速Software-Defined Networking (SDN) 和Network Functions Virtualization (NFV)，开放平台，网络可编程。



SAL（Service Abstraction Layer）为南向接口屏蔽具体协议，所有协议如OpenFlow通过SAL与控制器交互。

Topology Manager管理保存设备信息，可达性、性能等。ARP handler、Host Tracker、Device Manager、Switch Manager帮助生成拓扑数据库。

OpenDaylight有两类接口：

MD SAL：Nodes（交换机信息），Yang UI，Topology

AD SAL：Connection manager，Container，Network，Flows

## 安装

### 准备工作

依赖软件：**jdk 1.7，maven 3.1.1以上**，git

**Ubuntu 14.04：**

sudo apt-get install openjdk-7-jre

sudo apt-get install openjdk-7-jdk

sudo apt-get install git-core

sudo apt-get install wireshark

**CentOS 7:**

yum install java-1.7.0-openjdk

yum install java-1.7.0-openjdk-devel

yum install git

yum install wireshark

**重要：关闭防火墙**

systemctl stop firewalld

tar zxvf apache-maven-3.2.3-bin.tar.gz

mv apache-maven-3.2.3-bin maven

vi ~/.bash\_profile

export JAVA\_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.79-2.5.5.1.el7\_1.x86\_64

export MAVEN\_HOME=/root/maven

export MAVEN\_OPTS='-Xmx1048m -XX:MaxPermSize=512m'

PATH=${JAVA\_HOME}/bin:${MAVEN\_HOME}/bin:$PATH:$HOME/bin

export PATH

source ~/.bash\_profile

java -version

**重要：jdk用1.7版本，用1.8会抛异常NullPointerException**

mvn -v

**重要：maven版本3.1.1以上**

**重要：创建ODL的仓库**

cp -n ~/.m2/settings.xml{,.orig}

wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml > ~/.m2/settings.xml

### 直接安装

下载预编译包

unzip distribution-karaf-0.2.0-Helium.zip

cd distribution-karaf-0.2.0-Helium

./bin/karaf

执行feature:install <feature1-name> <feature2-name> ... <featureN-name>安装组件

|  |  |  |  |
| --- | --- | --- | --- |
| 组件名 | 描述 | Karaf 特征名 | 兼容性 |
| BGPCEP | 支持BGP LS PCEP | odl-bgpcep-all | all |
| Defense4All | 启动DDoS检测和保护 | n/a | all |
| Group Based Policy | Endpoint注册和策略库REST APIs | odl-groupbasedpolicy-ofoverlay | self-all |
| L2 Switch | 提供OpenFlow交换机的两层协议，主机跟踪 | odl-l2switch-switch-ui | self-all |
| LISP Flow Mapping | LISP控制面板服务，包括映射系统REST API和LISP协议 | odl-lispflowmapping-all | all |
| MD-SAL Clustering | 特殊兼容性，必须第一个安装 | odl-mdsal-clustering | special |
| Netconf over SSH | 提供SSH到Netconf-enabled设备的支持 | odl-netconf-connector-ssh | all |
| DLUX Web Interface | DLUX需要依赖来自OpenFlow Flow Programming和L2 Switch组件的信息进行绘制，以显示网络拓扑、流数据、主机位置等信息。安装其他组件时默认安装DLUX |  | all |
| RESTCONF API Support | 所有需要REST API的组件都会自动加载RESTCONF API | odl-restconf | all |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* all是可以与其他任何组件一同运行
* self+all是可以与all的组件一起运行，但不能和self+all的组件运行

feature:list 查看可选组件

feature:list -I 查看已安装组件

opendaylight-user@root> feature:install odl-dlux-all odl-restconf odl-l2switch-switch

**安装DLUX，就可登陆网站**，**界面有Topology，restconf提供授权功能**

opendaylight-user@root> feature:install odl-base-all odl-aaa-authn odl-restconf odl-adsal-northbound odl-mdsal-apidocs odl-l2switch-switch

opendaylight-user@root> feature:install odl-ovsdb-northbound

**界面会出现Nodes，Connection Manager、Network、Flows，Yang UI页签**

<http://localhost:8181/dlux/index.html>

用户名密码都是admin

**问题1：error executing command: NullPointerException**

用JDK 1.7 不要用8

**问题2：安装出现问题后，清除数据**

$ opendaylight-user@root>logout

$ rm -rf ~/distribution-karaf-0.2.0-Helium/data

$ ./bin/karaf clean

### helium稳定版本

#### 源码编译安装

首先要配置编译环境

<https://wiki.opendaylight.org/view/GettingStarted:Development_Environment_Setup#Edit_your_.7E.2F.m2.2Fsettings.xml>

git clone <https://git.opendaylight.org/gerrit/p/controller.git>

cd controller

git tag

git checkout release/helium 切换到helium稳定版

为避免编译过程中test的编译错误，使用mvn clean install –DskipTests

cd ~/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/

./run.sh

访问http://ip:8080，用户名密码都是admin

**问题1：pom.xml的relativePath为空**

[ERROR]   The project org.opendaylight.controller:sal-binding-broker-impl:1.2.0-SNAPSHOT (/root/OpenDayLight\_wiki/controller/opendaylight/md-sal/l-binding-broker/pom.xml) has 2 errors  
[ERROR]     'dependencies.dependency.version' for org.eclipse.xtend:org.eclipse.xtend.lib  
[ERROR]     'build.plugins.plugin.version' for org.eclipse.xtend:xtend-maven-plugin must be a valid version but is '${xtend.version}'.

.m2没有设置settings.xml文件

wget -q -O - https://raw.githubusercontent.com/opendaylight/odlparent/master/settings.xml > ~/.m2/settings.xml

**问题2：Failed to execute goal org.apache.maven.plugins:maven-enforcer-plugin**

maven版本低于3.1.1

**问题3：编译报错中央仓库 not found repo.maven.org/maven2**

DNS设置为8.8.8.8

#### 导入Eclipse

JDK1.7

修改eclipse.ini文件

--launcher.XXMaxPermSize

512M

-Xmx2048m

下载eclipse-java-luna-SR1a-win32-x86\_64，解压启动eclipse

luna版本自带m2e插件，不用安装，其他版本需安装m2e，推荐1.4和1.2，1.3有bug

1. **Xtend**

Help->Install New Software

Name：Xtend

Location：http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases

取消Show only the latest versions of available software和Group items by category

选择Xtend SDK 2.4.3.XXX和Xtend M2E extendsions 2.4.3.XXX

单击Next，OK，直至重启Eclipse

1. **pde**

如果eclipse不是RCP版本，需安装PDE，否则在安装tycho时会报错

Help->Install New Software

Name：Luna

Location：http://download.eclipse.org/releases/luna

选择Eclipse Plug-in Development Environment，Next，OK，重启

1. **m2eeclipse-tycho**

Help->Install New Software

Name：Tycho

Location：http://repo1.maven.org/maven2/.m2e/connectors/m2eclipse-tycho/0.7.0/N/0.7.0.201309291400

全选，Next，OK，重启

1. **build-helper-maven**

Windows->Preferences->Maven->Discovery->Open Catalog

选择buildhelper，Finish->Accept->Finish->重启

1. **Groovy**

Help->About Eclipse->版本号4.4

Help->Install New Software

Name：Groovy

Location：http://dist.springsource.org/release/GRECLIPSE/e4.4

如果版本号是4.3，则把Location的4.4改为4.3

选择Groovy-Eclipse和m2e Configurator for Groovy-Eclipse，Next，Accept，Finish，重启

1. **其他配置**

Window->Preferences->Java->Editor->Save Actions

选中"Perform the selected actions on save" 和"Additional actions"，不选其他

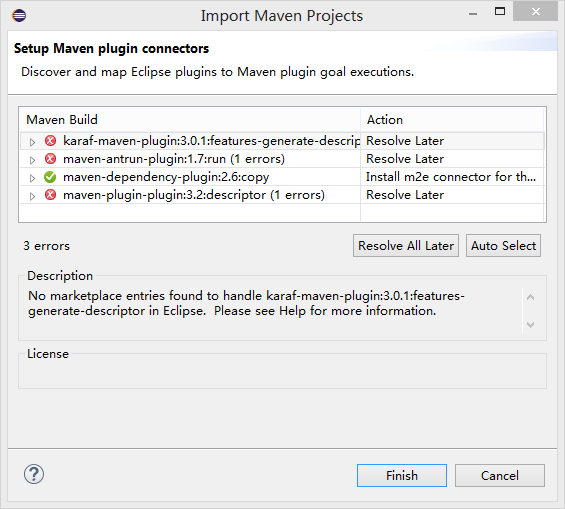
单击Configure…，选中Code Organizing中的Remove trailing whitespace和All lines，其他页签的全部取消，底部显示1 of 27 save actions activated，OK。

1. **导入源码**

下载源码，File->Import->Maven->Existing Maven Projects

选择controller根目录，所有项目，Finish

由于缺少maven-plugin-plugin, karaf-maven-plugin 和maven-antrun-plugin插件，会导致编译错误5000多个，所以Eclipse无法直接起服务，需要远程调试。



#### 远程调试

按照源码安装方式在远程以调试方式启动控制器

./run.sh -debug

Eclipse->Run->Debug

Remote Java Application->Connection type选择Standard(Socket Attach)，输入远程IP和端口8000

Source页签单击ADD，Java Projects，选择所有项目，单击Apply，单击debug进行调试

Window->Open Perspective->Debug，打开Debug视图

选择web项目->DaylightWeb->login方法上打断点

然后打开 <http://ip:8181>，登陆，在断点处停止

如果要调试启动过程

./run.sh -debugsuspend

### 最新源码

#### 源码编译安装

最新版本的ODL将插件从controller项目中分离出来，形成openflowplugin

git clone <https://git.opendaylight.org/gerrit/p/controller.git>

git clone <https://git.opendaylight.org/gerrit/p/openflowplugin.git>

git clone <https://git.opendaylight.org/gerrit/p/openflowjava.git>

git clone [https://git.opendaylight.org/gerrit/p/ integration.git](https://git.opendaylight.org/gerrit/p/%20integration.git)

cd integration/

mvn clean install -DskipTests

find . -name karaf

./karaf/opendaylight-karaf/target/assembly/bin/karaf

opendaylight-user@root> feature:install odl-base-all odl-aaa-authn odl-restconf odl-adsal-northbound odl-mdsal-apidocs odl-l2switch-switch

opendaylight-user@root> feature:install odl-ovsdb-northbound

cd controller/

mvn clean install -DskipTests

cd openflowplugin/

mvn clean install -DskipTests

cd openflowjava/

mvn clean install -DskipTests

**问题：java.lang.NoClassDefFoundError: org/apache/commons/lang/StringUtils**

删除 .m2

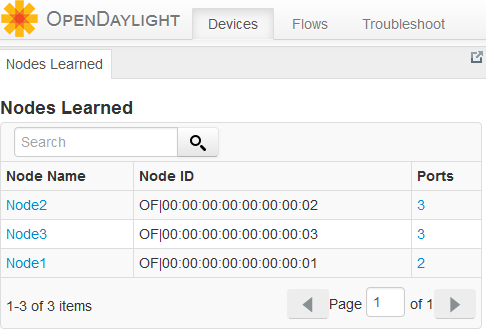
先执行mvn generate:create 将正确的包下载下来

再执行 mvn clean install -DskipTests

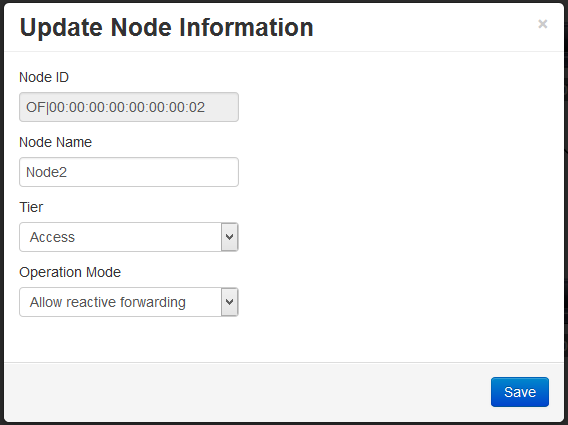
## 网站使用

### Devices

1. **Nodes Learned**



与ISwitchManager有关，所有交换机列表，单击Node Name，修改交换机信息



* 1. **Tier**

**Hierarchical internetworking model，Cisco提出的三层网络模型**

Unknown：未知

Access：接入层，终端或服务器都连接在Access层，通常为商业交换机平台，企业主要考虑的成本所在

Distribution：分发层，负责路由选择、过滤、QoS、WAN连接

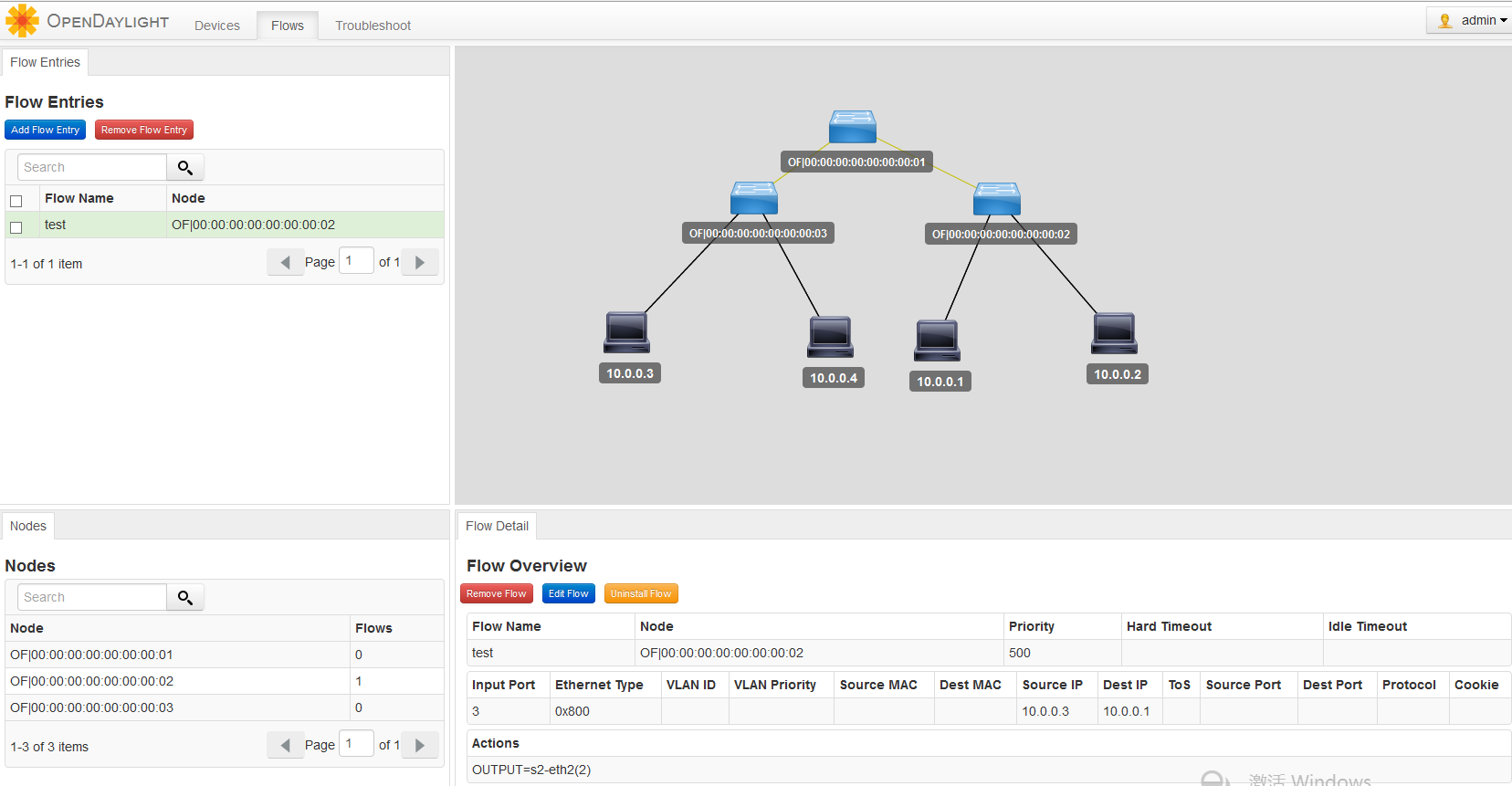
Core：核心层，负责为distribution层提供高速、冗余的传输服务，核心网络设备管理最高速的连接（10Gb以太网）

* 1. **Operation Mode，流表是否预设**

Allow reactive forwarding：如果没有匹配的流规则，则将请求封装成OFPpacket-in给控制器，缺点是有延迟

Proactive forwarding only：控制器预先将所有可能匹配到的流表放入交换机，无延迟

### Flows



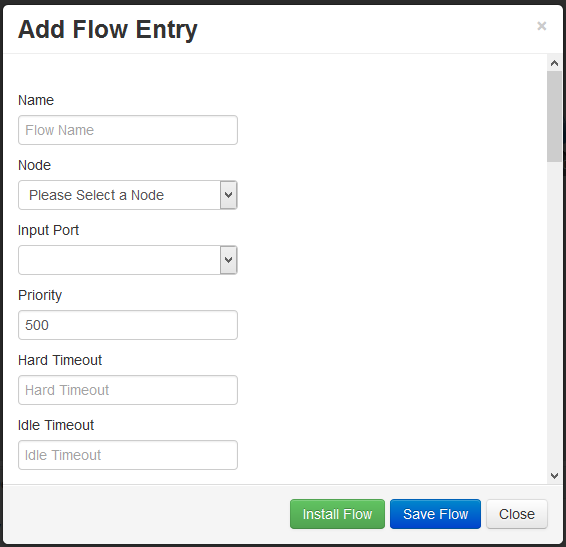
管理流表页面，包括创建、修改、删除、下发流表项，与IForwardingRulesManager有关

Flow Entries显示所有流表项，单击Add Flow Entry新增流表项

单击某一行流表项，Flow Overview中显示该流表项详细信息，并可删除、修改、回收

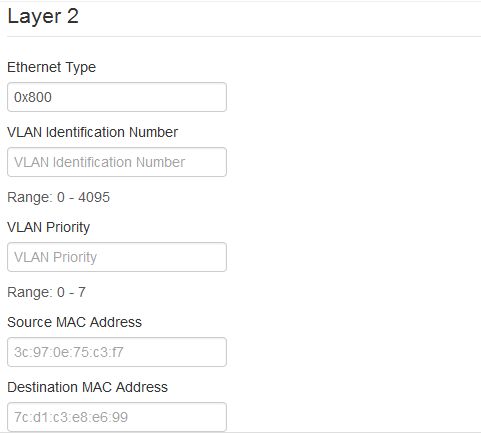
Nodes中显示每个交换机上有几条流表项

1. **新增流表**



Idle Timeout：该流表项多久（seconds）没有被匹配到，就从流表中删除

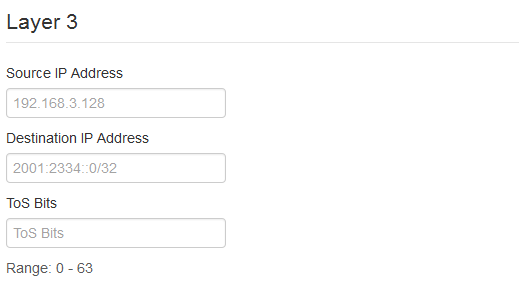
Hard Timeout：该流表项多久后（seconds）从流表中删除，即使中间被匹配到。



二层数据链路层匹配信息，包括设置VLAN和MAC

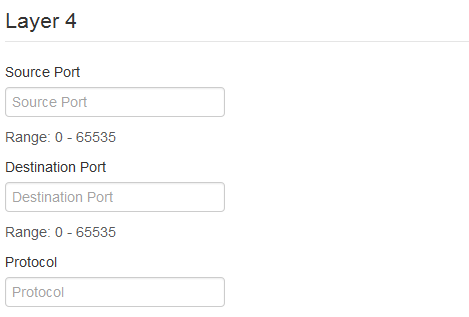
Ethernet Type：两字节字段，说明以太网帧中封装的类型

0x0800：IPv4，0x8100：VLAN，0x86DD：IPv6



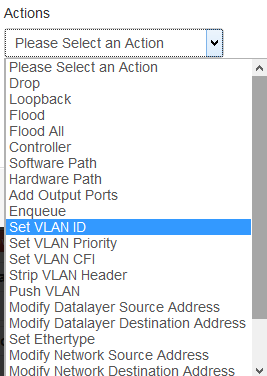
三层网络层匹配信息，包括设置IP和服务质量

ToS Bits：服务质量等级，范围0-63



四层传输层匹配信息，包括设置端口号和协议

Protocol：应用层协议，如FTP，SSH



### Troubleshoot

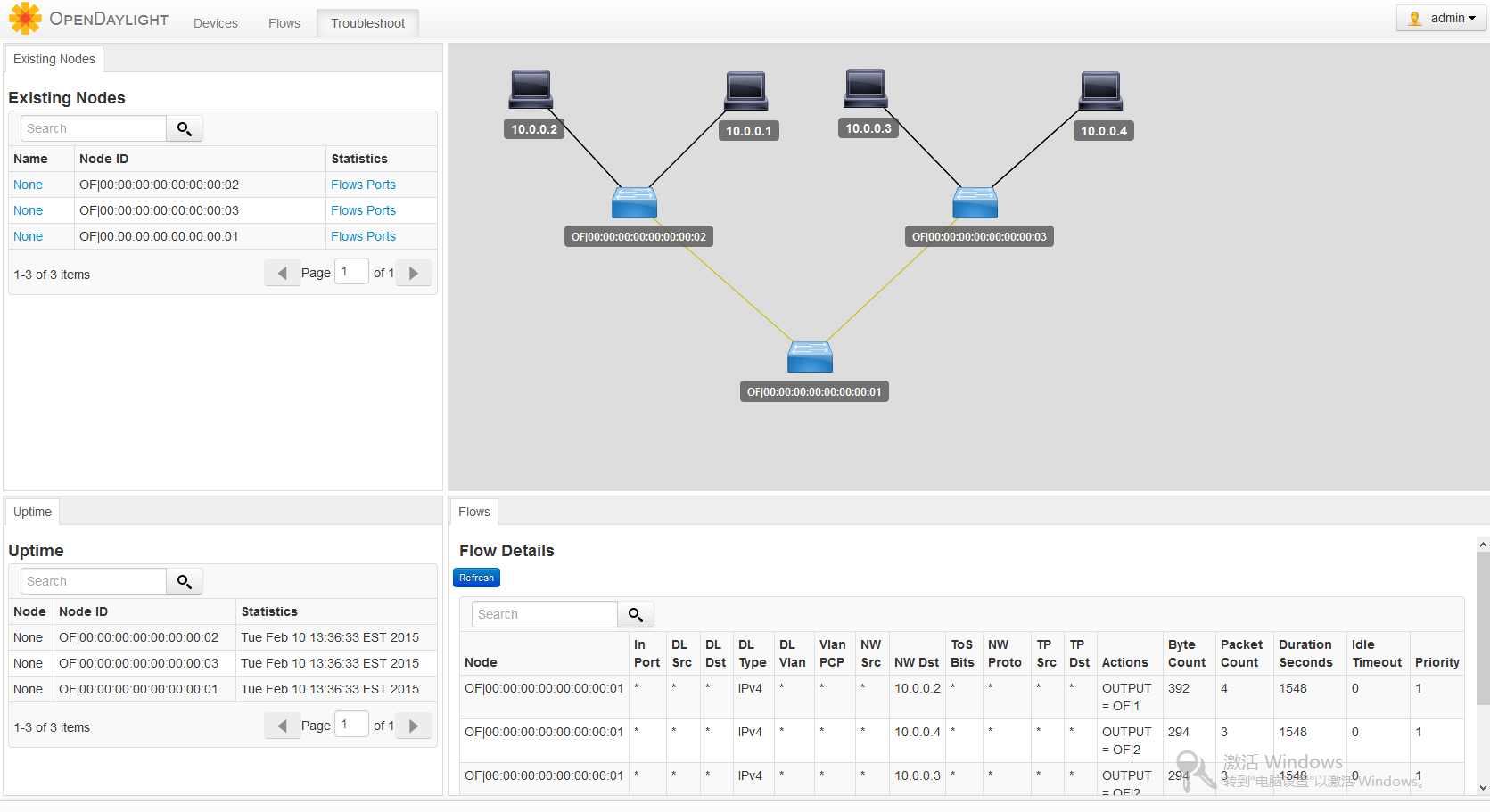
显示交换机统计信息，与IStatisticsManager有关

Existing Nodes显示所有交换机

单击某个交换机的Name，可查看该交换机的软硬件信息。

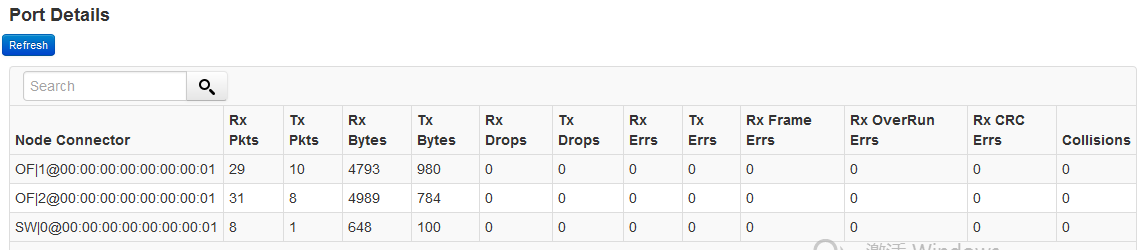
单击某个交换机的Flows，可查看该交换机上的所有流表的详细信息。

单击某个交换机的Ports，可查看该交换机上的所有端口的统计信息。



|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 数据链路层 Data Link Vlan | | | | | 网络层Network | | | | 传输层Transport | |
| Standard | | | VLAN | |
| Node | In Port | DL Src | DL Dst | DL Type | DL Vlan | Vlan PCP | NW Src | NW Dst | ToS Bits | NW Proto | TP Src | TP Dst |
| 交换机地址 | 网卡端口号  Ingress Port | 源MAC地址 | 目的MAC地址 | 数据链路层类型 | Vlan |  | 源IP | 目的IP | 保留字段  Type of Service | 网络层协议 | 源端口号 | 目的端口号 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Action | Byte Count | Packet Count | Duration Seconds | Idle Timeout | Priority |
| 操作 | 该流表项处理的字节数 | 该流表项处理的包个数 | 延迟秒数 | 空闲超时 | 优先级 |



## 系统架构

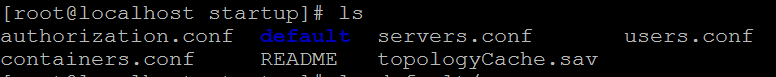
### 关键概念

**Container是网络域，ODL可以创建多个网络域，即多个Container，一个Container中包括了组成一个网络域的所有控制器、交换机和端口。**

默认Container为default，其相关配置保存在configuration/startup/default目录下，如果新建Container名为test，则相关配置保存在configuration/startup/test目录下。

**ODL的模块分为与Container相关的Container模块，如SwitchManager、TopologyManager、ForwardingRulesManager；与Container无关的Global模块，如UserManager，ClusterManager。**

ODL的配置分为与Container相关的配置，存放在以容器命名的目录下，如configuration/startup/default；与Container无关的配置存放在配置根目录下，如configuration/startup

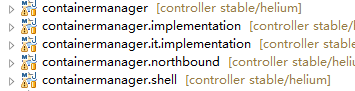


### 模块结构

一个模块大致分为6个项目：

* api：根据OSGi设计原理，对外发布的接口须单独列为一个项目，可选。
* implementation：根据OSGi设计原理，接口具体实现成为一个项目，必须。
* integrationtest：集成测试，可选。
* shell：根据是否需要实现Karaf Shell Console的命令，可选。
* northbound：根据是否需要提供北向接口，可选，保存在northbound目录中。





### 所用技术

**体系结构**

maven，OSGi

（Karaf）

**北向接口northbound**

Jersey

（SOAP，WSDL，RESTful，JAX-RS，JAX-WS，JAXB，Jersey，enunciate）

**南向接口**

Netty

Web服务

Spring

（SpringMVC，SpringSecurity）

**集群服务**

Infinispan

(JGroup)

**未完**

* maven：项目管理及自动构建工具，ODL利用各种plugin实现具体功能。
* OSGi：采用OSGi体系结构，做到了功能的隔离，解决了可扩展性、热部署等问题。
* Inﬁnispan ：用开源的数据网格平台实现Controller的集群。
* Netty：南向接口使用Netty来管理底层的并发IO。
* Jersey：北向接口使用Jersey提供REST接口。
* Spring：Web网站使用Spring MVC转发处理请求，Spring Security管理用户权限。
* SAL(Service Abstract Layer)：整个架构中引入了业务抽象层（思科的贡献），使得上层（北向）和下层（南向）之间的调用相互隔离，这个设计模式中的Dependence Inversion Principle 原则一致。
* MD(Model Drive)：使用Yang工具，使用业务模型驱动来设计接口、实现业务功能，根据yang文件，Yang工具直接生成业务管理的“骨架”，使开发者真正专注于具体业务。

### pom解析

#### 系统POM

controller/pom.xml：配置了整个项目的架构，指出commons.parent是父POM

commons.parent：设置scm（软件配置管理），distributionManagent（分发管理）

commons.opendaylight：**核心POM**，设置所有模块的版本，管理所有dependency和plugin的配置，设置reporting（site），设置repositories（dependency），设置pluginRepository（plugin），设置distributionManagement（deploy）。

commons.parent

controller.releasepom

commons.opendaylight

所有其他模块，如sal

解析模块

继承

继承

#### checkstype POM

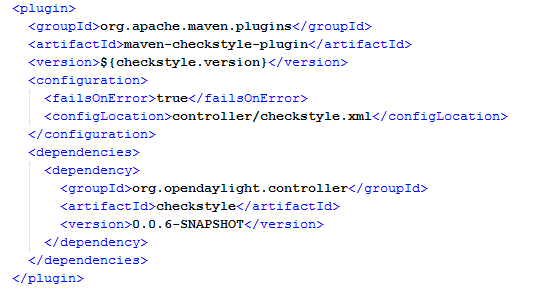
maven-checkstyle-plugin不支持报告聚合，用户无法在聚合项目的报告中得到所有模块的CheckStyle报告，聚会项目的配置方法如下：

1. 创建一个包含checkstyle规则文件的模块

checkstyle模块中定义maven-checkstyle-plugin所需的规则文件checkstyle.xml。

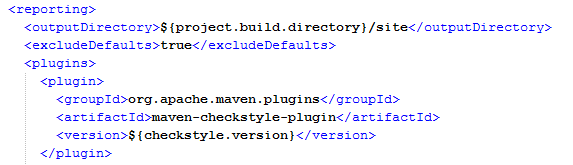
1. 在聚合模块的编译中配置maven-checkstyle-plugin依赖上面的模块

commons.opendaylight模块的pom.xml的<build>标签中配置maven-checkstyle-plugin依赖于checkstyle模块，并且配置文件路径为controller/checkstyle.xml。



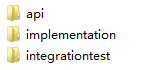
1. 在聚合模块的报告中配置maven-checkstyle-plugin

commons.opendaylight模块的pom.xml的<reporting>标签中配置maven-checkstyle-plugin



### OSGi设计

以switchmanager模块为例，根据OSGi设计原理，需对外发布的接口（Export-Package）都放在api目录中，独立成一个模块。接口实现放在implementation目录中。集成测试放在integrationtest目录中。



1. **MANIFEST.MF创建**

ODL基于OSGi实现系统架构，由pom.xml中org.apache.felix:**maven-bundle-plugin**构建MANIFEST.MF文件。

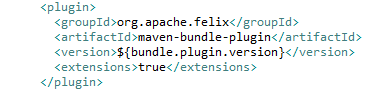
父commons.opendaylight/pom.xml中只定义了maven-bundle-plugin的版本和extensions。

extensions：是否加载plugin的扩展，默认为false。

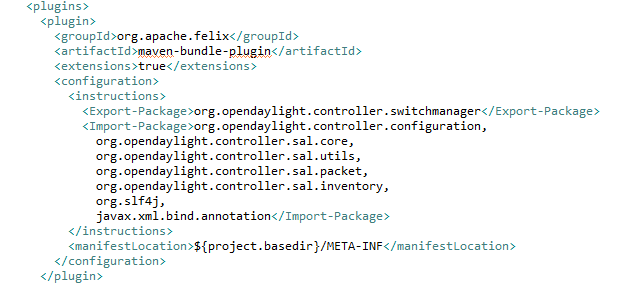
inherited：该pom是否被子pom继承，默认为true。

configuration：设置plugin内部属性。

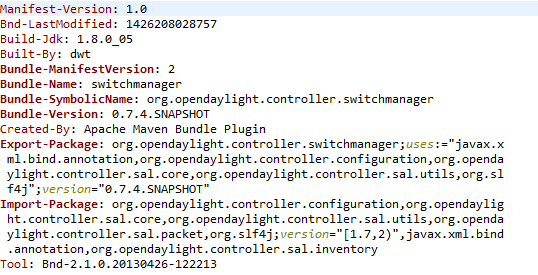
http://maven.apache.org/pom.html



子switchmanager/pom.xml中，指明该模块所有导入包（Import-Package），导出包（Export-Package），发布路径（Web-ContextPath）



执行maven install后创建的MANIFEST.MF如下：Created-By：Apache Maven Bundle Plugin



1. **ComponentActivatorAbstractBase**

Activator实现BundleActivator接口，用于接收bundle启动和停止事件，须放在implementation模块中，而不是api模块中（无需对外发布，并且需要通过BundleContext注册实例）。

**ComponentActivatorAbstractBase：Activator基础类，实现BundleActivator和IContainerAware接口，用于注册bundle中的全局实例和容器实例**

DependencyManager dm;

ConcurrentMap<ImmutablePair<String, Object>, Component> dbInstances：保存容器相关的实例，String是containerName，Object是实例

ConcurrentMap<Object, Component> dbGlobalInstances：保存容器无关的实例，Object是实例

**在bundle start时创建，stop时销毁，实例放入dbGlobalInstances中，与getImplementations()不同的是，其与容器无关**

protected Object[] getGlobalImplementations(){}

**为某个全局实例配置其依赖，依赖注入的关键所在**

protected void configureGlobalInstance(Component c, Object imp){

}

**在容器创建时创建，容器销毁时销毁，实例放入dbInstances中**

public Object[] getImplementations(){}

**为某个容器实例配置其依赖，依赖注入的关键所在**

public void configureInstance(Component c, Object imp, String containerName){}

BundleActivator.start模块启动

getGlobalImplementation获得全局实例

configureGlobalInstance配置全局实例，将其放入dbGlobalInstance

将Component放入DependencyManager中，启动过程结束

containerCreate容器创建

getImplementations获得容器实例

configureInstance配置容器实例，将其放入dbInstances中

将Component放入DependencyManager中，容器创建过程结束

Bundle创建成功后由于其继承自IContainerAware，因此OSGi发现其实现，将其加入ContainerManager的iContainerAware中，调用setIContainerAware

### northbound设计

#### 创建北向接口项目

利用mvn archetype:generate创建北向接口项目

在controller/opendaylight/northbound目录下执行mvn install

查看本地仓库中是否存在org/opendaylight/controller/app-northbound/0.0.4-SNAPSHOT

mvn archetype:generate -DarchetypeGroupId=org.opendaylight.controller -DarchetypeArtifactId=app-northbound -DarchetypeVersion=0.0.4-SNAPSHOT -DgroupId=org.opendaylight.controller -DartifactId=example -Dversion=0.0.1-SNAPSHOT

或者mvn archetype:generate -DarchetypeCatalog=local，选择app-northbound的编号，输入groupId，artifactId和version

#### jersey

**1）commons.opendaylight中pom.xml配置jersey**

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-client</artifactId>

<version>${jersey.version}</version>

</dependency>

<!-- Jersey for JAXRS -->

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-core</artifactId>

<version>${jersey.version}</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-server</artifactId>

<version>${jersey.version}</version>

</dependency>

<dependency>

<groupId>com.sun.jersey</groupId>

<artifactId>jersey-servlet</artifactId>

<version>${jersey-servlet.version}</version>

</dependency>

**2）若使用Jersey，须在web.xml中配置ServletContainer并设置初始化参数，以提供额外配置信息。有两种初始化方式：**

**1、直接设置参数，比如包含资源的package路径**

<servlet>

<servlet-name>Jersey REST Service</servlet-name>

<servlet-class>

com.sun.jersey.spi.container.servlet.ServletContainer

</servlet-class>

<init-param>

<param-name>com.sun.jersey.config.property.packages</param-name>

<param-value>sample.hello.resources</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>Jersey REST Service</servlet-name>

<url-pattern>/rest/\*</url-pattern>

</servlet-mapping>

**2、实现Application接口，利用Application提供额外的元数据，比如resource classes，Providers，properties。其中getClasses()返回资源类。getSingletons()返回资源实例。**

<servlet>

<servlet-name>JAXRSSwitchManager</servlet-name>

<servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>org.opendaylight.controller.northbound.commons.NorthboundApplication</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>JAXRSSwitchManager</servlet-name>

<url-pattern>/\*</url-pattern>

</servlet-mapping>

1. **NorthboundApplication继承自javax.ws.rs.core.Application，其覆盖getClasses()方法，通过Bundle Scanner Service(模块扫描服务)扫描所有@Path注释类，返回被JAXRS处理的类。**

**getSingletons()返回的实例必须以@Provider注释，且实现以下接口之一的实例：**

|  |  |
| --- | --- |
| ContextResolver<T> | 提供内容信息的Provider，可通过@Context注入 |
| ExceptionMapper<E extends Throwable> | 将Java Exceptions映射为Response的Provider |
| MessageBodyReader<T> | 将输入流转换为Java对象的Provider |
| MessageBodyWriter<T> | 将Java对象转换为输出流的Provider |
| Providers | 用于提供以上四种实例的Provider |

#### enunciate

**1）父模块commons.opendaylight的pom.xml中定义了maven-enunciate-plugin用于加强WebService**

<build>

<pluginManagement>

<plugins>

**//设置所有北向接口子模块将继承以下配置**

<plugin>

<groupId>org.codehaus.enunciate</groupId>

<artifactId>maven-enunciate-plugin</artifactId>

<version>${enunciate.version}</version>

<configuration>

//**要求所有子项目在根目录中创建enunciate的配置文件**

<configFile>enunciate.xml</configFile>

</configuration>

<dependencies>

<dependency>

//**可选配置项，生成Swagger UI**

<groupId>org.codehaus.enunciate</groupId>

<artifactId>enunciate-swagger</artifactId>

<version>${enunciate.version}</version>

</dependency>

<dependency>

<groupId>org.opendaylight.controller</groupId>

<artifactId>commons.logback\_settings</artifactId>

<version>0.0.5-SNAPSHOT</version>

</dependency>

</dependencies>

<executions>

<execution>

<goals>

<goal>docs</goal>

</goals>

</execution>

</executions>

</plugin>

…..

</plugins>

</pluginManagement>

</build>

**2）北向接口子模块须在pom.xml中使用maven-enunciate-plugin**

<plugin>

<groupId>org.codehaus.enunciate</groupId>

<artifactId>maven-enunciate-plugin</artifactId>

<dependencies>

<dependency>

<groupId>org.opendaylight.controller</groupId>

<artifactId>sal</artifactId>

<version>${sal.version}</version>

</dependency>

</dependencies>

</plugin>

**3）北向接口子模块在根目录创建enunciate.xml配置文件：**

<?xml version="1.0"?>

<enunciate label="full" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://enunciate.codehaus.org/schemas/enunciate-1.28.xsd">

**//配置REST服务的默认根路径，所有发布的URI前缀都为/controller/nb/v2/connectionmanager**

<services>

<rest defaultRestSubcontext="/controller/nb/v2/connectionmanager"/>

</services>

**//配置所用模块，docs模块的文档输出路径为rest目录，生成的HTML的标题为Connection Manager REST API，包含XML和JSON的例子**

<modules>

<docs docsDir="rest" title="Connection Manager REST API" includeExampleXml="true" includeExampleJson="true"/>

</modules>

</enunciate>

北向接口提供了查询表达式，以过滤返回对象，具体实现参考测试用例

<http://blog.dejavu.sk/2013/11/19/registering-resources-and-providers-in-jersey-2/>

北向接口调用方法

<https://wiki.opendaylight.org/view/OpenDaylight_Controller:Sample_Applications/SampleStatisticsApplication>

### shell设计

http://karaf.apache.org/manual/latest/developers-guide/extending.html

#### 创建shell项目

mvn archetype:create \

-DarchetypeArtifactId=maven-archetype-quickstart \

-DgroupId=org.apache.karaf.shell.samples \

-DartifactId=shell-sample-commands \

-Dversion=1.0-SNAPSHOT

#### pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>org.apache.karaf.shell.samples</groupId>

<artifactId>shell-sample-commands<artifactId>

<packaging>bundle</packaging>

<version>1.0-SNAPSHOT</version>

<name>shell-sample-commmands</name>

**引入shell依赖**

<dependencies>

<dependency>

<groupId>org.apache.karaf.shell</groupId>

<artifactId>org.apache.karaf.shell.console</artifactId>

<version>3.0.3-SNAPSHOT</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.felix</groupId>

<artifactId>maven-bundle-plugin</artifactId>

<version>2.4.0</version>

<configuration>

<instructions>

<Import-Package>

org.apache.felix.service.command,

org.apache.karaf.shell.commands,

org.apache.karaf.shell.console,

\*

</Import-Package>

</instructions>

</configuration>

</plugin>

</plugins>

</build>

</project>

### yang设计

## 核心模块

### 模块组成

**Service Abstract Layer (SAL)**

**Cluster Service**

**集群服务**

**Configuration Service**

**配置服务**

**Container Manager**

**容器服务**

**Connection** **Manager**

**连接服务**

**Switch Manager**

**交换机管理**

**ForwardingRules Manager**

**流表管理**

**Host Tracker**

**主机跟踪**

**Topology Manager**

**拓扑管理**

**Network Configuration**

**网络配置**

**User Manager**

**用户管理**

**Statistic Manager**

**状态管理**

**基础服务**

**Northbound**

**RestAPI**

**应用接口**

**Web网站**

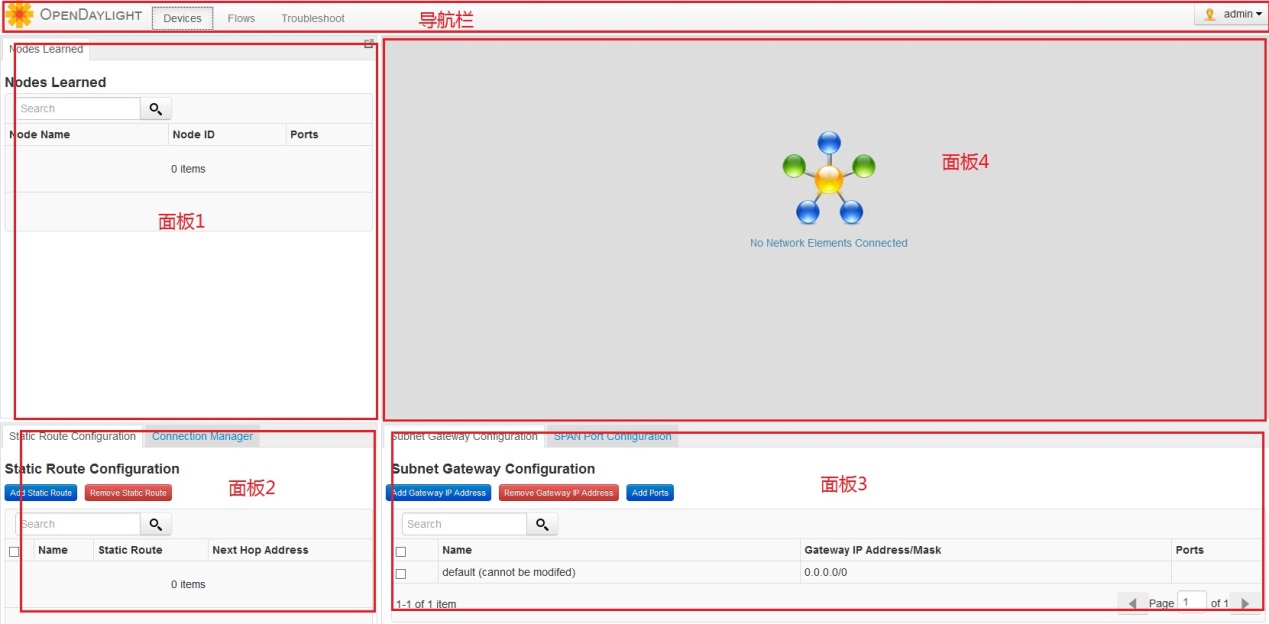
**shell**

**……**

* 底层为SAL（Service Abstract Layer，服务抽象层），作为其他模块的基础，将通用类和方法集中在该模块中，所有上层模块都依赖于该模块
* 中层为基础服务，ODL根据功能不同封装一个个服务（又称bundle，模块），对外提供接口调用
* 上层为应用接口，ODL利用基础服务提供的接口，实现了北向接口、Web网站及shell，供用户直接调用

### web网站模块

controller/opendaylight/web目录下是网站所有组成模块

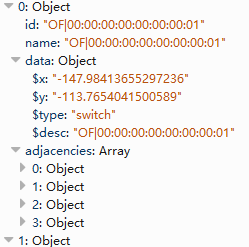


加载流程：main.jsp(主页)->open.js(初始化导航栏)->page.js(根据导航栏状态，请求devices或flows或troubleshoot的page.js，初始化面板1，2，3)->open-topology.js(初始化面板4)

**root**：提供网站主架构，包括用户认证，网站主页

**topology**：提供拓扑结构，核心是由open-topology发起/visual.json请求返回拓扑结构，绘制面板4

JSON结构如下：





**devices**：对于主页上的Devices页签所显示的内容

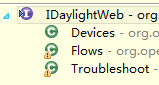
**flows**：对于主页上的Flows页签所显示的内容

**troubleshoot**：对于主页上的Troubleshoot页签所显示的内容

**brandfragment**：

**osgi-brandfragment**：右上角下拉菜单中的OSGI页面

IDaylightWeb：返回前端显示的bundle



### configuration系统配置模块

**该模块用于保存和恢复模块的配置文件**

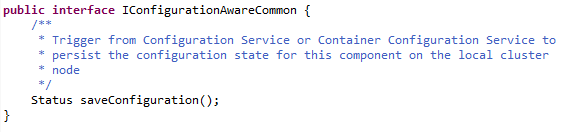
使用方法：

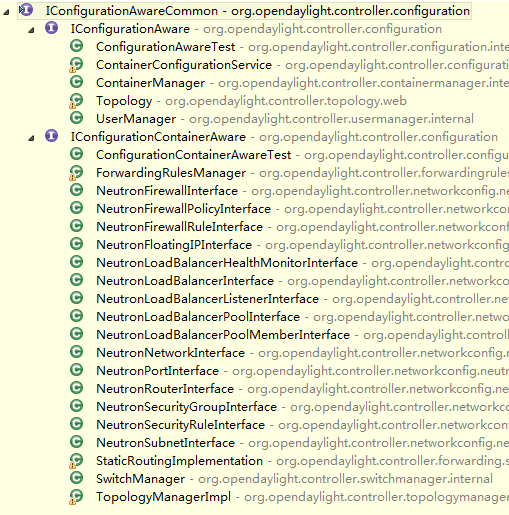
配置管理方式分为通用配置（global instance components，与具体容器无关）和容器配置（components in a container）。

如果是使用容器配置的模块，须实现IConfigurationContainerAware接口，导入IConfigurationContainerService对象；

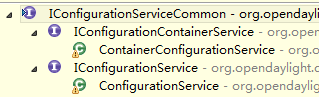
如果是使用通用配置的模块，须实现IConfigurationAware接口，导入IConfigurationService对象。

**IConfigurationAwareCommon：**接收配置事件的监听父接口，其唯一接口saveConfiguration用于监听保存配置的事件，被ConfigurationService或ContainerConfigurationService的saveConfiguration调用，在本地节点持久化配置状态。





**IConfigurationServiceCommon：**定义了保存、持久化、解析配置信息三个接口，分为IConfigurationService和IContainerConfigurationService两个子接口



**public** **interface** IConfigurationServiceCommon {

如果调用IConfigurationService，则保存所有配置，包括通用配置和容器配置，如果调用IContainerConfigurationSerivce，则保存当前容器的所有配置

Status saveConfigurations();

所有模块在saveConfiguration()中调用该方法，实现配置持久化

Status persistConfiguration(List<ConfigurationObject> config, String storeName);

所有模块在init()-loadConfiguration()中调用该方法，从文件中恢复原有配置

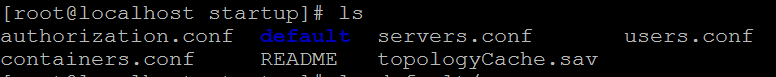
List<ConfigurationObject> retrieveConfiguration(IObjectReader reader, String storeName);

}

#### 配置文件

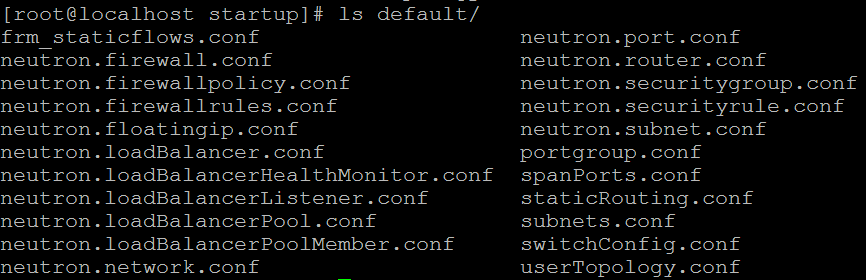
默认存放在

controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/configuration/startup下



可看到全局配置有authorization.conf,services.conf等

如果没有设置containerName，则默认为default，在default目录下保存与Container相关的配置



1. **全局配置**

users.conf：本地用户，UserManager

servers.conf：AAA Server，UserManager

authorization.conf：授权，UserManager

topologyCache.sav：拓扑，Topology

containers.conf：容器，ContainerManager

1. **容器配置**

frm\_staticflows.conf：静态流，ForwardingRulesManager

portgroup.conf：端口组，ForwardingRulesManager

neutron.firewall.conf：neutron防火墙，NeutronFirewallInterface

neutron.firewallpolicy.conf：neutron防火墙策略，NeutronFirewallPolicyInterface

neutron.firewallrules.conf：neutron防火墙规则，NeutronFirewallRuleInterface

neutron.floatingip.conf：neutron浮动IP，NeutronFloatingIPInterface

neutron.loadBalancerHealthMonitor.conf ：neutron负载平衡健康NeutronLoadBalancerHealthMonitorInterface

neutron.loadBalancer.conf：neutron负载平衡，NeutronLoadBalancerInterface

neutron.loadBalancerListener.conf：neutron负载监听，NeutronLoadBalancerListenerInterface

neutron.loadBalancerPool.conf：neutron负载池，NeutronLoadBalancerPoolInterface

neutron.loadBalancerPoolMember.conf：neutron负载池成员，NeutronLoadBalancerPoolMemberInterface

neutron.network.conf：neutron网络，NeutronNetworkInterface

neutron.port.conf：neutron端口，NeutronPortInterface

neutron.router.conf：neutron路由器，NeutronRouterInterface

neutron.securitygroup.conf：neutron安全组，NeutronSecurityGroupInterface

neutron.securityrule.conf：neutron安全规则，NeutronSecurityRuleInterface

neutron.subnet.conf：neutron子网，NeutronSubnetInterface

staticRouting.conf：静态路由，StaticRoutingImplementation

subnets.conf：子网，SwitchManager

spanPorts.conf：端口，SwitchManager

switchConfig.conf ：交换机，SwitchManager

userTopology.conf：用户拓扑，TopologyManagerImpl

### clustering.services集群服务模块

**opendaylight支持多个控制器组成的集群，控制器间需要共享缓存**

**该模块基于Infinispan提供集群缓存共享服务。**

**coordinator：协调者，即最老的opendaylight实例**

**ClusterManagerCommon利用ClusterManager实现具体操作缓存的方法，ClusterManager是对Infinispan进行封装，首先读取config/infinispan-config.xml，创建DefaultCacheManager对象，然后利用containerName\_cacheName作为缓存名称，创建缓存**



**GossipRouter**：用于绕开防火墙，以及减少节点之间的互相连接数，节点直接和router通信

**IClusterServicesCommon核心方法**：

根据缓存名和缓存模式创建缓存

ConcurrentMap<?, ?> **createCache**(String cacheName,

Set<IClusterServices.cacheMode> cMode) throws CacheExistException,

CacheConfigException;

根据缓存名返回缓存

ConcurrentMap<?, ?> **getCache**(String cacheName);

IClusterServices：

cacheMode分为四种，可两两组合

TRANSACTIONAL：事务，失败后回滚

NON\_TRANSACTIONAL：非事务，失败后不回滚，TRANSACTIONAL不可与NON\_TRANSACTIONAL共用

ASYNC：异步，请求发出即返回

SYNC：同步， 请求发出直到接收到返回值才返回，SYNC不可与ASYNC共用

cacheProps分为TRANSACTION\_PROP(事务属性),CLUSTERING\_PROP(集群属性),LOCKING\_PROP(锁属性)

流程是

### usermanager用户管理模块

**该模块用于管理登陆用户的状态，提供两种认证授权方式，1)本地存储用户信息localUserConfigList;2)远程授权服务器，AAA Server**

**usermanager**包括：

* HttpSessionListener->**ISessionManager**->SessionManager：会话管理接口
* UserDetailsService->**IUserManager**->UserManager：用户接口
* ConfigurationObject->**UserConfig**->**AuthorizationConfig** ：用户授权配置
* ConfigurationObject->**ServerConfig**：远程AAA Server配置

**usermanager.implementation**：用户管理实现模块

* HttpSessionListener->ISessionManager->**SessionManager**：会话管理实现类

ControllerUISessionManager配置在web.xml中，用于在会话创建，销毁时调用SessionManager完成sessionMap的增删操作，未使用该对象。

* UserDetailsService->IUserManager->**UserManager**：用户管理实现类

**UserManager的主要成员：**

ConcurrentMap<String, UserConfig> **localUserConfigList**; 局部用户配置列表，key=username，value=UserConfig （包含用户名、密码、角色），保存在users.conf中

ConcurrentMap<String, ServerConfig> **remoteServerConfigList**; 远程服务器配置列表，key=ip，value=ServerConfig（包含服务器IP，密钥，协议），保存在servers.conf中

ConcurrentMap<String, AuthorizationConfig> **authorizationConfList**; 授权用户配置列表，key=username，value= AuthorizationConfig（包含用户名，密码，角色），保存在authorization.conf中

ConcurrentMap<String, AuthenticatedUser> **activeUsers**; 有效用户，当前登录的用户，key=username，value= AuthenticatedUser（包含角色，授权日期）

ConcurrentMap<String, IAAAProvider> **authProviders**; 授权认证提供者，key=providername，value=IAAAProvider

IClusterGlobalServices **clusterGlobalService**; 集群服务

IConfigurationService **configurationService**; 配置服务

IContainerAuthorization containerAuthorizationClient;

### containermanager容器管理模块

**该模块用于创建、删除、配置Container**

**ContainerFlow**：与容器相关的流表

Match match：所有匹配字段

**ContainerImpl**：容器实现，实现IContainer接口

String containerName：容器名称

IContainerInternal iContainerInternal：内部容器，用于获取容器信息ContainerData

IContainerInternal由ContainerManager继承，根据containerName返回containerData

**ContainerData**：容器数据，包含容器相关的所有属性

String containerAdminRole：容器管理员角色

String containerOperatorRole：容器操作员角色

String name：容器名称

ConcurrentMap<Node, Set<NodeConnector>> swPorts：容器中包含的所有交换机和交换机上的端口

short staticVlan：静态Vlan

List<ContainerFlow> cFlowList：所有流表集合

**ContainerManager**：

IClusterGlobalServices clusterServices：集群服务

IConfigurationService configurationService：配置服务

ConcurrentMap<String, ContainerConfig> containerConfig：容器名称和容器配置的映射，String为containerName

ConcurrentMap<String, ContainerData> containerData：容器名称和容器数据的映射，String为containerName

ConcurrentMap<NodeConnector, CopyOnWriteArrayList<String>> nodeConnectorToContainers：一个Port对应的容器，可能一个Port由多个容器共享

ConcurrentMap<Node, Set<String>> nodeToContainers：一个Switch对应的容器

ConcurrentMap<String, Object> containerChangeEvents：容器改变事件，String为containerName，Object为NodeConnectorChangeEvent，ContainerFlowChangeEvent，ContainerChangeEvent

Set<IContainerAware> iContainerAware：容器创建和销毁时的接口事件

Set<IContainerListener> iContainerListener：容器监听器，

Set<IContainerLocalListener> iContainerLocalListener：本地容器监听器

### switchmanager交换机管理模块

**该模块用于管理所有实体组件的信息，包括Controller，Switch，Node，NodeConnector，Subnet，SpanPort**

**Switch**

**交换机**

**Node**

**抽象核心**

**NodeConnector**

**端口Port**

**SwitchConfig**

**交换机配置**

**Subnet**

**子网，例如10.0.0.254/24**

**SubnetConfig**

**子网配置**

**SpanConfig**

**？？？**

**Node**：switch的抽象对象

Object nodeID：switch ID，根据nodeType不同，设置不同类型的ID值，如Integer，Short

String nodeType：switch类型，分为OPENFLOW,PCEP,ONEPK,PRODUCTION，其他自定义的在INodeFactory

String nodeIDString：根据nodeID和nodeType转换为字符串，格式为Node Type|Node ID，例如OF|00:00:00:00:00:00:00:01

**NodeConnector**：端口Port

Object nodeConnectorID：port ID，根据nodeConnectorType不同，设置不同类型的ID值，如Integer，Short

String nodeConnectorType：port类型，分为

CONTROLLER(PACKET\_IN),ALL(OFPP\_ALL),SWSTACK(OFPP\_LOCAL),…..，其他自定义的在INodeConnectorFactory中

Node nodeConnectorNode：所附属的节点

String nodeConnectorIDString：根据nodeConnectorID和nodeConnectorType转换为字符串，格式为Port Type|Port ID@Node Type|Node ID，例如OF|1@OF|00:00:00:00:00:00:00:01

**Switch**：交换机

byte[] dataLayerAddress：MAC地址的byte[]形式

Set<NodeConnector> nodeConnectors：该交换机上的端口列表

List<NodeConnector> spanPorts：

Node node：交换机核心对象

**SwitchConfig**：交换机配置，通过其可配置Switch

String nodeId：switch ID

Map<String, Property> nodeProperties：该switch的配置，包括description，tier，forwarding属性

**Subnet**：子网，例如10.0.0.254/24

InetAddress networkAddress：网络地址，如10.0.0.254

InetAddress subnetPrefix：子网前缀，如10.0.0.0

short subnetMaskLength：子网掩码长度，如24

short vlan：vlan号

Set<NodeConnector> nodeConnectors：子网内包含的端口，如果为null，则是FlatLayer2

**SubnetConfig**：子网配置，通过其可获取Subnet

String name：子网名称，取个代号，如subnet

String subnet：子网，格式为A.B.C.D/MM，A.B.C.D是默认网关IP（L3），ARP查询IP（L2）

List<String> nodeConnectors：端口名称列表

**SpanConfig**：什么意思？？？

String nodeId：交换机ID

String spanPort：端口号

**ISwitchManager**：涉及Subnet，Node，NodeConnector，Span Port的配置



List<NodeConnector> spanNodeConnector：span node端口列表

ConcurrentMap<SpanConfig, SpanConfig> spanConfigList：spanConfig 映射表

ConcurrentMap<InetAddress,Subnet> subnets：网关IP与子网映射表，InetAddress为网关IP，例如10.0.0.254

ConcurrentMap<String, SubnetConfig> subnetsConfigList：subnetConfig列表，String为子网名称

ConcurrentMap<String, SwitchConfig> nodeConfigList：交换机ID与SwitchConfig映射表，String为node ID

ConcurrentMap<Node,Map<String, Property>> nodeProps：交换机和属性映射表

ConcurrentMap<Node,Map<String, NodeConnector>> nodeConnectorNames：交换机和端口映射表

ConcurrentMap<NodeConnector,Map<String,Property>> nodeConnectorProps：端口与属性映射表，String是属性名称

ConcurrentMap<String, Property> controllerProps：控制器属性，String是属性名称

IInventoryService inventoryService：

IStatisticsManager statisticsManager：

IControllerProperties controllerProperties：Controller属性，就是Coordinator的MAC地址

IConfigurationContainerService configurationService：配置文件服务

**ISpanAware**：spanUpdate接收Span Port添加或删除的事件

**ISwitchManagerAware**：subnetNotify接受添加、删除子网事件；modeChangeNotify接受Node mode事件

IInventoryListener：notifyNode接收Node属性修改事件，notifyNodeConnector接收NodeConnector属性修改事件

### forwardingrulesmanager流表管理模块

**该模块用于管理转发规则，包括静态流表**

**Action**

**匹配动作**

**ActionType**

**动作类型**

**Flow**

**流表项**

**Match**

**所有匹配字段**

**1:n**

**MatchType**

**匹配类型**

**MatchField**

**匹配字段**

**FlowEntry**

**流表实体**

**Node**

**交换机**

**ActionType**：动作类型，枚举型

String id：ID

int minValue：最小值

int maxValue：最大值

**Action**：抽象动作父类，其子类定义了具体动作所需的各项参数

ActionType type：动作类型

boolean isValid：该动作是否有效

**MatchType**：匹配类型，枚举型

String id：ID

int index：匹配索引，1的偏移量递增，用于在Match中设置matches，通过一个int表示共有几个匹配字段

Class<?> dataType：匹配字段类型

long minValue：字段允许的最小值

long maxValue：字段允许的最大值

**MatchField**：匹配字段

MatchType type：匹配的是哪个字段

Object value：该字段的值是多少

Object mask：该字段的掩码多少

boolean isValid：该匹配字段是否有效

**Match**：某一流表项中所有的匹配字段

Map<MatchType, MatchField> fields：

int matches：以1的个数代表一共有多少个匹配项，以及哪些匹配项

**Flow**：一条流表项，由多个匹配字段和动作组成

Match match：所有匹配字段

List<Action> actions：动作集合

short priority：优先级

short idleTimeout：多长时间没匹配到则删除该流表

short hardTimeout：超过该时间则删除该流表

long id：ID

**FlowEntry：**一个流表实体，将一个交换机与一条流表关联起来

String groupName：所属组，可通过组删除

String flowName：流表名称

Node node：该流表所在的交换机

Flow flow：流表内容，包括匹配字段和动作

ConcurrentMap<Integer, FlowConfig> staticFlows;

ConcurrentMap<Integer, Integer> staticFlowsOrdinal;

ConcurrentMap<String, PortGroupConfig> portGroupConfigs;

ConcurrentMap<PortGroupConfig, Map<Node, PortGroup>> portGroupData;

ConcurrentMap<String, Object> TSPolicies;

IContainerManager containerManager;

IConfigurationContainerService configurationService;

boolean inContainerMode; // being used by global instance only

boolean stopping;

### connectionmanager连接管理模块

**该模块管理交换机到控制器的连接**

### topologymanager拓扑管理模块

### hosttracker主机跟踪

## 基础知识

### Java remote debug

在debugging视图中**远程调试应用**，需要在remote JVM启动时设置额外参数，在Eclipse中设置Remote Java Application。

**启动JVM命令：**

$ java -Xdebug -Xrunjdwp:transport=dt\_socket, server=y,address=8000,suspend=n -jar start.jar

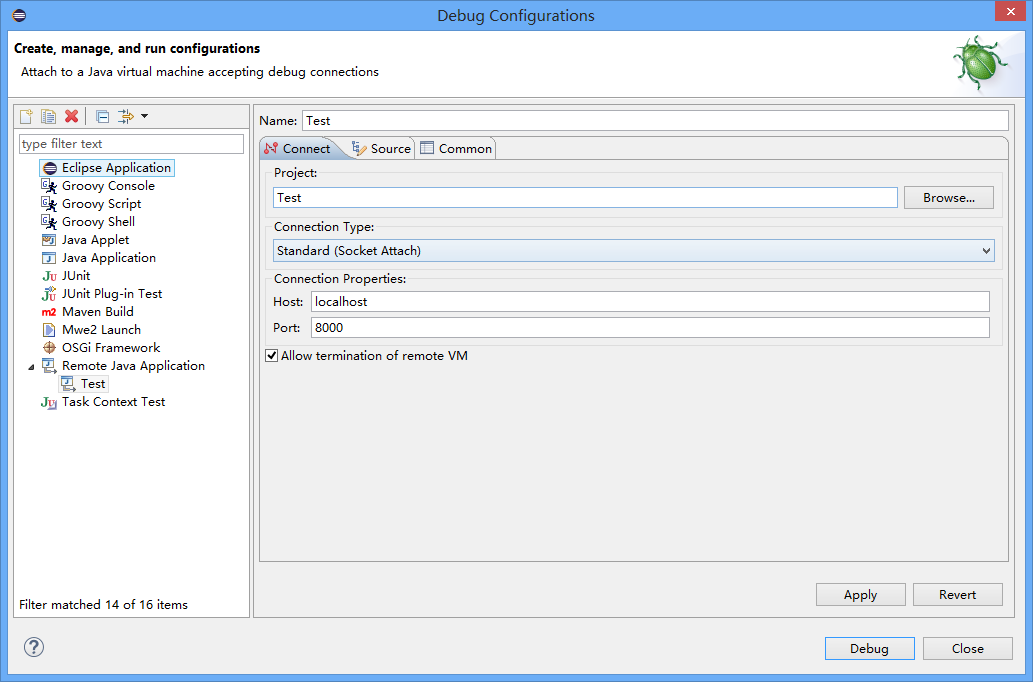
-Xdebug：启动调试

-Xrunjdwp:<sub-options> ：在VM中加载JDWP实现。

* transport：必填，与Eclipse连接的传输名称；
* server：默认为n，y为等待Eclipse连接，n为主动连接到Eclipse；
* address：当server=n时，必填，连接到的Eclipse IP:Port，当server=y时，等待Eclipse连接到该Port。
* suspend：默认为y，y为VM暂停，直到Eclipse连接。

Eclipse中通过Connection Type的Socket Listen和Socket Attach设置server=n和y。

http://www.ibm.com/developerworks/cn/opensource/os-eclipse-javadebug/



### CORS Filter

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Access\_control\_CORS

<http://tomcat.apache.org/tomcat-7.0-doc/config/filter.html#CORS_Filter>

Tomcat7.0之后开始支持CORS（**跨域资源共享**）vs同源策略

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

在服务器的web.xml中添加如下过滤器：

<filter>

<filter-name>CorsFilter</filter-name>

<filter-class>org.apache.catalina.filters.CorsFilter</filter-class>

</filter>

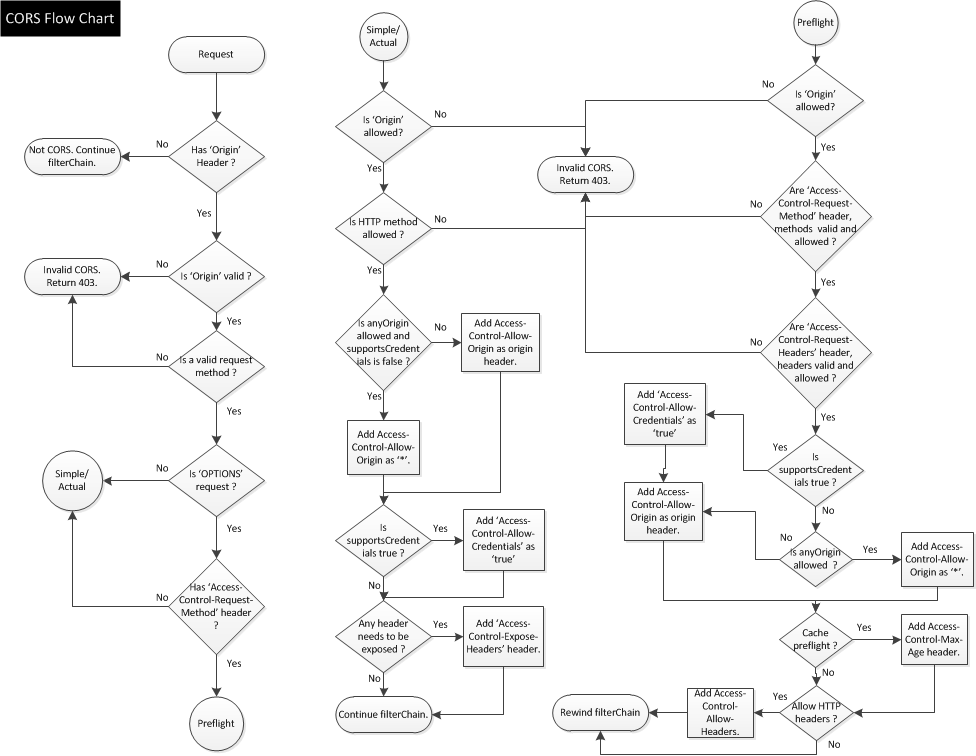
<filter-mapping>

<filter-name>CorsFilter</filter-name>

<url-pattern>/\*</url-pattern>

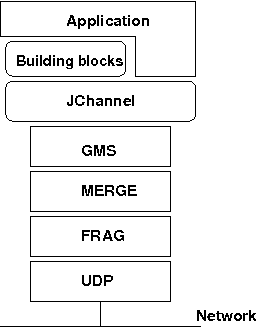
</filter-mapping>

filter会在HttpServletResponse的header中添加Access-Control-\*，告诉浏览器是否允许跨域访问及其详细设置。



### JGroups

**集群消息通信工具**



The architecture of JGroups

JGroups由Channel，building blocks，protocol stack三层组成。

应用发送消息后，由Channel将消息传送给最高层的protocol stack，再依次加工向下传递，直到最底层的protocol，接收过程相反，依次向上解析传递至Channel，并调用receive()。

<http://www.jgroups.org>

<http://www.oschina.net/translate/jgroups-writing-a-simple-application>

### Infinispan

**集群共享缓存** (支持集群和单节点)。vs ehcache，memcache

<http://infinispan.org/tutorials/embedded/index.html>

使用JGroups实现消息通信，使用Map/Reduce处理缓存数据

maven依赖：

<dependency>

<groupId>org.infinispan</groupId>

<artifactId>infinispan-embedded</artifactId>

<version>7.1.0.Final</version>

</dependency>

CacheManager：缓存容器，管理通用配置和资源，管理集群

CacheManager cacheManager = new DefaultCacheManager();

判断缓存是否存在

cacheManager.cacheExists(cacheName);

获得cache

Cache<String, Person> cache = cacheManager.getCache();

Person person = cache.get(personName);

cache.put(personName, person);

设置超时

cache.put(personName, person, 5, TimeUnit.SECONDS);

停止cacheManager

cacheManager.stop();

配置DefaultCacheManager

ConfigurationBuilder config = new ConfigurationBuilder();

config.expiration().lifespan(5, TimeUnit.SECONDS);

cacheManager = new DefaultCacheManager(config.build());

配置集群

GlobalConfigurationBuilder global = GlobalConfigurationBuilder.defaultClusteredBuilder();

global.transport().clusterName("WeatherApp");

cacheManager = new DefaultCacheManager(global.build(), config.build());

监听集群成员变化事件

@Listener

public class ClusterListener {

@ViewChanged

public void viewChanged(ViewChangedEvent event) {

}

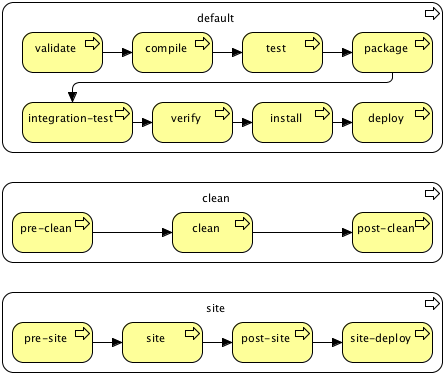
}

### maven

#### maven lifecycle

Maven的三套生命周期：

* Clean Lifecycle 在进行真正的构建之前进行一些清理工作。
* Default Lifecycle 构建的核心部分，编译，测试，打包，部署等等。
* Site Lifecycle 生成项目报告，站点，发布站点。



mvn clean：

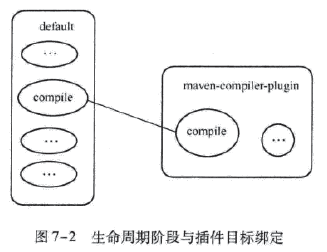
* pre-clean  执行一些需要在clean之前完成的工作
* clean  移除所有上一次构建生成的文件
* post-clean  执行一些需要在clean之后立刻完成的工作

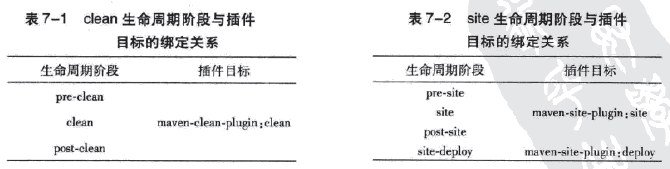
mvn site：

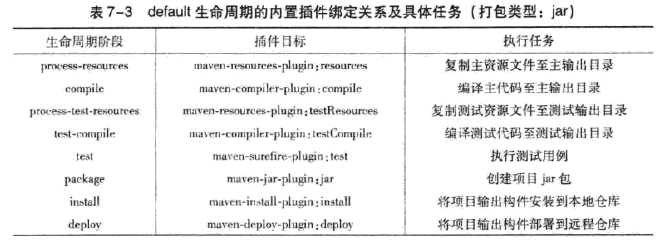
* pre-site     执行一些需要在生成站点文档之前完成的工作
* site    生成项目的站点文档
* post-site     执行一些需要在生成站点文档之后完成的工作，并且为部署做准备
* site-deploy     将生成的站点文档部署到特定的服务器上

mvn deploy：

* validate
* generate-sources
* process-sources 复制并处理Java文件，至目标目录，准备打包
* generate-resources
* process-resources     复制并处理资源文件，至目标目录，准备打包。
* compile     编译项目的源代码。
* process-classes
* generate-test-sources
* process-test-sources
* generate-test-resources
* process-test-resources     复制并处理资源文件，至目标测试目录。
* test-compile     编译测试源代码。
* process-test-classes
* test     使用合适的单元测试框架运行测试。这些测试代码不会被打包或部署。
* prepare-package
* package     接受编译好的代码，打包成可发布的格式，如 JAR 。
* pre-integration-test
* integration-test
* post-integration-test
* verify
* install     将包安装至本地仓库，以让其它项目依赖。
* deploy     将最终的包复制到远程的仓库，以让其它开发人员与项目共享。





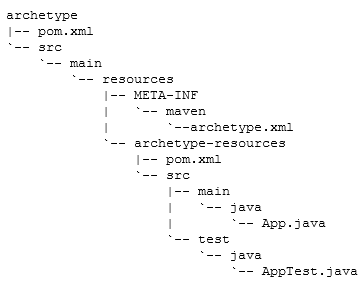


#### maven archetype

**创建模版项目**

target\generated-sources\archetype目录下包含项目原型，由以下几部分组成：

* archetype descriptor(即src/main/resources/META-INF/maven/archetype.xml)，其包含了可被archetype generation机制正确处理的所有文件和分类信息；
* src/main/resources/archetype-resources/下将被拷贝到新项目中的所有工程文件；
* src/main/resources/archetype-resources/pom.xml项目pom.xml
* 根目录下的pom.xml



定义maven工程模板：

* 创建maven项目，其包含了archetype:generate所需的groupId, artifactId,version
* 在project下执行mvn archetype:create-from-project，生成target/generated-sources/archetype目录，里面包含了创建项目所需的所有信息
* 执行mvn install将其安装到本地仓库中，或者mvn deploy发布到远端仓库中
* 执行mvn archetype:generate -DarchetypeGroupId=<archetype-groupId> -DarchetypeArtifactId=<archetype-artifactId> -DarchetypeVersion=<archetype-version> -DgroupId=<my-groupId> -DartifactId=<my-artifactId> 利用其创建新maven项目

或者mvn archetype:generate -DarchetypeCatalog=local

#### maven-checkstyle-plugin

**检查编码规范**

mvn checkstyle:check

目标：

checkstyle:check：执行checkstyle分析和输出错误到console，停止编译

checkstyle:checkstyle：输出任务，执行checkstyle分析，失败不停止编译，生成HTML报告

checkstyle:checkstyle-aggreagte：输出任务，执行checkstyle分析，失败不停止编译，生成HTML报告

checkstyle:help：输出maven-checkstyle-plugin帮助信息

maven-checkstyle-plugin内置了4种规范

* config/sun\_checks.xml
* config/maven\_checks.xml
* config/turbine\_checks.xml
* config/avalon\_checks.xml

运行checkstyle检查

mvn checkstyle:checkstyle

输出结果：target/checkstyle-result.xml

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-checkstyle-plugin</artifactId>

<version>2.10</version>

<configuration>

<configLocation>${basedir}/checkstyle.xml</configLocation>自定义检查文件

</configuration>

<executions>

<execution>

<id>checkstyle</id>

<phase>validate</phase>

<goals>

<goal>check</goal>

</goals>

<configuration>

<failOnViolation>true</failOnViolation>

</configuration>

</execution>

</executions>

</plugin>

#### maven-bundle-plugin

**配置并输出Bundle**，输出MANIFEST.MF文件

<http://felix.apache.org/documentation/subprojects/apache-felix-maven-bundle-plugin-bnd.html>

#### maven-enunciate-plugin

**加强Web Service功能**

当利用标准Java技术开发WebService时，在编译过程中加入enunciate，则enunciate引擎会根据配置文件和注释信息，创建完整的HTML文档，客户端库（包括Java, .NET, iOS, Ruby, Flex, AJAX, GWT, etc）供第三方开发者使用，接口定义文档（包括WSDL，WADL，XML-Schema，etc）

http://enunciate.codehaus.org

调用enunciate的四种方式：

* Maven：在pom.xml中配置maven-enunicate-plugin
* Ant
* Command-Line Scripts：下载zip包解压，配置环境变量，执行enunciate命令
* Programmatically in Java：详见API

**maven plugin：**

根据应用场景不同，依赖的模块也不同，maven-enunciate-plugin包含所有基础base模块。

| **groupId** | **artifactId** | **modules included** |
| --- | --- | --- |
| org.codehaus.enunciate | maven-enunciate-plugin | all "base" modules |
| org.codehaus.enunciate | maven-enunciate-spring-plugin | all "base" modules + spring-app |
| org.codehaus.enunciate | maven-enunciate-cxf-plugin | all "base" modules except jaxws-ri and jersey + cxf |
| org.codehaus.enunciate | maven-enunciate-jboss-plugin | all "base" modules except jaxws-ri and jersey + jboss |
| org.codehaus.enunciate | maven-enunciate-slim-plugin | basic-app only |



**Base Modules：**

enunciate被划分为模块，不同模块实现不同功能

| module | description |
| --- | --- |
| [c](https://github.com/stoicflame/enunciate/wiki/Module-C) | Generates the C/C++ client libraries. |
| [csharp](https://github.com/stoicflame/enunciate/wiki/Module-C-Sharp) | Generates the C# client libraries. |
| [docs](https://github.com/stoicflame/enunciate/wiki/Module-Docs) | Generates the API documentation. |
| [java-client](https://github.com/stoicflame/enunciate/wiki/Module-Java-Client) | Generates the Java client libraries. |
| [jaxws-ri](https://github.com/stoicflame/enunciate/wiki/Module-JAX-WS-RI) | Generates support classes for deploying SOAP endpoints using [JAX-WS RI](https://jax-ws.java.net/). |
| [jaxws-support](https://github.com/stoicflame/enunciate/wiki/Module-JAX-WS-Support) | Generates the support files and classes necessary to support a JAX-WS application according to the [JAX-WS Specification](https://jax-ws.java.net/). |
| [jersey](https://github.com/stoicflame/enunciate/wiki/Module-Jersey) | Generates the support for deploying your endpoints according to the JAX-RS specification using [Jersey](https://jersey.java.net/). |
| [obj-c](https://github.com/stoicflame/enunciate/wiki/Module-Obj-C) | Generates the Objective-C client libraries. |
| [php](https://github.com/stoicflame/enunciate/wiki/Module-PHP) | Generates the PHP client libraries. |
| [xml](https://github.com/stoicflame/enunciate/wiki/Module-XML) | Generates the XML Schemas and IDL files for the API. |

**enunciate.xml 配置文件：**

**//以<enunciate >为根结点，label是enunciate project的标签，在client-side libraries命名时会用到**

<enunciate label="full" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://enunciate.codehaus.org/schemas/enunciate-1.28.xsd">

**//禁用enunciate规则，不是所有规则都可以禁用**

<disable-rule id="..."/>

**//在生成文档或客户端代码时引用或排除的Facets（用@Facets注释的对象）**

<facets>

<include name="Public"/>

<exclude name="Admin"/>

</facets>

**//在compile-time时指明部署的hostname和context**

<deployment protocol="https" host="mydomain.com" context="/api"/>

**//配置SOAP或REST的默认根路径**

<services>

<soap defaultSoapSubcontext="/ws"/>

<rest defaultRestSubcontext="/api"/>

</services>

</enunciate>

### WebService

<http://www.codedata.com.tw/java/java-restful-1-jersey-and-jax-rs>

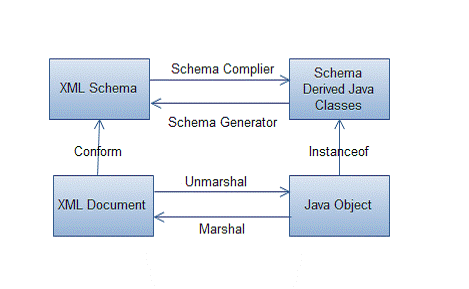
http://www.mkyong.com/tutorials/jax-rs-tutorials/

#### JAX-RS & JAX-WS & JAXB

JAX-RS：Java API for RESTful Web Services，RESTful

JAX-WS：Java API for XML Web Services，SOAP

JAXB：Java Architecture for XML Binding， JAX-WS中XML与Java类的数据绑定由JAXB实现。



**JAX-RS核心注释**：

|  |  |
| --- | --- |
| @PATH | 定义Root Resource Class或Resource Method的URI路径 |
| @GET, @POST, @PUT, @DELETE | 定义Resource Method的Http Method |
| @Consumes | 请求Request的Mime Type。根据Http Request Header的Content-Type判断是否符合该REST API的要求。 |
| @Produces | 返回Response的Mime Type，根据Http Request Header的Accept判断该REST API是否符合请求要求。 |
| @PathParam | 路径参数 |
| @QueryParam | 请求参数 |
| @FormParam | 表单参数 |
| @HeaderParam | 请求头参数 |
| @Context | 将Container的Context注射到POJO中 |

**JAXB核心注释：**

[**http://www.ibm.com/developerworks/cn/webservices/1003\_sunzg\_jaxb/**](http://www.ibm.com/developerworks/cn/webservices/1003_sunzg_jaxb/)

<http://www.ibm.com/developerworks/cn/java/j-lo-jaxrs/>

Web 资源可以有不同的表现形式，服务端与客户端之间需要一种称为**内容协商机制（Content Negotiation）**：作为服务端，Resource 方法的 Produces 标注用于指定响应体的数据格式（MIME 类型），Consumes 标注用于指定请求体的数据格式；作为客户端，Accept 请求头用于选择响应体的数据格式，Content-Type 请求头用于标识请求体的数据格式。

JAX-RS 依赖于 MessageBodyReader 和 MessageBodyWriter 的实现来自动完成返回值到响应体的序列化以及请求体到实体参数的反序列化工作，其中，XML 格式的请求／响应数据与 Java 对象的自动绑定依赖于 JAXB 的实现。

#### Jersey

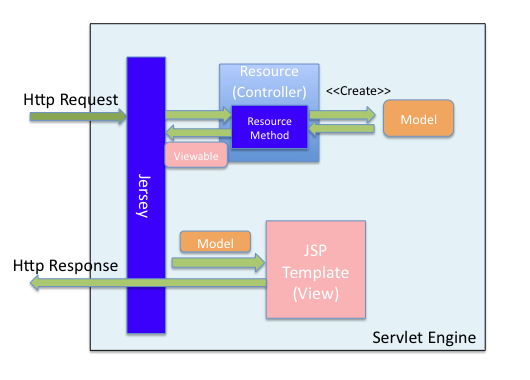
**发布RESTFUL服务**

[http://waylau.gitbooks.io/jersey-2-user-guide](http://waylau.gitbooks.io/jersey-2-user-guide/content/Chapter%201.%20Getting%20Started%20%E5%BC%80%E5%A7%8B/1.1.%20Creating%20a%20New%20Project%20from%20Maven%20Archetype%20%E4%BB%8EMaven%20Archetype%E5%88%9B%E5%BB%BA%E4%B8%80%E4%B8%AA%E6%96%B0%E9%A1%B9%E7%9B%AE.html)

JAX-RS的参考实现有Sun 的Jersey、Apache 的 CXF 以及 JBoss 的 RESTEasy

Jersey分为sun和glassfish版本，2.0以前是sun，2.0以后是glassfish

Jersey相对Spring MVC为轻量级，并且可以利用JAX-RS的API。



<http://www.mkyong.com/webservices/jax-rs/restful-java-client-with-jersey-client/>

### Netty

异步事件驱动网络应用框架，用于快速开放高效服务器与客户端协议。

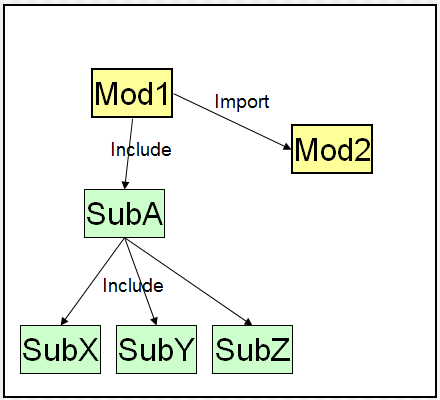
### Yang

数据模型语言，定义分层树形结构用于网络基本操作，如配置、状态、远程调用，提醒等。，XML格式

<http://www.cs.vu.nl/~kielmann/asfr11/slides/yang.pdf>

#### 语法

##### module&submodule



**module名称必须唯一**

module acme-module {

**Header statements：namespace定义XML URI，namespace和module name共同组成唯一索引**

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

prefix acme;

**Linkage statement**

import "yang-types" {

prefix yang;

}

include "acme-system";

**Meta information**

organization "ACME Inc.";

contact joe@acme.example.com;

description "The module for entities

implementing the ACME products";

**Revision history**

revision 2007-06-09 {

description "Initial revision.";

}

…

}

##### leaf

只包含一个值，没有子节点，单个实例

YANG Example:

leaf host-name {

type string;

mandatory true;

config true;

description "Hostname for this system";

}

NETCONF XML Encoding:

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

##### leaf-list

只包含一个值，没有子节点，多个实例

YANG Example:

leaf-list domain-search {

type string;

ordered-by user;

description "List of domain names to search";

}

NETCONF XML Encoding:

<domain-search>high.example.com</domain-search>

<domain-search>low.example.com</domain-search>

<domain-search>everywhere.example.com</domain-search>

##### container

本身没有值，包含多个子节点，单个实例

YANG Example:

container system {

container services {

container ssh {

presence "Enables SSH";

description "SSH service specific configuration";

// more leafs, containers and stuff here...

}

}

}

NETCONF XML Encoding:

<system>

<services>

<ssh/>

</services>

</system>

##### list

通过指定key，唯一限定实体，包含多个子节点，本身不包含值，多个实例

YANG Example:

list user {

key name;

leaf name {

type string;

}

leaf uid {

type uint32;

}

leaf full-name {

type string;

}

leaf class {

type string;

default viewer;

}

}

NETCONF XML Encoding:

<user>

<name>glocks</name>

<full-name>Goldie</full-name>

<class>intruder</class>

</user>

<user>

<name>snowey</name>

<full-name>Snow</full-name>

<class>free-loader</class>

</user>

<user>

<name>rzull</name>

<full-name>Repun</full-name>

</user>

##### prefix

在引入外部模块时可通过prefix设置别名，这样可简化在模块内部的引用长度；

如将ietf-yang-types重命名为yang

import ietf-yang-types { prefix yang; }

在其他地方可直接通过yang应用其中的node

type yang:date-and-time

##### must

通过条件等式限制大小关系，比如retry-timer必须小于access-timeout，否则打印error-message。

container timeout {

leaf access-timeout {

description "Maximum time without server response";

units seconds;

mandatory true;

type uint32;

}

leaf retry-timer {

description "Period to retry operation";

units seconds;

type uint32;

must "$this < ../access-timeout" {

error-app-tag retry-timer-invalid;

error-message "The retry timer must be "

+ "less than the access timeout";

}

}

}

##### augment

通过引入模型system/login/user，在原有模型基础上增加node expire，原有模型不变。

YANG Example:

augment system/login/user {

leaf expire {

type yang:date-and-time;

}

}

NETCONF XML Encoding:

<user>

<name>alicew</name>

<class>drop-out</class>

<other:expire>2112-04-01T12:00:00</other:expire>

</user>、

##### mandatory

如果父节点存在，则必须设置当前节点

##### presence

如果dynamic-bootp节点存在，则当前range节点设置生效

container range {

presence "enables dynamic address assignment";

leaf dynamic-bootp {

type empty;

description

"Allows BOOTP clients to get addresses in this range";

}

}

<range>

<dynamic-bootp/>

</range>

##### when

当when的条件满足时，增加node，when是XPath表达式

YANG Example:

augment system/login/user {

when "class = wheel";

leaf shell {

type string;

}

}

NETCONF XML Encoding:

<user>

<name>alicew</name>

<class>wheel</class>

<other:shell>/bin/tcsh</other:shell>

</user>

##### Built-in types

内置类型，可直接使用

|  |  |
| --- | --- |
| Category | Types |
| Integral | {,u}int{8,16,32,64} |
| String | string, enumeration, boolean |
| Binary Data | binary |
| Bit fields | bits |
| References | instance-identifier, keyref |
| Other | empty |

##### Derived types

延伸类型，限制属性有range，length，pattern（正则式）

YANG Example:

typedef percent {

type uint16 {

range "0 .. 100";

}

description "Percentage";

}

leaf completed {

type percent;

}

NETCONF XML Encoding:

<completed>20</completed>

##### union

允许leaf包含一系列可选值

YANG Example:

leaf limit {

description "Number to allow";

type union {

type uint16 {

range "0 .. 100";

}

type enumeration {

enum none {

description "No limit";

}

}

}

}

NETCONF XML Encoding:

<limit>20</limit>

或<limit>none</limit>

##### grouping

定义可重用的node集合，通过grouping可多次引用node，重定义

YANG Example

grouping target {

leaf address {

type inet:ip-address;

description "Target IP address";

}

leaf port {

type inet:ip-port;

description "Target port number";

}

}

container peer {

container destination {

uses target;

}

}

NETCONF XML Encoding:

<peer>

<destination>

<address>192.0.2.1</address>

<port>22</port>

</destination>

</peer>

##### choice

只允许choice中的一个成员出现在datastore中

YANG Example:

choice transfer-method {

leaf transfer-interval {

description "Frequency at which file transfer happens";

type uint {

range "15 .. 2880";

}

units minutes;

}

leaf transfer-on-commit {

description "Transfer after each commit";

type empty;

}

}

NETCONF XML Encoding:

<transfer-on-commit/>

##### anyxml

允许node中包含任意XML内容

YANG Example:

anyxml software-version {

description "Number to allow";

}

NETCONF XML Encoding:

<software-version>

<base>A10.2</base>

<routing>B4.2</routing>

<snmp>C87.12</snmp>

</software-version>

##### rpc

包含method名称，input、output参数

rpc activate-software-image {

input {

leaf image-name {

type string;

}

}

output {

leaf status {

type string;

}

}

}

<rpc xmlns="urn:mumble">

<activate-software-image>

<image-name>image.tgz</image-name>

</activate-software-image>

</rpc>

##### notification

包含name，content

YANG Example:

notification link-failure {

description "A link failure has been detected";

leaf if-index {

type int32 { range "1 .. max"; }

}

leaf if-name {

type keyref {

path "/interfaces/interface/name";

}

}

}

## 问题

问题1：page.js如何调试？

**问题2：enunciate：Plugin execution not covered by lifecycle configuration?**

<http://eclipse.org/m2e/documentation/m2e-execution-not-covered.html>

**问题3：eclipse jetty support jsp？**

eclipse内置的jetty plugin不支持JSP解析，在pom.xml增加dependency

<dependency>

<groupId>org.mortbay.jetty</groupId>

<artifactId>jsp-2.1-glassfish</artifactId>

<version>${jsp.glassfish.version}</version>

</dependency>

Action中，其他操作要在OUTPUT之前，如SET\_NW\_DST等。

主机接收包后必须匹配目的IP和MAC地址都对，才处理包

问题：Error initializing: org.codehaus.plexus.velocity.DefaultVelocityComponent@bdb5543

java.lang.NoClassDefFoundError: org/apache/commons/lang/StringUtils

**问题：com sun mirror apt annotationprocessor missing**

enunciate不支持jdk 1.8导致

**问题：在window下编译，enunciate bug，报错：illegal unquoted character**

controller\opendaylight\commons\opendaylight\pom.xml

<enunciate.version>1.28</enunciate.version>

1.28改为1.27

# Mininet

Mininet是由一些虚拟的终端节点（end-hosts）、交换机、路由器连接而成的**网络仿真器**，采用轻量级的虚拟化技术使得系统可以和真实网络相媲美。

SDN研发测试平台，支持OpenFlow、Open vSwitch，支持复杂拓扑和自定义拓扑，提供Python API。

其他OpenFlow网络仿真器，vEmulab，EstiNet，[ns-3](http://www.nsnam.org/)

## 安装

只支持ubuntu

1. **系统安装**

可以直接下载安装了mininet的ubuntu镜像

1. **源码安装**

sudo apt-get install git

git clone git://github.com/mininet/mininet

mininet/util/install.sh -a

sudo mn --test pingall

1. **package安装**

如果之前安装过mininet或OpenvSwitch，则先删除原来的文件

sudo rm -rf /usr/local/bin/mn /usr/local/bin/mnexec \

/usr/local/lib/python\*/\*/\*mininet\* \

/usr/local/bin/ovs-\* /usr/local/sbin/ovs-\*

lsb\_release –a //查询版本

根据ubuntu版本号，执行以下一条命令

Mininet 2.1.0 on Ubuntu 13.10: sudo apt-get install mininet

Mininet 2.0.0 on Ubuntu 13.04: sudo apt-get install mininet

Mininet 2.0.0 on Ubuntu 12.10: sudo apt-get install mininet/quantal-backports

Mininet 2.0.0 on Ubuntu 12.04: sudo apt-get install mininet/precise-backports

关闭openvswitch-controller

sudo service openvswitch-controller stop

sudo update-rc.d openvswitch-controller disable

sudo mn --test pingall

**mininet与opendaylight连接**

将opendaylight启动后，执行ip为opendaylight的IP，创建3层树拓扑。

sudo mn --controller=remote,ip=192.168.214.145 --topo tree,2

mininet> pingall

刷新opendaylight网页就可看到mininet网络。

在桌面中执行 sudo wireshark，打开抓包软件

## 使用

### mn命令

sudo mn --switch ovsk --controller ref --topo tree,depth=2,fanout=8 --test pingall //创建一个树形网络，深度2，执行pingall测试

**重要参数：**

**-c ：**sudo mn -c //如果mininet崩溃，则清除配置

**--test** ：sudo mn --test cmd //启动后执行cmd操作，完成后退出

如sudo mn --test pingpair（连通测试），sudo mn --test iperf（性能测试）

**--controller**：配置控制器

* default：mininet自带controller
* none
* nox
* ovsc
* ref
* remote：sudo mn --controller=remote,ip=[controller IP],port=[controller listen port]，ip默认为'127.0.0.1'，port默认为6633
* ryu：日本的一款controller

**--topo**：配置拓扑

* linear：线性拓扑，sudo mn --topo=linear,4 //4个switch连成一条线，4个host，每个switch连1个host
* minimal：默认值，最小拓扑，1个switch，2个host，sudo mn
* reversed
* single：单个交换机拓扑，sudo mn --topo=single,3 //1个switch，3个host
* torus：环形拓扑
* tree：树形拓扑，sudo mn --topo=tree,2 //2层拓扑

**-x**：启动每个host和switch的xterm，在桌面执行sudo mn -x

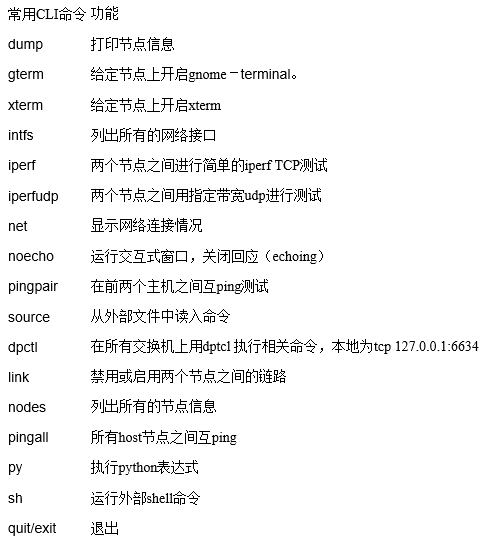
**-v：**日志等级，info，debug，output，warning

**-mac：**设置Host的MAC地址不变

--switch：设置switch类型

--innamespece：让switch工作在独立命名空间中。host在独立命名空间，switch和controller在ROOT命名空间。如sudo mn --innamespece --switch user

### CLI命令



mininet> nodes //查看全部节点

如果第一命令是node或switch的名字，则让其执行命令，例如

mininet> h2 ping h3

mininet> h2 ifconfig

//启动HTTP服务，发送请求，关闭HTTP服务

mininet> h1 python -m SimpleHTTPServer 80 &

mininet> h2 wget -O - h1

mininet> h1 kill %python

mininet> link s1 h2 up/down 启用/禁用链路

# dpctl dump-flows tcp:127.0.0.1:6634 显示交换机上所有流表

mininet> net 查看所有网络接口

mininet> xterm h1 开启h1的term终端

term> wireshark

### 自定义topo

sudo mn --controller=remote,ip=192.168.1.4  --topo mytopo --custom ~ / topo.py --mac

from mininet.topo import Topo

class MyTopo( Topo ):

"Simple topology example."

def \_\_init\_\_( self ):

"Create custom topo."

# Initialize topology

Topo.\_\_init\_\_( self )

# Add hosts and switches

Host1 = self.addHost( 'Client' )

Host2 = self.addHost( 'Server' )

Switch1 = self.addSwitch( 's1' )

Switch2 = self.addSwitch( 's2' )

# Add links

self.addLink( Switch1, Switch2 )

self.addLink( Switch2, Switch1 )

self.addLink( Switch1, Host1 )

self.addLink( Switch1, Host2 )

topos = { 'mytopo': ( lambda: MyTopo() ) }

mininet> xterm h1

xterm>wireshark

mininet> h3 python -m SimpleHTTPServer 80 >& /tmp/http.log &

mininet> h1 wget -O - h3

### Python API

通过python api定义拓扑，例子为mininet/custom/ topo-2sw-2host.py，命令为

sudo mn --custom ~/mininet/custom/topo-2sw-2host.py --topo mytopo --test pingall

from mininet.net import Mininet

from mininet.topolib import TreeTopo

tree4 = TreeTopo(depth=2,fanout=2)

net = Mininet(topo=tree4)

net.start()

h1, h4 = net.hosts[0], net.hosts[3]

print h1.cmd('ping -c1 %s' % h4.IP())

net.stop()

# OpenvSwitch

虚拟交换机，支持多种Linux虚拟化技术（如KVM，Xen/XenServer，VirtualBox）

支持标准接口管理协议（如NetFlow，sFlow(R)，802.1ag）

ovs-vswitchd：交换机进程，包括支持流交换的Linux内核模块；

ovsdb-server：轻量级数据库服务器进程，为ovs-vswitchd获取配置；

ovs-dpctl：配置交换机内核模块的工具；

用于在Citrix XenServer和Red Hat 安装的脚本

ovs-vsctl：ovs-vswitchd配置工具；

ovs-appctl：发送命令给ovs-vswitchd执行的工具。

其他工具：

ovs-ofctl：查询配置OpenFlow交换机和控制器的工具；

ovs-pki：为OpenFlow交换机创建并管理public-key的框架；

ovs-testcontroller：简单OpenFlow控制器，用于测试；

ovs-tcpundump：tcpdump的补丁，用于解析OpenFlow消息。

## 安装

查看FAQ中，OpenvSwitch版本对应支持的Linux内核版本范围，2.3.x对应2.6.32至3.14.x

依赖软件：GNU make，GCC 4.x，libc，libssl（用于OpenFlow控制器与交换机的通信加密），Python 2.x（x>= 4）

[root@localhost ~]# yum install openssl

[root@localhost ~]# ls -l /dev/urandom 检查/dev/urandom存在

crw-rw-rw- 1 root root 1, 9 Feb 3 17:43 /dev/urandom

[root@localhost ~]# ls -l /dev/net/tun 支持TAP的设备，检查/dev/net/tun存在

crw-rw-rw- 1 root root 10, 200 Feb 3 17:43 /dev/net/tun

下载解压，切换至root

[root@localhost ~]# tar zxvf openvswitch-2.3.1.tar.gz

[root@localhost ~]# mkdir openvswitch

[root@localhost openvswitch-2.3.1]# cd openvswitch-2.3.1/

[root@localhost openvswitch-2.3.1]# ./configure --prefix=/root/openvswitch --with-linux=/lib/modules/`uname -r`/build

安装模块

[root@localhost openvswitch-2.3.1]# make modules\_install

[root@localhost openvswitch-2.3.1]# modprobe openvswitch

查看openvswitch模块的信息

[root@localhost openvswitch-2.3.1]# modinfo openvswitch

[root@localhost openvswitch-2.3.1]# lsmod | grep openvswitch

openvswitch 88623 0

[root@localhost openvswitch-2.3.1]# modinfo /lib/modules/`uname -r`/kernel/net/bridge/bridge.ko

安装软件

[root@localhost openvswitch-2.3.1]# make

[root@localhost openvswitch-2.3.1]# make install

配置环境

[root@compute openvswitch-2.3.1]# vi ~/.bash\_profile

PATH=$PATH:$HOME/bin:~/openvswitch/bin:~/openvswitch/sbin

export PATH

[root@compute openvswitch-2.3.1]# source ~/.bash\_profile

**配置OpenvSwitch数据库**

[root@ localhost openvswitch-2.3.1]# ovsdb-tool create ~/openvswitch/etc/openvswitch/conf.db vswitchd/vswitch.ovsschema

启动ovs-vswitch之前，需要配置ovsdb-server数据库

[root@localhost openvswitch-2.3.1]# ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock --remote=db:Open\_vSwitch,Open\_vSwitch,manager\_options --private-key=db:Open\_vSwitch,SSL,private\_key --certificate=db:Open\_vSwitch,SSL,certificate --bootstrap-ca-cert=db:Open\_vSwitch,SSL,ca\_cert --pidfile --detach

[root@localhost openvswitch-2.3.1]# ps -ef | grep ovsdb-server

初始化数据库

[root@localhost openvswitch-2.3.1]# ovs-vsctl --no-wait init

启动Open vSwitch daemon，连接同一个Unix socket

[root@localhost openvswitch-2.3.1]# ovs-vswitchd --pidfile --detach

测试是否成功，创建网桥，添加eth0和vif1.0至网桥

[root@localhost openvswitch-2.3.1]# ovs-vsctl add-br br0

[root@localhost openvswitch-2.3.1]# ovs-vsctl add-port br0 eth0

[root@localhost openvswitch-2.3.1]# ovs-vsctl add-port br0 vif1.0

[root@localhost openvswitch-2.3.1]# ovs-vsctl show

停止Open vSwitch

% kill `cd /usr/local/var/run/openvswitch && cat ovsdb-server.pid ovs-vswitchd.pid`

# FlowVisor

FlowVisor是OpenFlow 控制器和交换机之间的透明代理，通过分片机制让每个控制器控制一个虚拟网络（slice），分片间互相隔离。

## 安装

推荐：4 Cores，4 GB java heap size

最小：2 Cores，1 GB java heap size

sudo apt-get install openjdk-7-jdk openjdk-7-jre openjdk-7-jre-lib

vi ~/.bashrc

export JAVA\_HOME=/usr/lib/jvm/java-7-openjdk-amd64

export JRE\_HOME=${JAVA\_HOME}/jre

export CLASSPATH=.:${JAVA\_HOME}/lib:${JRE\_HOME}/lib

export MAVEN\_HOME=/home/dwt/maven

export MAVEN\_OPTS="-Xms1024m -XX:MaxPermSize=256m"

PATH=${JAVA\_HOME}/bin:${JRE\_HOME}/bin:${MAVEN\_HOME}/bin:$PATH

export PATH

source ~/.bashrc

sudo apt-get install build-essential ant

git clone git://github.com/OPENNETWORKINGLAB/flowvisor.git

cd flowvisor

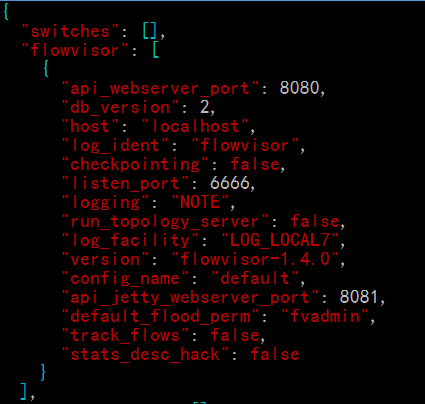
sudo useradd flowvisor

sudo make fvuser=flowvisor fvgroup=flowvisor install

flowvisor和opendaylight部署在同一台机器上，6633端口会冲突，修改flowvisor的listen\_port为6666

sudo -u flowvisor fvconfig generate /etc/flowvisor/config.json //创建配置文件

sudo vi /etc/flowvisor/config.json



启动

sudo -u flowvisor flowvisor /etc/flowvisor/config.json //采用修改后的配置文件启动

sudo /etc/init.d/flowvisor start //采用DB启动

sudo /etc/init.d/flowvisor status //查看是否启动

vi /var/log/flowvisor/flowvisor-stderr.log //查看出错日志

fvctl add-slice s1 tcp:localhost:6633 [localhost@gmail.com](mailto:localhost@gmail.com)

fvctl list-slice-info s1

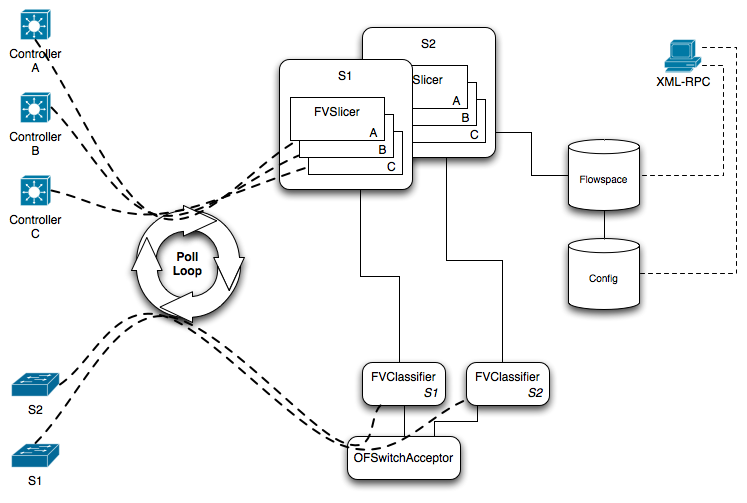
## CLI

|  |  |
| --- | --- |
| -u | User |
| -n | Hostname or IP of FlowVisor |
| -p | RPC Port Number (defaults to 8080) |

|  |  |
| --- | --- |
| Command | Description |
| add slice | Interactive tool for adding a slice |
| change slice "<name>" [flowvisor:flowspace] | Interactive tool for modifying a slice config  Using the optional flowspace parameter starts a sub-interpreter for manipulating the flowspace of a slice. |
| exit | Exits the CLI |
| no cache | Disables the device cache |
| quit | Exits the CLI |
| reset devicecache | Clears the device cache |
| reset linkcache | Clears the link cache |
| show device <dpid> [flowvisor:stats] | Get device information.  Optionally gather stats information from the device. |
| show device list [flowvisor:detail] | Show the list of DPIDs currently connected to FlowVisor.  This list is cached unless no cache is used or reset devicecache is executed.  The detail keyword will show the socket connection information as well. |
| show flowspace | Display the entire flowspace managed by this FlowVisor instance. |
| show link list | Display the topology that FlowVisor is aware of.  If FlowVisor does not have topology detection enabled, these links will be a best guess. |
| show slice list | Shows the list of configured slices. |
| show slice "<name>" [stats | flowvisor:flowspace] | Shows the slice configuration.  If the flowspace option is used, it shows the flowspace allocated to this slice.  If the stats option is used, your screen will be flooded with a wall of gibberish. |

## 使用

fvctl add-slice s1 tcp:localhost:6633 localhost@gmail.com



所有交换机和控制器的连接都通过一个Poll Loop。每个交换机通过一个FVClassifier与FlowVisor连接。每个分区生成一个FVSlicer。