

# OpenDaylight MD-SAL 应用开发入门

SDN QQ群：地球某某

QQ：564103786

Weibo：[Openflow我爱老婆](#)

# 本入门资料的面向对象

- \* 对象

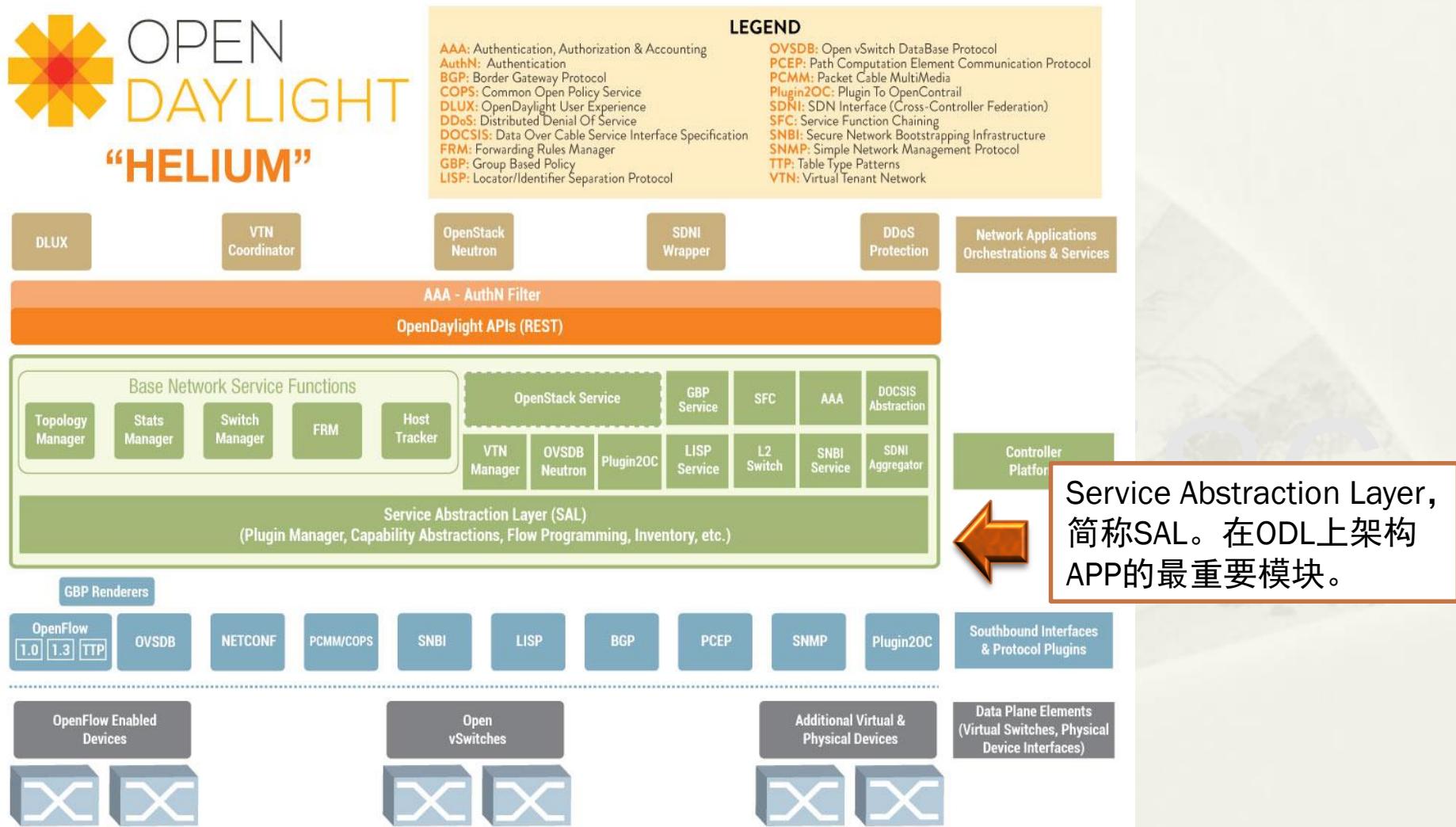
- \* SDN APP开发工程师。
  - \* 不甘被定义，想要突破自己的网工。

- \* 如果你有以下背景，看本资料会比较轻松

- \* 想用OpenDaylight Controller做为SDN应用平台。
  - \* 对OpenDaylight Project有一定了解。
  - \* 有少许Java编程基础。
  - \* 知道SDN是什么。
  - \* 对Maven, Xml, Git, Yang, Rest API, OSGI 技术有初步了解。

本入门教程只涉及纯技术，不涉及业界趋势，市场战略等。  
目的只有一个，尽量帮助你理解MD-SAL Plugin的开发流程，在开发时可以专注于  
网络功能和逻辑开发上，而不是花大量时间在研究MD-SAL Plugin开发方法上。

# ODL Helium版 软件架构



# Service Abstraction Layer - SAL

- \* 隶属于ODL Controller Platform。
- \* 由运行在OSGI Framework上的Bundle实现。
- \* 在ODL Controller Platform和Protocol Plugin之间实现抽象层的功能。
- \* 是ODL Controller Platform的最核心模块之一。
- \* 控制模块间数据交互，数据存储读取，API调用。
- \* 包含两个种类
  - \* AD-SAL (API-Driven SAL)
  - \* MD-SAL (Model-Driven SAL)

AD-SAL会逐渐被MD-SAL所替代而废弃掉。

在Helium版中，使用AD-SAL的模块和MD-SAL模块的应用都存在  
下一版Lithium版，基本大部分使用AD-SAL的应用都会移植到MD-SAL

# SAL中的一些用语

## \* Plugin

- \* 通过SAL实现的ODL Controller Framework的功能模块。
- \* 运行在ODL OSGI Framework上，与ODL控制器共享同一个JVM资源。
- \* 南向Plugin (Southbound Plugin , SB Plugin)
  - \* 向SAL提供管理、控制南向设备或服务的操作接口。
- \* 北向Plugin (Northbound Plugin, NB Plugin)
  - \* 通过利用SAL所提供的北向服务API，实现某些网络功能，并向应用提供统一的抽象服务和相应的API。

## \* 抽象服务(Abstraction Service)

- \* SAL所提供的网络抽象化服务。
  - \* E.g. Topology Service, Flow Service, Statistics Service...
- \* 南向服务API(Southbound Service API, SB API)
  - \* 通过南向Plugin实现管理、控制南向设备或服务。
- \* 北向服务API(Northbound Service API, NB API)
  - \* 向北向Plugin开放抽象服务。

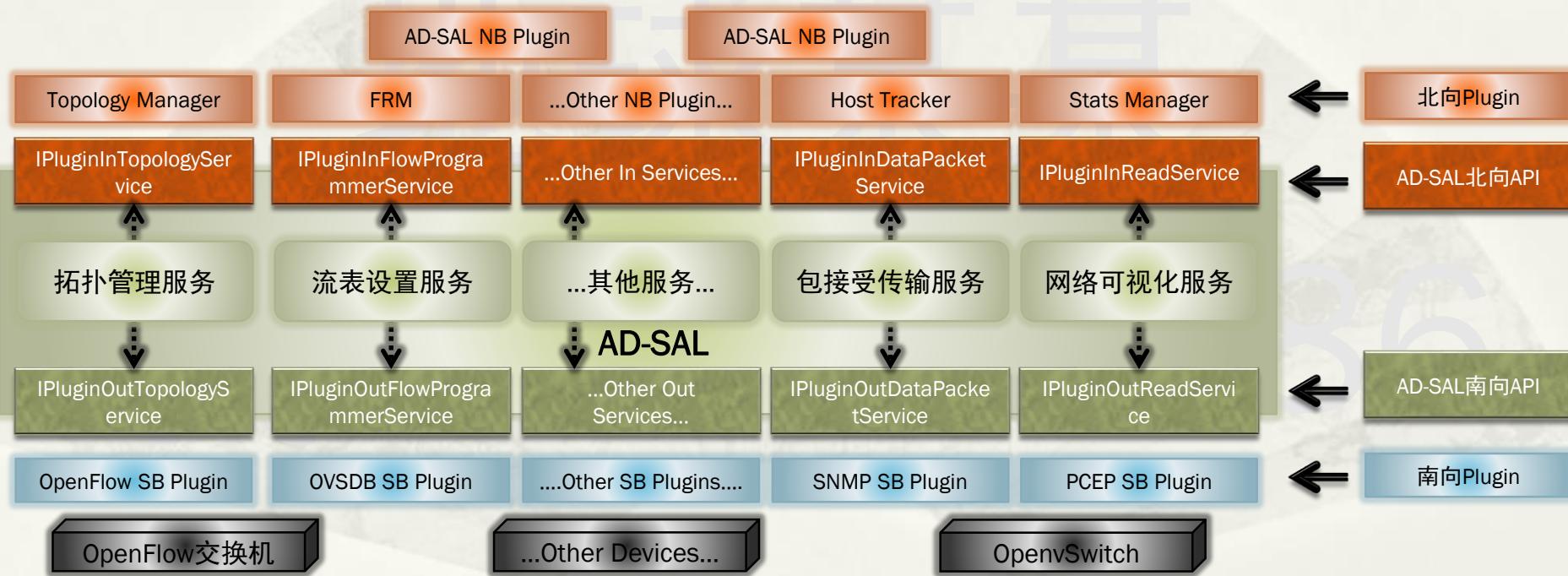
控制器角度所说的内部应用就是SAL的北向Plugin，只是用语不同而已



- \* 从ODL 控制器的角度来看，ODL应用 (Application, APP)分为内部应用和外部应用两种。
- \* 外部应用 (External APP)
  - \* 运行在ODL OSGI Framework外。
  - \* 利用ODL控制器和内部应用所提供的北向Rest API来实现的控制器外部应用。
- \* 内部应用(Internal APP)
  - \* 和ODL Controller Platform运行在同一OSGI Framework，分享相同的JVM资源。
  - \* 利用ODL控制器和其他内部应用所提供的API来实现的控制器内部应用。

# AD-SAL Overview

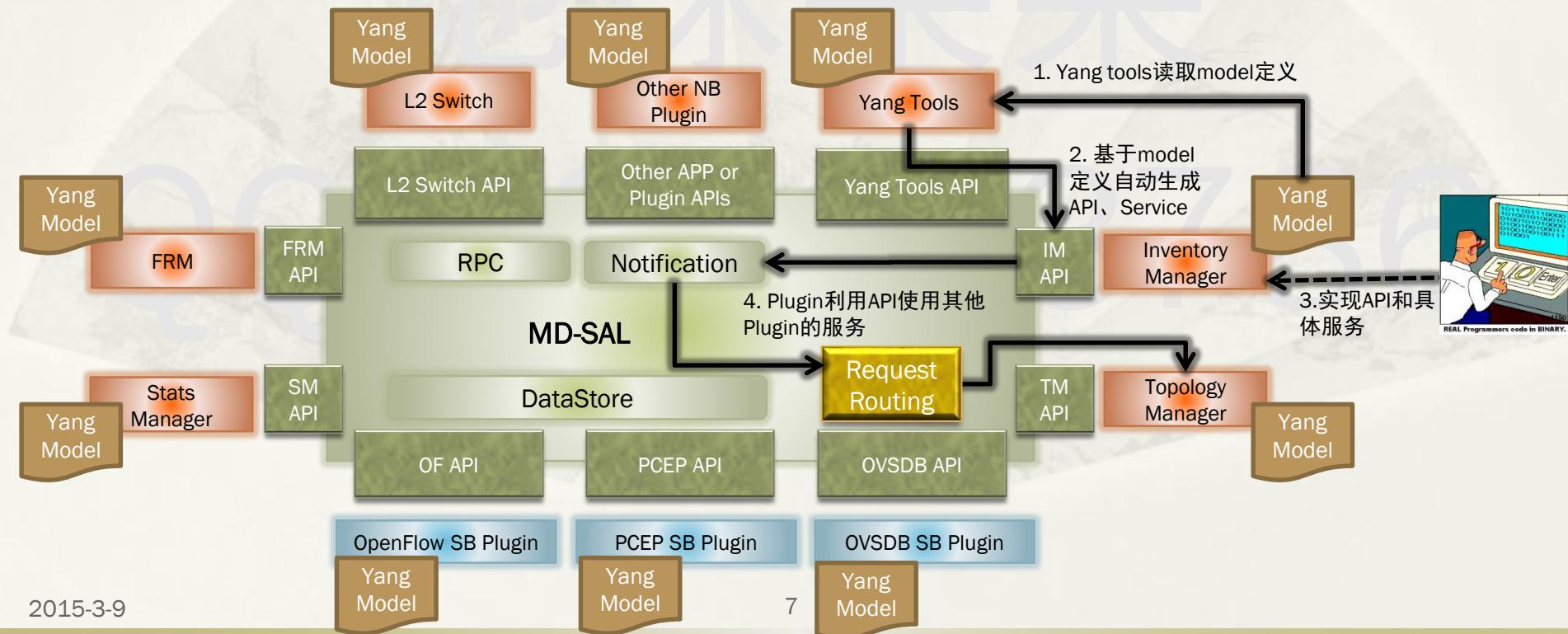
- \* 定义抽象服务，吸收南向协议差异，提供统一的抽象服务和API，并提供相应的Request Routing
- \* 北向Plugin可以通过调用AD-SAL的北向API来实现对南向Plugin的调用，操作其所管理的设备和服务。
- \* AD-SAL中，抽象服务由南向和北向API实现，南北向API是一对一映射关系。
- \* 开发者在使用AD-SAL开发时需要考虑到下层协议Plugin对抽象层所提供的功能的支持程度。



AD-SAL中，南北向API是1：1的对应关系，同一API无法被复用。所有南北向Plugin的功能都需要定义相应的AD-SAL API来承载，造成AD-SAL模块肥大化、实现复杂化、维护集中化，影响整个软件架构的可扩展性和可维护性。

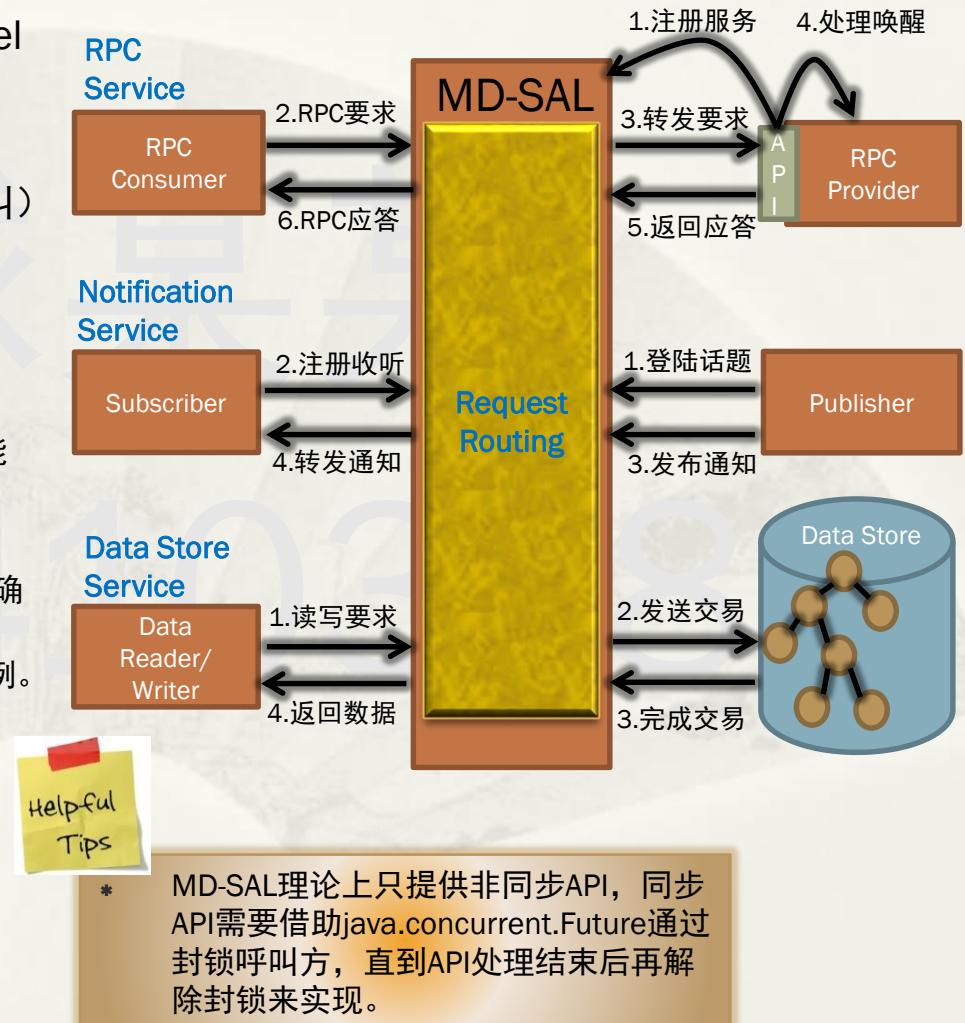
# MD-SAL Overview

- \* 提供Request Routing和用来定义抽象服务和相应API的基础框架。
  1. 抽象服务和相应API是由各个Plugin通过yang model和service来定义，而不是由MD-SAL定义。
  2. Yang Tools Plugin通过各个Plugin的model定义来自动生成API、Service Interface和相应Java代码。
  3. 开发者通过实现自动生成的Service Interface来实现具体的API和服务内容。
  4. Plugin通过MD-SAL和生成的API（RPC, Notification）、DataStore去利用其他各个Plugin的服务和数据。
- \* 所有功能模块的信息交互，数据存储调用都通过MD-SAL完成。



# MD-SAL的主要功能

- \* MD-SAL的主要功能就是管理基于Yang Model (RFC 6020) 定义的各种Plugin。
- \* 主要提供以下功能
  - \* **RPC** (Remote Procedure Call – 远程功能呼叫)
    - \* 提供服务的远程呼叫接口
  - \* **Notification** (通知)
    - \* 提供Notification收听, 发行等功能。
  - \* **Data Store** (数据存储)
    - \* 提供数据存储, 读取, Transaction处理等功能
  - \* **Request Routing** (要求定位)
    - \* 提供要求定位功能, 把外部来的RPC, Notification和Data Store要求定位、传送到正确的Plugin和Node Instance去进行处理。
    - \* Node Instance指的是Yang结构树上的节点实例。
  - \* RestConf Subsystem
    - \* 自动定义和创建RestConf API的Plugin
  - \* Config Subsystem
    - \* 提供统一的配置文件管理功能的Plugin



# MD-SAL Plugin

- \* 根据Plugin和MD-SAL的关联方式，Plugin分为如下两种。
  - \* BA (Binding-Aware)
    - \* 指使用根据Yang Model定义而自动生成的Java Bindings的Plugin。
  - \* BI (Binding-independent)
    - \* 与BA相反，BI指的是使用DOM (Document Object Model) 格式编程接口的API，不依赖于Java Bindings的Plugin。
- \* 根据Plugin的数据消费方式，Plugin分为如下三种。
  - \* Consumer
    - \* 使用其他Plugin所提供的Service和API来实现某些功能的Plugin。
  - \* Provider
    - \* 向其他Plugin提供Service和API的Plugin。
  - \* Broker
    - \* 同时具有Consumer和Provider特性的Plugin



\* Java Bindings指的是由Yang Tools根据定义的Yang Model自动生成的Service或API的Java代码。

本教程只涉及BA种类的Consumer, Provider, 当然也包括Broker。

# MD-SAL Plugin Dev Life Cycle

- \* 开发一个简单的实验原型级别的MD-SAL Plugin，在使用ODL Helium的前提下，大体要完成如下步奏。
  - \* 最重要的是想好要做什么。
    - \* Consumer? Provider? Broker?
    - \* 用不用DataStore? 大体要用到哪些Plugin提供的功能?
    - \* 等等
  - \* 创建Package和Modules。
  - \* 配置Maven pom.xml。
  - \* 定义Yang Model。
  - \* 利用Java Bindings实现具体的Service或者API。
  - \* 关联Config Subsystem。
  - \* 搞清依存关系，把Plugin打成Karaf Feature。
  - \* 把Feature导入到ODL Helium。
  - \* 通过RestConf或者APIdocs测试看效果。

为了简化内容、深化基础，关于Cluster, HA, Test Code等不在本教材中涉及。

# 本教程所要讲解的例子

## \* 基础篇

- \* 简单的RPC例子
  - \* Provider
  - \* Consumer
- \* 简单的Notification例子
- \* 简单的Data Store例子
  - \* 写入
  - \* 读取

## \* 应用篇

- \* 使用I2switch和openflowplugin的简单的Mac Filter的例

在例子的讲解中会涉及到Maven, Xml, Yang等技术，本资料只对所涉及的部分作简单讲解，不做相应技术的全面讲解。想要详细了解的话，网上很多资料，请自行Google。

# 开始前的开发环境配置

- \* ODL开发需要的tool和lib
  - \* git
  - \* Maven 3.0.4以上
  - \* Java 7 JDK
- \* 环境变量配置
  - \* 配置Maven环境变量 (\$M2\_HOME, \$PATH)
  - \* 配置JDK环境变量 (\$JAVA\_HOME)
  - \* 配置Maven配置文件 (~/.m2/settings.xml)
- \* 配置方法参照：
  - \* 配置Maven和JDK环境变量
    - \* 根据使用的OS在网上google下，有很多。
  - \* 配置Maven配置文件
    - \* [https://wiki.opendaylight.org/view/GettingStarted:Development\\_Environment\\_Setup](https://wiki.opendaylight.org/view/GettingStarted:Development_Environment_Setup)

本资料涉及的开发流程是以使用Linux的bash为前提，  
不包括使用Eclipse等IDE开发环境的情况。

# 基础篇

简单的RPC例子 - Provider

# Sample Source Code

- \* 例子的Source Code在以下github中，请自行下载。

- \* [https://github.com/eartheart/odl\\_helium\\_md-sal.git](https://github.com/eartheart/odl_helium_md-sal.git)

- \* 包括source code和编译后的生成制品。

```
[root@odl ~]# git clone https://github.com/eartheart/odl_helium_md-sal.git
```

- \* 下载后的目录路径如下。

```
[root@odl odl_helium_md-sal]# pwd  
/root/odl_helium_md-sal  
[root@odl odl_helium_md-sal]# ls  
README.md  sdnsp
```

- \* **注意：**在后面的例子说明部分，是使用sdnap为最上层目录来讲解的，而非odl\_helium\_md-sal/sdnsp。

- \* 下载后你可能的目录路径：

```
[root@odl sdnsp]# pwd  
/root/odl_helium_md-sal/sdnsp
```

- \* 后面例子说明中使用的目录路径：

```
[root@odl sdnsp]# pwd  
/root/sdnsp
```

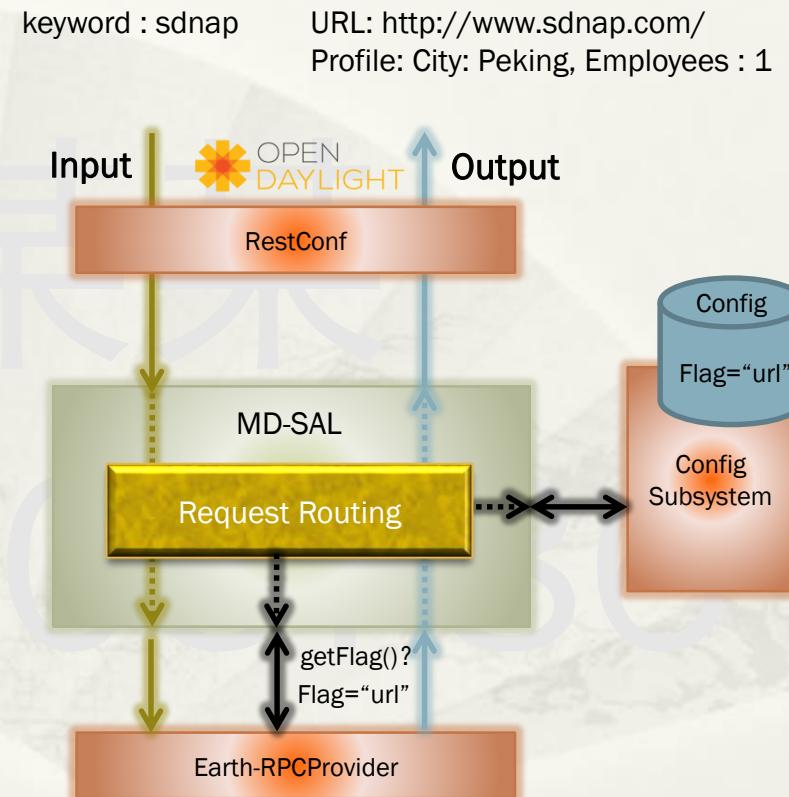
# Steps

---

- \* 想好要做什么
- \* 创建Package和Modules

# 想好要做什么

- \* 做一个简单的RPC Provider例子
  - \* rpcprovider
    - \* 一个简单的SDN学习网站、QQ群查询功能
- \* Specification
  - \* 接收输入，根据输入返回相应网站、群信息。
  - \* 通过Config Subsystem取得Flag的配置。
  - \* 当配置文件的Flag = “url”的时候：
    - \* keyword是sdnap，返回sdnap网址和Profile
      - \* URL: http://www.sdnnap.com/
      - \* Profile: City: Peking, Employees : 1
    - \* keyword是sdnlab，返回sdnlab网址和Profile
      - \* URL: http://www.sdnlab.com/
      - \* Profile: City : Nanking, Employees : 2
  - \* 当配置文件的Flag = “qq”的时候：
    - \* keyword是sdnap，返回sdnap群和Profile
      - \* QQ: 279796875
      - \* Profile: City : Peking, Employees : 1
    - \* keyword是sdnlab，返回sdnlab群和Profile
      - \* QQ: 194240432
      - \* Profile: City : Nanking, Employees : 2



# Steps

---

- \* 想好要做什么
- \* 创建Package和Modules

# 创建Package和Modules

- \* 上层ODL Package
  - \* org.opendaylight
- \* 本资料例子的Package
  - \* org.opendaylight.sdnapp
- \* 在sdnap下创建Plugin的目录
  - \* rpcprovider
- \* 在sdnap/rpcprovider下创建以下四个子modules的目录
  - \* model : 定义service和服务 API (yang model)
  - \* implementation: 实现具体service (yang model, java code)
  - \* configuration: 定义配置文件和配置信息API (yang model, xml配置文件)
  - \* karaf: 定义Karaf Feature和Feature间的相互依赖关系 (xml配置文件)

```
[root@odl rpcprovider]# pwd  
/root/sdnapp/rpcprovider  
[root@odl rpcprovider]# ls  
configuration implementation karaf model  
[root@odl rpcprovider]#
```

# Steps

---

- \* 配置rpcprovider的Maven pom.xml
- \* 制作Model Module
- \* 制作Configuration Module
- \* 制作Implementation Module
- \* 制作Karaf Module

# XML Namespace

- \* 本资料中所有pom.xml都使用以下namespace。

```
<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">
```

- \* 在例子讲解中，上述namespace和以下xml header都被省略掉了，如果需要请自行添加。

```
<?xml version="1.0" encoding="UTF-8"?>
```

推荐先从github把例子checkout出来，然后使用本资料对照着阅读。

# 配置rpcprovider的Maven pom.xml

- \* 在sdnap/rpcprovider下创建rpcprovider Plugin的最上层pom.xml

```
[root@odl rpcprovider]# pwd  
/root/sdnan/rpcprovider  
[root@odl rpcprovider]# vim pom.xml[]
```

- \* 配置Plugin最上层Maven pom.xml

```
<modelVersion>4.0.0</modelVersion>
```

指定Maven model的版本号。

```
<parent>  
  <groupId>org.opendaylight.odlparent</groupId>  
  <artifactId>odlparent</artifactId>  
  <version>1.4.4-Helium-SR2</version>  
  <relativePath />  
</parent>
```

Plugin的预定工作平台是Helium SR2，指定继承1.4.4-Helium-SR2的odlparent的配置。  
odlparent在远程Repository，所以  
<relativePath>设为空。  
odlparent的版本信息可以通过查看相应  
branch, tag的pom.xml来确认。

```
<groupId>org.opendaylight.sdnan</groupId>  
<artifactId>rpcprovider-top</artifactId>  
<version>1.0.0-Helium-SR2</version>  
<name>${project.artifactId}</name>
```

设定Plugin最上层Maven工程的信息。  
artifactId设为rpcprovider-top, version设为  
1.0.0-Helium-SR2。

```
<packaging>pom</packaging>
```

指定使用pom (Project Object Model)。

# 配置rpcprovider的Maven pom.xml

- \* 设定本Plugin开发时使用的一些共同配置的变量。
  - \* 可以通过\${变量名}在相同pom.xml或者继承者的pom.xml中参照这些指定的配置。
- \* 自动生成的代码的路径配置的具体使用和内容会在后面相应章节中说明。

```
<properties>
    <salGeneratorPath>
        ${project.build.directory}/generated-sources/yang-gen-sal
    </salGeneratorPath>

    <jmxGeneratorPath>
        ${project.build.directory}/generated-sources/yang-gen-config
    </jmxGeneratorPath>

    <configFilePath>
        ${project.build.directory}/classes/initial/66-rpcprovider-config.xml
    </configFilePath>

    <featurePath>${project.build.directory}/classes/features.xml</featurePath>
    <karFilePath>${project.build.directory}/${project.artifactId}-${project.version}.kar</karFilePath>

    <yangtools.version>0.6.4-Helium-SR2</yangtools.version>
    <mdsal.version>1.1.2-Helium-SR2</mdsal.version>
    <config.version>0.2.7-Helium-SR2</config.version>
    <mojo.build.helper.version>1.8</mojo.build.helper.version>
    <maven.resources.version>2.7</maven.resources.version>
    <karaf.version>3.0.1</karaf.version>
    <rpcprovider.version>1.0.0-Helium-SR2</rpcprovider.version>
</properties>
```

指定自动生成（通过yang model）的sal的java bindings的路径。  
\${project.build.directory} 指的是编译时，编译对象的pom.xml的所在目录路径。

指定自动生成（通过yang model）的config subsystem相关的java bingdings的路径。

指定本Plugin的配置文件目录。

指定本Plugin的karaf feature配置文件目录。

指定本Plugin的kar文件输出目录。

指定一些共同的版本信息。

# 配置rpcprovider的Maven pom.xml

- \* 指定所需要的OSGI bundle依赖和编译工具信息。

```
<dependencies>  
    <dependency>  
        <groupId>org.opendaylight.yangtools</groupId>  
        <artifactId>yang-binding</artifactId>  
        <version>${yangtools.version}</version>  
    </dependency>  
    <dependency>  
        <groupId>org.opendaylight.yangtools</groupId>  
        <artifactId>yang-common</artifactId>  
        <version>${yangtools.version}</version>  
    </dependency>  
</dependencies>
```

指定yang model生成的Java bindings的依赖关系，这里指定依赖yang-binding和yang-common。

```
<build>  
    <plugins>  
        <plugin>  
            <groupId>org.apache.felix</groupId>  
            <artifactId>maven-bundle-plugin</artifactId>  
            <version>${maven.bundle.version}</version>  
            <extensions>true</extensions>  
        </plugin>  
    </plugins>  
</build>
```

配置编译信息，指定编译时使用的plugin。这里指定使用maven-bundle-plugin。

# 配置rpcprovider的Maven pom.xml

## \* 指定关联modules

```
<modules>  
    <module>model</module>  
    <module>implementation</module>  
    <module>configuration</module>  
    <module>karaf</module>  
</modules>
```

指定一开始我们创建的4个  
modules的目录: model,  
configuration, implementation,  
karaf

# odlparent

- \* odlparent的pom.xml中定义了它所在版本中所有OpenDaylight应用的一些共同配置（Lib的版本号，依赖条件等等）。
- \* 开发相应版本的应用时，Maven配置最好继承该版本的odlparent配置。

```
[root@odl odlparent]# pwd  
/root/mydata/odl_source/odlparent  
[root@odl odlparent]# git branch  
* helium-sr2  
  master  
  stable/helium  
[root@odl odlparent]# ls  
pom.xml  
[root@odl odlparent]# vim pom.xml
```

- \* 打开相应OpenDaylight版本的odlparent的pom.xml。
  - \* 相应OpenDaylight版本指你所开发的Plugin将要运行的ODL平台的版本。本资料中使用最新版Helium-SR2(2015年3月1日截至)。

```
<groupId>org.opendaylight.odlparent</groupId>  
<artifactId>odlparent</artifactId>  
<version>1.4.4-Helium-SR2</version>  
<packaging>pom</packaging>
```

前面slide中最上层pom的<parent>要素的信息就是根据这里设定的。

# Steps

---

- \* 配置rpcprovider的Maven pom.xml
- \* 制作Model Module
- \* 制作Configuration Module
- \* 制作Implementation Module
- \* 制作Karaf Module

# Model Module

---

- \* 在model module中定义rpcprovider的Service API（种类， 输入输出等）。
- \* 通过YangTools Plugin的yang-maven-plugin自动生成Java bindings。
- \* 生成提供Java bindings的OSGI Bundle。

# 配置Model Module的pom.xml

- \* 在sdnap/rpcprovider/model下创建model的pom.xml

```
[root@odl model]# pwd  
/root/sdnan/rpcprovider/model  
[root@odl model]# vim pom.xml []
```

- \* 配置model的pom.xml

```
<modelVersion>4.0.0</modelVersion>  
  
<parent>  
  <groupId>org.opendaylight.sdnan</groupId>  
  <artifactId>rpcprovider-top</artifactId>  
  <version>1.0.0-Helium-SR2</version>  
  <relativePath>..</relativePath>  
</parent>  
  
<groupId>org.opendaylight.sdnan</groupId>  
<artifactId>model</artifactId>  
<version>1.0.0-Helium-SR2</version>  
<name>${project.artifactId}</name>  
  
<packaging>bundle</packaging>
```

继承rpcprovider plugin的最上层pom.xml。  
<relativePath>指定其相对路径为上一层目录。

设定当前的model module的信息。

指定model module以osgi bundle形式编译打包。

# 配置Model Module的pom.xml

- \* 配置OSGI Bundle依赖。

```
<dependencies>  
    <dependency>  
        <groupId>org.opendaylight.yangtools.model</groupId>  
        <artifactId>ietf-inet-types</artifactId>  
        <version>2010.09.24.7-SNAPSHOT</version>  
    </dependency>  
</dependencies>
```

指定依赖信息。本例子后面会用到 yang model的ietf-inet-types。



- \* Yang Tools中可以使用的yang model如下（2015年3月1日截止）
  - \* IETF: ietf-inet-types; ietf-yang-types; ted; network-topology; isis-topology; l3-unicast-igp-topology; ospf-topology
  - \* IANA: iana-afn-safi; iana-if-type
  - \* ODL L2 Types:.opendaylight-l2-types
  - \* Yang Extensions for OpenDaylight: ODL Yang Extensions
- \* 使用时需要指定的信息和其他详细内容请参照以下URL
  - \* [https://wiki.opendaylight.org/view/YANG\\_Tools:Available\\_Models](https://wiki.opendaylight.org/view/YANG_Tools:Available_Models)

# 配置Model Module的pom.xml

- \* 配置yang-maven-plugin编译信息。

```
<build>
  <plugins>
    <plugin>

      <groupId>org.opendaylight.yangtools</groupId>
      <artifactId>yang-maven-plugin</artifactId>
      <version>${yangtools.version}</version>

    </plugin>
  </plugins>
<dependencies>

  <dependency>
    <groupId>org.opendaylight.yangtools</groupId>
    <artifactId>maven-sal-api-gen-plugin</artifactId>
    <version>${yangtools.version}</version>
  </dependency>

</dependencies>
```

指定使用yang-maven-plugin

配置yang-maven-plugin编译时生成SAL API的依赖信息。



编译信息的配置还没有结束，下一页slide继续。

# 配置Model Module的pom.xml

```
<executions>
  <execution>
    <goals>
      <goal>generate-sources</goal>
    </goals>
    <configuration>
      <yangFilesRootDir>src/main/yang</yangFilesRootDir>
      <codeGenerators>
        <generator>
          <codeGeneratorClass>
            org.opendaylight.yangtools.maven.sal.api.gen.plugin.CodeGeneratorImpl
          </codeGeneratorClass>
          <outputBaseDir>${salGeneratorPath}</outputBaseDir>
        </generator>
      </codeGenerators>
      <inspectDependencies>true</inspectDependencies>
    </configuration>
  </execution>
</executions>
</plugin>
</plugins>
</build>
```

指定执行Plugin的目的为 generate-resources。  
(生成Java bindings)

指定yang module文件的目录路径。

配置代码生成器。  
<outputBaseDir>指定生成的SAL API的java bindings代码的输出目录路径。

# 在Model Module中创建Yang Model

- \* 在model module下创建yang model定义文件目录。
  - \* src/main/yang

```
[root@odl model]# pwd  
/root/sdnapp/rpcprovider/model  
[root@odl model]# cd src/main/yang/  
[root@odl yang]# pwd  
/root/sdnapp/rpcprovider/model/src/main/yang  
[root@odl yang]# 
```

- \* 在src/main/yang目录下创建yang module定义文件。
  - \* Yang module的作用是把多个service归类。
  - \* 一个MD-SAL Plugin可以包含一个或者多个yang module。
  - \* Yang module的主文件名和该文件定义的yang module的名字要保持一致。
    - \* Model module的yang module取名为 earth-rpcprovider，所以文件名如下。

```
[root@odl yang]# pwd  
/root/sdnapp/rpcprovider/model/src/main/yang  
[root@odl yang]# ls  
earth-rpcprovider.yang  
[root@odl yang]# 
```

# 定义Yang Module

## earth-rpcprovider.yang

- \* 定义earth-rpcprovider.yang

```
[root@odl yang]# pwd  
/root/sdnap/rpcprovider/model/src/main/yang  
[root@odl yang]# vim earth-rpcprovider.yang []
```

```
module earth-rpcprovider {
```

指定module名字（和文件名一致）

```
yang-version 1;
```

指定yang的版本号。  
指定module的namespace，module的namespace不能重复。  
指定被参照、引用时的prefix。

```
namespace "urn:.opendaylight:sdnap:rpcprovider";  
prefix provider;
```

```
import ietf-inet-types {
```

导入要使用的其他yang module。这里  
import ietf-inet-types (RFC 6021)

```
prefix inet;  
revision-date 2010-09-24;  
}
```

```
description "Model definition for sdnap earth-rpcprovider.";
```

设定module描述和版本信息。

```
revision 2015-03-01 {
```

```
description "Initial revision of earth-rpcprovider.";  
}
```

# 定义Yang Module

## earth-rpcprovider.yang

```
typedef keyword-type {  
    type string {  
        length "1 .. 8";  
        pattern "[a-zA-Z]*";  
    }  
}  
  
grouping profile {  
  
    leaf city {  
        type string;  
    }  
  
    leaf employees {  
        type uint32;  
    }  
}
```

定义类型keyword-type为长度1到8的由半角字母组成的字符串。

把city和employees归类定义为profile。

# 定义Yang Module

## earth-rpcprovider.yang

```
rpc earth-rpcprovider {
```

设定rpc名字。

```
    input {  
        leaf keyword {  
            type keyword-type;  
        }  
    }
```

定义rpc的输入keyword。Keyword的类型设为前面定义的keyword-type类型。

```
    output {
```

定义rpc的输出url, qq, city, employees四个输出。city和employees在前面被归类在profile, 在这里直接引入profile。

```
        leaf url {  
            type inet:uri;  
        }
```

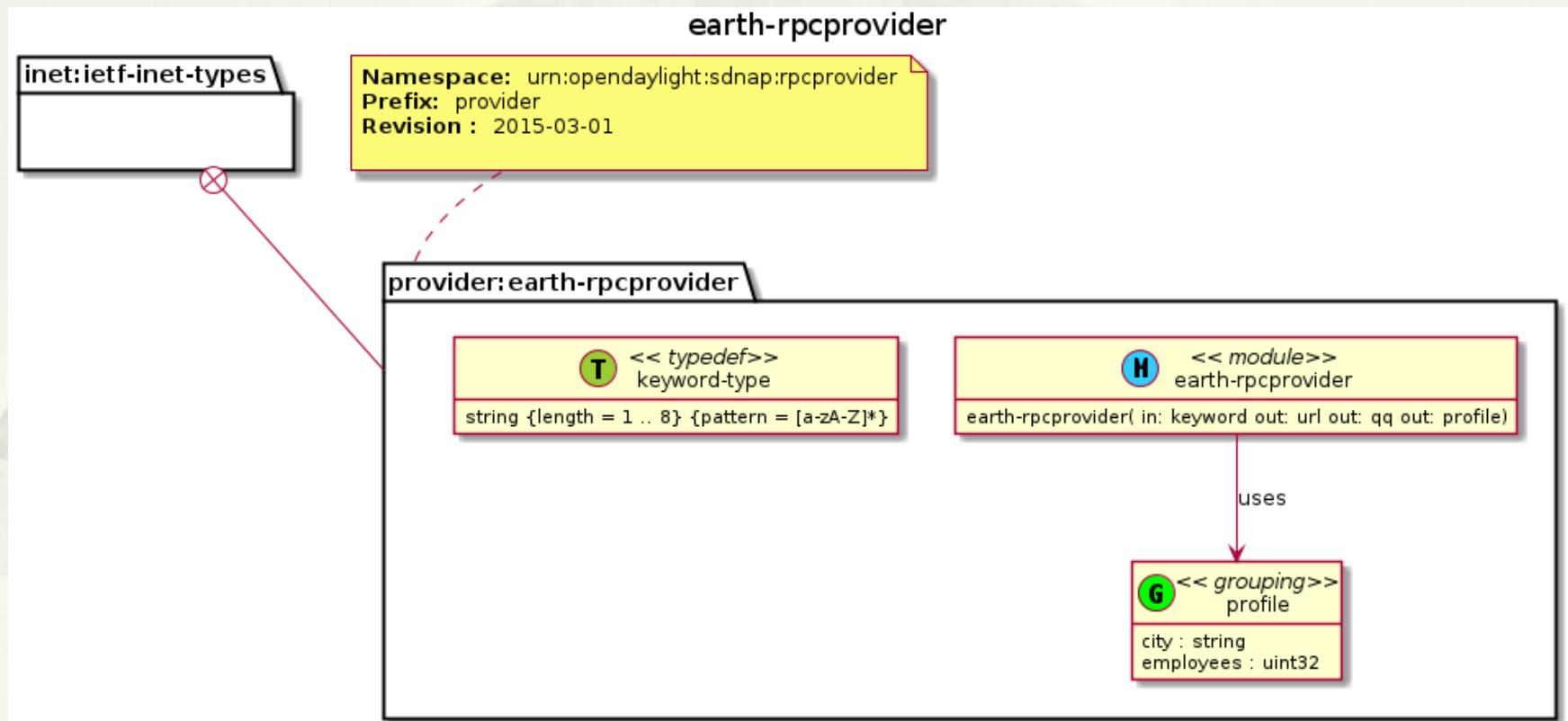
```
        leaf qq {  
            type string;  
        }
```

```
        uses profile;  
    }
```

```
}
```

# UML Diagram

- \* 定义好的earth-rpcprovider module的UML图如下：



# Tree Structure

Module: [earth-rpcprovider](#), Namespace: [urn:opendaylight:sdnab:rpcprovider](#), Prefix: [provider](#)

Element [\[+\]](#)[Expand all](#) [\[-\]](#)[Collapse all](#)

	Schema	Type	Flags	Opts	Status	Path
▶ <a href="#">earth-rpcprovider</a>		module				
▶ <a href="#">provider:rpcs</a>						
▶ <a href="#">earth-rpcprovider</a>	rpc				current	/provider:earth-rpcprovider
▶ <a href="#">input</a>	input		no config		current	/provider:earth-rpcprovider/provider:input
▶ <a href="#">keyword</a>	leaf	keyword-type	no config	?	current	/provider:earth-rpcprovider/provider:input/provider:keyword
▶ <a href="#">output</a>	output		no config		current	/provider:earth-rpcprovider/provider:output
▶ <a href="#">url</a>	leaf	inet:uri	no config	?	current	/provider:earth-rpcprovider/provider:output/provider:url
▶ <a href="#">qq</a>	leaf	string	no config	?	current	/provider:earth-rpcprovider/provider:output/provider:qq
▶ <a href="#">city</a>	leaf	string	no config	?	current	/provider:earth-rpcprovider/provider:output/provider:city
▶ <a href="#">employees</a>	leaf	uint32	no config	?	current	/provider:earth-rpcprovider/provider:output/provider:employees

# Java Bindings

---

- \* 根据pom.xml的配置， yang-maven-plugin会把yang module文件自动编译成Java代码。
  - \* 我们需要了解这些代码和使用方法，才能通过他们实现实际的服务和API。
- \* 本资料中每个例子都会对该例子中使用的yang module的Java Bindings进行讲解。



# Java Bindings的一些生成规则

- \* 生成的Java bindings的Package名
  - \* 基本Package名是根据yang module中设定的namespace和revision决定的。
  - \* 其他的Package名是由相应的节点名和基本Package名组成的。

```
module earth-rpcprovider {  
  
    yang-version 1;  
    namespace "urn:opendaylight:sdnab:rpcprovider";  
    prefix provider;  
    //...snip...  
    revision 2015-03-01 {  
        description "Initial revision of rpcprovider model.";  
    }  
}
```



```
package org.opendaylight.yang.gen.v1.urn.opendaylight.sdnab.rpcprovider.rev150301;
```



# Java Bindings的一些生成规则

- \* 生成的Java Class, Interface, Property等的名称
  - \* 生成Java代码的时候会把yang module定义中的名称的“-”去掉，转换成 CamelCase标记法。

```
typedef keyword-type {  
    type string {  
        length "1 .. 8";  
        pattern "[a-zA-Z]*";  
    }  
}
```



```
public class KeywordType {  
    //...snip...  
}
```

- \* 生成的Method
  - \* 自动生成Property的getter和setter method

```
rpc earth-rpcprovider {  
    input {  
        leaf keyword {  
            type keyword-type;  
        }  
    }  
}
```



```
public KeywordType getKeyword() {  
    //...snip...  
}
```



```
public EarthRpcproviderInputBuilder  
setKeyword(KeywordType value) {  
    //...snip...  
}
```

# earth-rpcprovider Module的Java Bindings

- \* 根据model module的pom.xml配置， SAL Java bindings会被输出到以下目录。
  - \* \${project.build.directory}/generated-sources/yang-gen-sal
- \* 编译后model module的SAL Java bindings的基本输出目录。

```
[root@odl yang-gen-sal]# pwd  
/root/sdnapp/rpcprovider/model/target/generated-sources/yang-gen-sal  
[root@odl yang-gen-sal]# []
```

- \* SAL Java package的输出目录
  - \* SAL Java bindings的基本输出目录 + Package目录

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapp.rpcprovider.rev150301;
```

```
[root@odl rev150301]# pwd  
/root/sdnapp/rpcprovider/model/target/generated-sources/yang-gen-sal/org/opendaylight/yang/gen/v1/urn/opendaylight/sdnapp/rpcprovider/rev150301
```



- \* 在所有module的pom.xml没有完全配置好前，建议单独编译相应module。

```
[root@odl model]# pwd  
/root/sdnapp/rpcprovider/model  
[root@odl model]# mvn clean install -DskipTests[]
```

# earth-rpcprovider Module的Java Bindings

- \* 生成的SAL Java bindings。

```
[root@v157-7-239-39 rev150301]# pwd  
/root/sdnap/rpcprovider/model/target/generated-sources/yang-gen-sal/org/opendaylight/yang/gen/v1/urn/opendaylight/sdnap/rpcprovider/rev150301  
[root@v157-7-239-39 rev150301]# find ./ | fgrep .java  
./EarthRpcproviderInput.java  
./EarthRpcproviderInputBuilder.java  
./Profile.java  
./$YangModelBindingProvider.java  
./EarthRpcproviderOutputBuilder.java  
./$YangModuleInfoImpl.java  
./EarthRpcproviderService.java  
./EarthRpcproviderOutput.java  
./KeywordType.java  
[root@v157-7-239-39 rev150301]# □
```

# earth-rpcprovider Module的Java Bindings

- \* typedef所定义的类型会生成相应的Java Class。

```
typedef keyword-type {
    type string {
        length "1 .. 8";
        pattern "[a-zA-Z]*";
    }
}
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.rpcprovider.rev150301.KeywordType
```

```
public class KeywordType {
    //...snip...
    public KeywordType(java.lang.String _value) {
        //...snip...
    }
    //...snip...
}
```

# earth-rpcprovider Module的Java Bindings

- \* grouping所定义的归类会生成相应的Java Interface。

```
grouping profile {  
    leaf city { type string; }  
    leaf employees { type uint32; }  
}
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapp.rpcprovider.rev150301.Profile
```

```
public interface Profile extends DataObject {  
    public static final QName QNAME =\\  
        org.opendaylight.yangtools.yang.common.QName.create\\  
        ("urn:opendaylight:sdnapp:rpcprovider","2015-03-01","profile");;  
    java.lang.String getCity();  
    java.lang.Long getEmployees();  
}
```

# earth-rpcprovider Module的Java Bindings

- \* rpc所定义的RPC会生成Java Interface。
  - \* 生成的RPC的Interface名是RPC名 + “Service”字符串。
  - \* 同时还会生成所定义的Input和Output的Builder和Interface。

```
rpc earth-rpcprovider {  
    input {  
        leaf keyword { type keyword-type; }  
    }  
    output {  
        leaf url { type inet:uri; }  
        leaf qq { type string; }  
        uses profile;  
    }  
}
```



org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.rpcprovider.rev150301.EarthRpcproviderService

```
public interface EarthRpcproviderService extends RpcService {  
    Future<RpcResult<EarthRpcproviderOutput>> earthRpcprovider(EarthRpcproviderInput input);  
}
```

# earth-rpcprovider Module的Java Bindings

- \* Input的Builder和Interface。
  - \* Interface名为RPC名 + “Input”字符串。
  - \* Build Class名为Interface名 + “Builder”字符串。



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.EarthRpcproviderInputBuilder
```

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.EarthRpcproviderInput
```

```
public interface EarthRpcproviderInput extends DataObject, \  
Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.\  
EarthRpcproviderInput> {  
    public static final QName QNAME = org.opendaylight.yangtools.yang.common.QName.create\  
("urn:opendaylight:sdnapi:rpcprovider", "2015-03-01", "input");  
    KeywordType getKeyword();  
}
```

# earth-rpcprovider Module的Java Bindings

- \* Output的Builder和Interface。
  - \* Interface名为RPC名 + “Output”字符串。
  - \* Build Class名为Interface名 + “Builder”字符串。



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.EarthRpcproviderOutputBuilder
```

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.EarthRpcproviderOutput
```

```
public interface EarthRpcproviderOutput extends Profile, DataObject, \  
Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapi.rpcprovider.rev150301.EarthRpcpro  
viderOutput> {  
    public static final QName QNAME = org.opendaylight.yangtools.yang.common.QName.create\  
("urn:opendaylight:sdnapi:rpcprovider", "2015-03-01", "output");  
    Uri getUrl();  
    java.long.String getQq();  
}
```

# Steps

---

- \* 配置rpcprovider的Maven pom.xml
- \* 制作Model Module
- \* 制作Configuration Module
- \* 制作Implementation Module
- \* 制作Karaf Module

# Configuration Module

- \* 通过build-helper-maven-plugin指定向MD-SAL Config Subsystem提供的配置文件信息。
- \* 在Configuration Module中定义配置文件。
- \* 生成提供配置文件的Jar。



- \* MD-SAL Config Subsystem
  - \* 统一管理MD-SAL Plugin的配置文件。
  - \* 管理MD-SAL Plugin的启动、初始化和启动时必要的配置注入。
  - \* 配置注入是通过NetConf（RFC 6241）的edit-config操作完成的。
  - \* Plugin需要使用通过yang生成的JMX API来完成配置读写。
    - \* 内部应用一般不直接使用NetConf
  - \* 管理Plugin之间的依赖关系。
- \* 推荐使用Config Subsystem代替OSGI Bundle Activator。
  - \* 如果OSGI Bundle Activator的话，可能造成在初始化未完成之前启动MD-SAL Plugin的情况。
- \* 向Plugin提供读取、写入配置信息的JMX API。
  - \* 通过Plugin的yang module定义生成关联配置文件的读写API。

# 配置Configuration Module的pom.xml

- \* 在sdnap/rpcprovider/configuration下创建configuration的pom.xml

```
[root@odl configuration]# pwd  
/root/sdnapp/rpcprovider/configuration  
[root@odl configuration]# vim pom.xml []
```

- \* 配置configuration的pom.xml

```
<modelVersion>4.0.0</modelVersion>  
  
<parent>  
  <groupId>org.opendaylight.sdnapp</groupId>  
  <artifactId>rpcprovider-top</artifactId>  
  <version>1.0.0-Helium-SR2</version>  
  <relativePath>..</relativePath>  
</parent>  
  
<groupId>org.opendaylight.sdnapp</groupId>  
<artifactId>configuration</artifactId>  
<version>1.0.0-Helium-SR2</version>  
<name>${project.artifactId}</name>  
  
<packaging>jar</packaging>
```

继承rpcprovider plugin的最上层pom.xml。  
<relativePath>指定其相对路径为上一层目录。

设定当前的configuration module的信息。

指定configuration以jar形式打包。

# 配置Configuration Module的pom.xml

- \* 配置build-helper-maven-plugin编译信息。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>build-helper-maven-plugin</artifactId>
      <version>${mojo.build.helper.version}</version>
    </plugin>
    <executions>
      <execution>
        <id>attach-artifacts</id>
        <goals>
          <goal>attach-artifact</goal>
        </goals>
      </execution>
    <executions>
      <phase>package</phase>
```

指定使用build-helper-maven-plugin

指定执行目的为attach-artifact。  
(打包时生成配置文件)

指定执行的阶段为package。



编译信息的配置还没有结束，下一页slide继续。

# 配置Configuration Module的pom.xml

```
<configuration>
  <artifacts>
    <artifact>
      <file>
        ${configFilePath}
      </file>
      <type>xml</type>
      <classifier>config</classifier>
    </artifact>
  </artifacts>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

指定配置文件的输出路径。  
configFilePath的定义在rpcprovider plugin最上层pom.xml中。

指定配置文件类型和识别名。

# 在Configuration Module中创建配置文件

- \* 在configuration module下创建配置文件目录。
  - \* src/main/resources/initial/

```
[root@odl configuration]# pwd  
/root/sdnapp/rpcprovider/configuration  
[root@odl configuration]# cd src/main/resources/  
[root@odl resources]# pwd  
/root/sdnapp/rpcprovider/configuration/src/main/resources  
[root@odl resources]# 
```

- \* 在src/main/resources/initial目录下创建配置文件。
  - \* 配置文件名设为：801-rpcprovider-config.xml
  - \* 通过文件前面的数字可以指定Plugin初始化时的配置文件的读取顺序。

```
[root@odl initial]# pwd  
/root/sdnapp/rpcprovider/configuration/src/main/resources/initial  
[root@odl initial]# ls  
801-rpcprovider-config.xml  
[root@odl initial]# 
```

# 设定配置文件

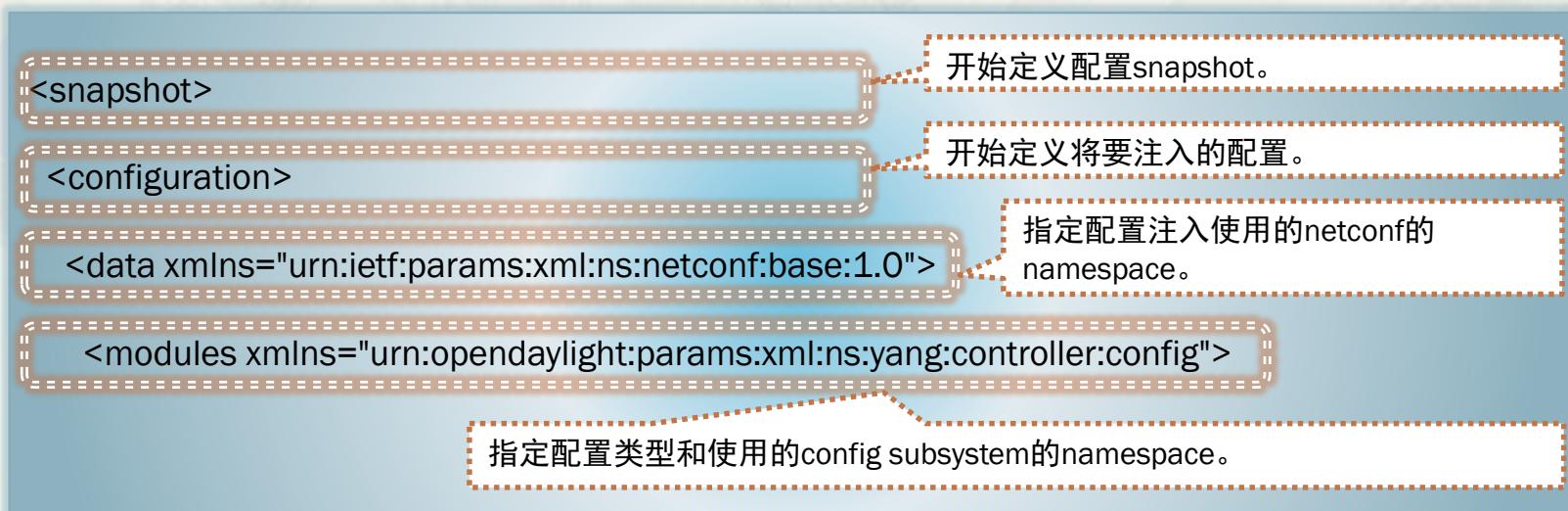
## 801-rpcprovider-config.xml

- \* 打开801-rpcprovider-config.xml

```
[root@odl initial]# pwd  
/root/sdnapp/rpcprovider/configuration/src/main/resources/initial  
[root@odl initial]# vim 801-rpcprovider-config.xml
```

- \* 定义801-rpcprovider-config.xml

- \* <configuration>下的配置在Plugin启动初始化时都会被做为配置数据、实例等注入到MD-SAL中。以方便Plugin内部使用，或者向外部使用者提供配置读写JMX API和RestConf API。
- \* 配置文件和定义Plugin的yang module需要关联。定义Plugin的yang module放在implementation module下。关于详细的关联将在implementation module的部分进行讲解。



# 设定配置文件

## 801-rpcprovider-config.xml

```
<module>
```

配置启动时做为module注入的数据。

```
<type xmlns:rpcprovider="urn:opendaylight:sdnapi:rpcprovider:provider-impl">  
    rpcprovider:rpcprovider-provider-impl  
</type>  
<name>rpcprovider-provider-impl</name>
```

指定注入对象Plugin信息。

指定定义Plugin的yang module的namespace和识别字符（module名），并设定名称。这里预先指定rpcprovider的implementation的yang module的namespace和module名。Implementation的yang module定义稍后介绍。

```
<rpc-registry>
```

开始配置注入数据。<rpc-registry>对应定义Plugin的yang module中的container。

```
<type xmlns:binding="urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding">  
    binding:binding-rpc-registry  
</type>  
<name>binding-rpc-broker</name>
```

指定注入md-sal-binding的binding-rpc-registry对应的Java Class：  
org.opendaylight.controller.sal.binding.api.RpcProviderRegistry

# 设定配置文件

## 801-rpcprovider-config.xml

```
<contents-switch-flag>url</contents-switch-flag>  
</module>  
</modules>  
</data>  
</configuration>  
</snapshot>
```

定义前面Specification中说到的Flag (url or qq) 的配置名和初始值。  
注意这里直接指定字符串本身。不需要字符串



# 使用Notification和DataStore时的实例注入

- \* 使用MD-SAL的Notification和DataStore时需要配置的注入实例的一个例子。
  - \* 虽然本例子不使用，但在以后讲解Notification和DataStore的时候会提及，在这里做为Tips进行一下简单的说明。
- \* Notification
  - \* Namespace为： urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
  - \* Type为binding： binding-notification-service
  - \* Java Class为： org.opendaylight.controller.sal.binding.api.NotificationProviderService
- \* DataStore
  - \* Namespace为： urn:opendaylight:params:xml:ns:yang:controller:md:sal:binding
  - \* Type为binding： binding-async-data-broker
  - \* Java Class为： org.opendaylight.controller.md.sal.binding.api.DataBroker

- \* 除了上例，SAL还有提供其他的实例也可以使用，这里就不一一介绍了。

# Steps

---

- \* 配置rpcprovider的Maven pom.xml
- \* 制作Model Module
- \* 制作Configuration Module
- \* 制作Implementation Module
- \* 制作Karaf Module

# Implementation Module

---

- \* 在Implementation module中定义Plugin的yang module。
- \* 实现具体服务内容。
  - \* Java coding。
- \* 通过YangTools Plugin的yang-maven-plugin自动生成Plugin的Java bindings。
- \* 生成提供服务实现、Plugin Java bindings的OSGI Bundle。

# 配置Implementation Module的pom.xml

- \* 在sdnap/rpcprovider/implementation下创建implementation的pom.xml

```
[root@odl implementation]# pwd  
/root/sdnapp/rpcprovider/implementation  
[root@odl implementation]# vim pom.xml []
```

- \* 配置Implementation的pom.xml

```
<modelVersion>4.0.0</modelVersion>  
  
<parent>  
  <groupId>org.opendaylight.sdnapp</groupId>  
  <artifactId>rpcprovider-top</artifactId>  
  <version>1.0.0-Helium-SR2</version>  
  <relativePath>../</relativePath>  
</parent>  
  
<groupId>org.opendaylight.sdnapp</groupId>  
<artifactId>implementation</artifactId>  
<version>1.0.0-Helium-SR2</version>  
<name>${project.artifactId}</name>  
  
<packaging>bundle</packaging>
```

继承rpcprovider plugin的最上层pom.xml。  
<relativePath>指定其相对路径为上一层目录。

设定当前的implementation module的信息。

指定implementation module以osgi bundle形式编译打包。

# 配置Implementation Module的pom.xml

- \* 配置OSGI Bundle依赖。

```
<dependencies>

    <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>model</artifactId>
        <version>${rpcprovider.version}</version>
    </dependency>

    <dependency>
        <groupId>org.opendaylight.controller</groupId>
        <artifactId>sal-binding-config</artifactId>
        <version>${mdsal.version}</version>
    </dependency>

</dependencies>
```

指定model module的信息。  
服务实现需要使用model的Java bindings。

指定sal-binding-config的信息。  
服务实现需要使用sal-binding的Java bindings。

# 配置Implementation Module的pom.xml

- \* 配置yang-maven-plugin编译信息。

```
<build>
  <plugins>
    <plugin>
      <groupId>org.opendaylight.yangtools</groupId>
      <artifactId>yang-maven-plugin</artifactId>
      <version>${yangtools.version}</version>
      <dependencies>
        <dependency>
          <groupId>org.opendaylight.controller</groupId>
          <artifactId>yang-jmx-generator-plugin</artifactId>
          <version>${config.version}</version>
        </dependency>
        <dependency>
          <groupId>org.opendaylight.yangtools</groupId>
          <artifactId>maven-sal-api-gen-plugin</artifactId>
          <version>${yangtools.version}</version>
        </dependency>
      </dependencies>
```

指定使用yang-maven-plugin。

配置yang-maven-plugin编译时生成JMX API的依赖信息。

配置yang-maven-plugin编译时生成SAL API的依赖信息。



编译信息的配置还没有结束，下一页slide继续。

# 配置Implementation Module的pom.xml

```
<executions>
  <execution>

    <goals>
      <goal>generate-sources</goal>
    </goals>

    <configuration>
      <yangFilesRootDir>src/main/yang</yangFilesRootDir>
    </configuration>

    <codeGenerators>

      <generator>
        <codeGeneratorClass>
          org.opendaylight.controller.config.yangjmxgenerator.plugin.JMXGenerator
        </codeGeneratorClass>
        <outputBaseDir>${jmxGeneratorPath}</outputBaseDir>

        <additionalConfiguration>
          <namespaceToPackage1>
            urn:opendaylight:params:xml:ns:yang:controller==org.opendaylight.controller.config.yang
          </namespaceToPackage1>
        </additionalConfiguration>
      </generator>
    </codeGenerators>
  </execution>
</executions>
```

指定执行Plugin的目的为 generate-resources。  
(生成Java bindings)

指定yang module文件的目录路径。

配置JMX API代码生成器。  
<outputBaseDir>指定生成的J API的java bindings 的输出目录路径。

设定yang namespace到 java package的转换规则。



编译信息的配置还没有结束，下一页slide继续。

# 配置Implementation Module的pom.xml

```
<generator>
  <codeGeneratorClass>
    org.opendaylight.yangtools.maven.sal.api.gen.plugin.CodeGeneratorImpl
  </codeGeneratorClass>
  <outputBaseDir>${salGeneratorPath}</outputBaseDir>
</generator>
</codeGenerators>

<inspectDependencies>true</inspectDependencies>

</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

配置SAL API代码生成器。  
<outputBaseDir>指定生成的SAL API的java bindings的输出目录路径。

<inspectDependencies>  
设为true时，当前 module依赖的其他 module的yang文件也会被做为搜索对象。

# 在Implementation Module中创建Yang Model

- \* 在implementation module下创建yang model定义文件目录。
  - \* src/main/yang

```
[root@odl implementation]# pwd  
/root/sdnap/rpcprovider/implementation  
[root@odl implementation]# cd src/main/yang/  
[root@odl yang]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/yang  
[root@odl yang]# 
```

- \* 在src/main/yang目录下创建yang module定义文件。
  - \* Implementation module的定义Plugin的yang module取名为earth-rpcprovider-impl，文件名如下：

```
[root@odl yang]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/yang  
[root@odl yang]# ls  
earth-rpcprovider-impl.yang  
[root@odl yang]# 
```

# 定义Yang Module

## earth-rpcprovider-impl.yang

- \* 定义earth-rpcprovider-impl.yang

```
[root@odl yang]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/yang  
[root@odl yang]# vim earth-rpcprovider-impl.yang
```

```
module earth-rpcprovider-impl {  
    yang-version 1;  
    namespace "urn:opendaylight:sdnap:earth:rpcprovider-impl";  
    prefix earth-rpcprovider-impl;  
  
    import config {  
        prefix config;  
        revision-date 2013-04-05;  
    }  
  
    import opendaylight-md-sal-binding {  
        prefix md-sal;  
        revision-date 2013-10-28;  
    }
```

指定module名字（和文件名一致）

指定yang的版本号。  
指定module的namespace， module的namespace不能重复。  
指定被参照、引用时的prefix。

导入要使用的其他yang module。这里导入config。（config subsystem service）

导入要使用的其他yang module。这里导入md-sal-binding。（md-sal service）

# 定义Yang Module

## earth-rpcprovider-impl.yang

```
description "Definition of sdnap :: earth :: rpcprovider-impl";  
revision 2015-3-3 {  
    description "Initial revision";  
}  
  
identity earth-rpcprovider-impl {  
    base config:module-type;  
    config:java-name-prefix EarthRpcProvider;  
}
```

设定module描述和版本信息。

设定识别字符。

指定通过config subsystem service从配置的module type导出识别字符。

设定生成的Java class的前缀。

# 定义Yang Module

## earth-rpcprovider-impl.yang

```
augment "/config:modules/config:module/config:configuration" {
```

```
case earth-rpcprovider-impl {  
    when "/config:modules/config:module/config:type = 'earth-rpcprovider-impl';"
```

定义Config subsystem的yang结构树中configuration choice的选择分支。

```
        container rpc-registry {  
            uses config:service-ref {  
                refine type {  
                    mandatory false;  
                    config:required-identity md-sal:binding-rpc-registry;  
                }  
            }  
        }
```

设定module type识别字符是earth-rpcprovider-impl的情况下分支定义。

```
        leaf contents-switch-flag {  
            type string;  
        }
```

指定关联配置文件的实例注入。向Plugin注入binding-rpc-registry的java实例，并生成相应getter和setter method。

创建rpc-registry container，指定其为 configuration choice的分支。

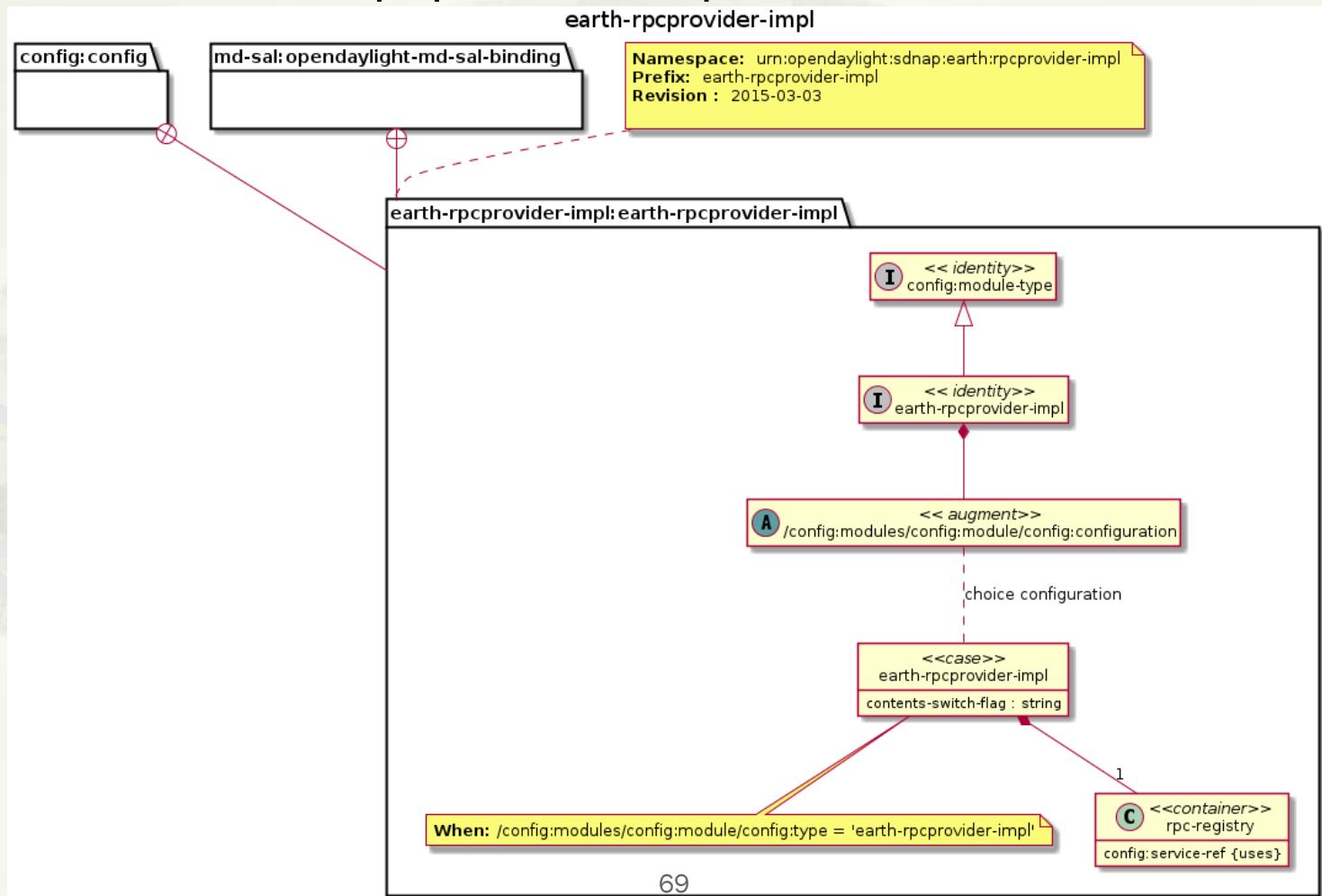
使用config的service-ref归类（包括leaf name和leaf type）。

重新设定service-ref归类的leaf type的必要性，并指定通过edit-config注入的实例的识别字符。

指定关联配置文件的配置变量注入。注入contents-switch-flag，并生成相应getter和setter method。

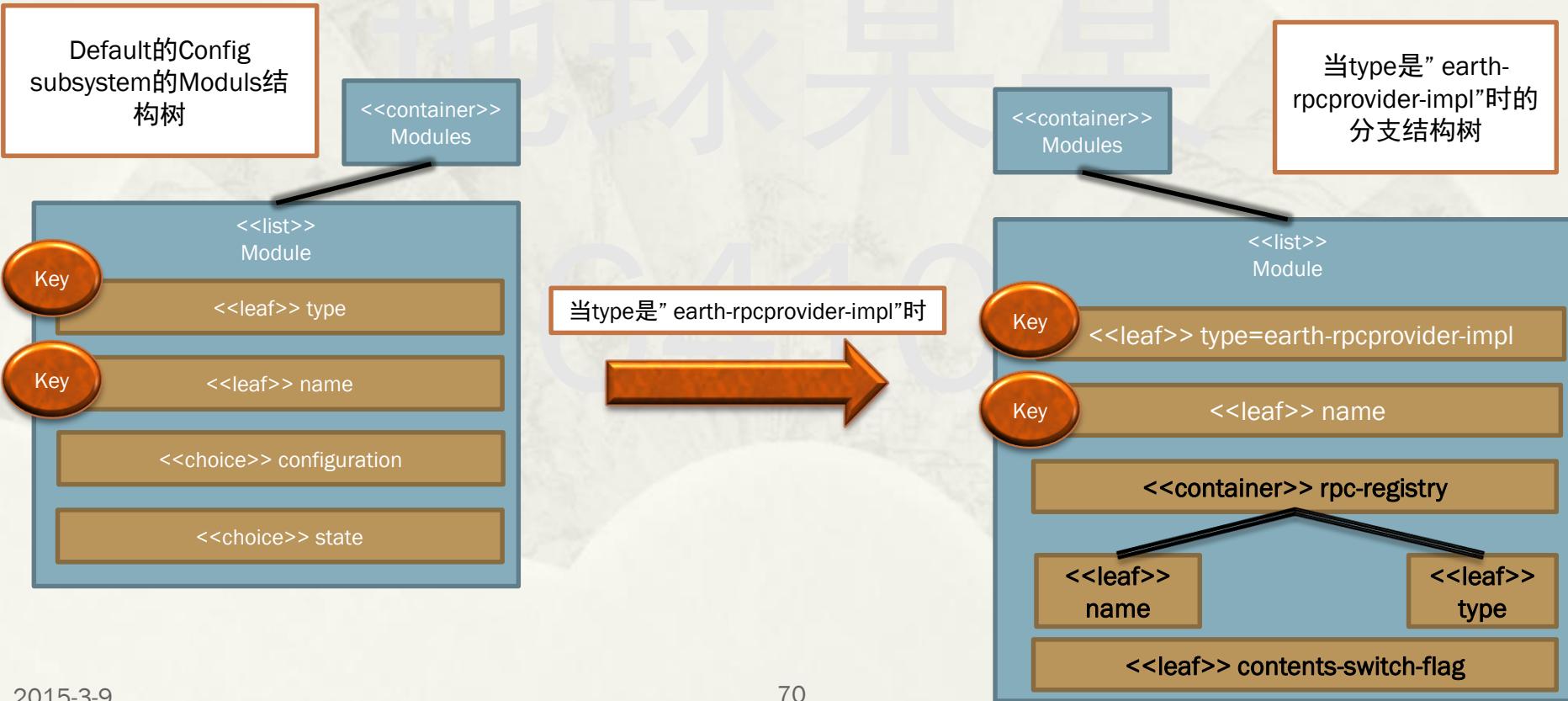
# UML Diagram

- \* 定义好的earth-rpcprovider-impl module的UML图如下：



# 定义config:choice configuration分支

- \* 前面在earth-rpcprovider-impl yang module的简介中涉及到了使用Augment statement定义choice configuration的分支yang结构树。
- \* 例子中的Augment statement所做的事如下：
  - \* 在下图中省略了一些属性、配置对象的重新定义等内容。



# 使用Notification和DataStore时Yang实例注入设定

- \* 在前面的Tips中，介绍了使用Notification和DataStore时的配置文件的实例注入设定例。
- \* 虽然在本例子中不会使用到，不过在这里做为Tips，介绍下相对应的yang设定例。

```
/*
container notification-service {
    uses config:service-ref {
        refine type {
            mandatory false;
            config:required-identity md-sal:binding-notification-service;
        }
    }
}

container data-broker {
    uses config:service-ref {
        refine type {
            mandatory false;
            config:required-identity md-sal:binding-async-data-broker;
        }
    }
}
*/
```

使用Notification时的设定。

使用DataBroker时的设定。

# earth-rpcprovider-impl Module的 SAL Java Bindings

- \* 根据Implementation module的pom.xml配置， SAL的Java bindings会被输出到以下目录。
  - \* \${project.build.directory}/generated-sources/yang-gen-sal
- \* 编译后implementation module的SAL Java bindings的基本输出目录。

```
[root@odl yang-gen-sal]# pwd  
/root/sdnap/rpcprovider/implementation/target/generated-sources/yang-gen-sal  
[root@odl yang-gen-sal]# 
```

- \* SAL Java package的输出目录
  - \* SAL Java bindings的基本输出目录 + Package目录

org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303;

```
[root@v157-7-239-39 rev150301]# pwd  
/root/sdnap/rpcprovider/model/target/generated-sources/yang-gen-sal/org/opendaylight/yang/gen/v1/urn/opendaylight/sdnap/rpcprovider/rev150301
```



- \* 在所有module的pom.xml没有完全配置好前，建议单独编译相应。

```
[root@v157-7-239-39 implementation]# pwd  
/root/sdnap/rpcprovider/implementation  
[root@v157-7-239-39 implementation]# mvn clean install -DskipTests
```

# earth-rpcprovider-impl Module的 SAL Java Bindings

- \* 生成的SAL Java bindings。

```
[root@odl rev150303]# pwd
/root/sdnapp/rpcprovider/implementation/target/generated-sources/yang-gen-sal/org
/opendaylight/yang/gen/v1/urn:opendaylight:sdnap/earth/rpcprovider/impl/rev15030
3
[root@odl rev150303]# find ./ | fgrep .java
./EarthRpcproviderImpl.java
./$YangModelBindingProvider.java
./$YangModuleInfoImpl.java
./modules/module/configuration/EarthRpcproviderImpl.java
./modules/module/configuration/earth/rpcprovider/impl/RpcRegistryBuilder.java
./modules/module/configuration/earth/rpcprovider/impl/RpcRegistry.java
./modules/module/configuration/EarthRpcproviderImplBuilder.java
[root@odl rev150303]# 
```

# earth-rpcprovider-impl Module的 SAL Java Bindings

- \* identity定义会生成相应的Abstract Class。

```
identity earth-rpcprovider-impl {  
    base config:module-type;  
    config:java-name-prefix EarthRpcProvider;  
}
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnapearth.rpcprovider.impl.rev150303.\  
EarthRpcproviderImpl
```

```
public abstract class EarthRpcproviderImpl extends ModuleType {  
    public static final QName QNAME = org.opendaylight.yangtools.yang.common.QName\  
.create("urn:opendaylight:sdnapearth:rpcprovider-impl","2015-03-03",\  
"earth-rpcprovider-impl");  
    public EarthRpcproviderImpl() {  
    }  
}
```

# earth-rpcprovider-impl Module的 SAL Java Bindings

- \* augment 定义会生成相应的Java package。
- \* case 生成相应的Builder和Interface。

```
augment "/config:modules/config:module/config:configuration" {  
    case earth-rpcprovider-impl {  
        when "/config:modules/config:module/config:type = 'earth-rpcprovider-impl'";
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.modules.\  
module.configuration.EarthRpcproviderImplBuilder
```

```
org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.modules.\  
module.configuration.EarthRpcproviderImpl
```

```
public interface EarthRpcproviderImpl extends DataObject,  
Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.modules.module.\\  
configuration.EarthRpcproviderImpl>, Configuration{  
    public static final QName QNAME = org.opendaylight.yangtools.yang.common.QName.create\\  
("urn:opendaylight:sdnap:earth:rpcprovider-impl","2015-03-03","earth-rpcprovider-impl");
```

```
    RpcRegistry getRpcRegistry();  
    java.lang.String getContentsSwitchFlag();
```



```
    container rpc-registry {  
        //...snip...  
        leaf contents-switch-flag { type string;}
```

# earth-rpcprovider-impl Module的 SAL Java Bindings

- \* 生成相应的Builder和Interface。

```
augment "/config:modules/config:module/config:configuration" {
    case earth-rpcprovider-impl {
        when "/config:modules/config:module/config:type = 'earth-rpcprovider-impl'";
        container rpc-registry {
            uses config:service-ref {
                refine type { mandatory false; config:required-identity md-sal:binding-rpc-registry;}
            }
        }
    }
}
```



org.opendaylight.yang.gen.v1.urn.opendaylight.**earth.rpcprovider.impl**.rev150303.modules.\  
module.configuration.earth.rpcprovider.impl. **RpcRegistryBuilder**

org.opendaylight.yang.gen.v1.urn.opendaylight.**earth.rpcprovider.impl**.rev150303.modules.\  
module.configuration.earth.rpcprovider.impl. **RpcRegistry**

```
public interface RpcRegistry extends ChildOf<Module>,
Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.modules.module.\\
configuration.earth.rpcprovider.impl.RpcRegistry
```

# Plugin的MXBean Java Bindings

- \* 根据Implementation model的pom.xml配置， MXBean Java bindings会被输出到以下目录。
  - \* JMX等共同处理部分在以下目录，一般不做修改。
    - \* \${project.build.directory}/generated-sources/yang-gen-config
  - \* 具体实现Plugin个性化服务、 API的Java代码在以下目录，一般需要coding。
    - \* src/main/java

# Plugin的JMX Java Bindings

- \* 编译后Plugin的JMX Java bindings的基本输出目录。

```
[root@odl yang-gen-config]# pwd  
/root/sdnap/rpcprovider/implementation/target/generated-sources/yang-gen-config  
[root@odl yang-gen-config]# 
```

- \* JMX Java package的输出目录
  - \* JMX Java bindings的基本输出目录 + Package目录

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303;
```

```
[root@odl rev150303]# pwd  
/root/sdnap/rpcprovider/implementation/target/generated-sources/yang-gen-config/  
org/opendaylight/yang/gen/v1/urn/opendaylight/sdnap/earth/rpcprovider/impl/rev15  
0303
```

# Plugin的JMX Java Bindings

- \* 生成的JMX Java bindings。

```
[root@odl rev150303]# pwd  
/root/sdnapp/rpcprovider/implementation/target/generated-sources/yang-gen-config/  
org/opendaylight.yang/gen/v1/urn/opendaylight/sdnapp/earth/rpcprovider/impl/rev15  
0303  
[root@odl rev150303]# find ./ | fgrep .java  
./AbstractEarthRpcProviderModuleFactory.java  
./AbstractEarthRpcProviderModule.java  
./EarthRpcProviderModuleMXBean.java  
[root@odl rev150303]# ]
```

# Plugin的JMX Java Bindings

- \* Identity的config定义会生成相应的：
  - \* Abstract Class Factory, Abstract Class, MXBean Interface。

```
identity earth-rpcprovider-impl {  
    base config:module-type;  
    config:java-name-prefix EarthRpcProvider;}
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303.AbstractEarthRpcProviderModuleFactory
```

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303.AbstractEarthRpcProviderModule
```

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303.EarthRpcProviderModuleMXBean
```

```
public interface EarthRpcProviderModuleMXBean {
```

```
    public java.lang.String getContentsSwitchFlag();  
    public void setContentsSwitchFlag(java.lang.String contentsSwitchFlag);  
    public javax.management.ObjectName getRpcRegistry();  
    public void setRpcRegistry(javax.management.ObjectName rpcRegistry);
```

```
}
```

```
container rpc-registry {  
//...snip...  
leaf contents-switch-flag { type string;
```



# Plugin的服务实现Java Bindings

- \* 编译后Plugin的服务实现Java Bindings的基本输出目录。

```
[root@odl java]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java  
[root@odl java]# 
```

- \* 服务实现Java package的输出目录
  - \* 服务实现Java bindings的基本输出目录 + Package目录

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303;
```

```
[root@odl rev150303]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java/org/opendaylight/yang/gen/v  
1/urn/opendaylight/sdnap/earth/rpcprovider/impl/rev150303  
[root@odl rev150303]# 
```



- \* src/main/java下的文件和目录，一经生成后，即使重新编译module也不会被删除或者更新。如果在编译后修改了yang module定义文件的关键识别字符，重新编译时可能出现编译错误。这时候需要手动删除src/main/java。

# Plugin的服务实现Java Bindings

- \* 生成的服务实现Java bindings。

```
[root@odl rev150303]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java/org/opendaylight/yang/gen/v  
1/urn/opendaylight/sdnap/earth/rpcprovider/impl/rev150303  
[root@odl rev150303]# find ./ | fgrep .java  
./EarthRpcProviderModule.java  
./EarthRpcProviderModuleFactory.java  
[root@odl rev150303]# ]
```

# Plugin的服务实现Java Bindings

- \* 生成如下Java Class。

```
identity earth-rpcprovider-impl {  
    base config:module-type;  
    config:java-name-prefix EarthRpcProvider;}
```



```
org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.\  
EarthRpcProviderModuleFactory
```

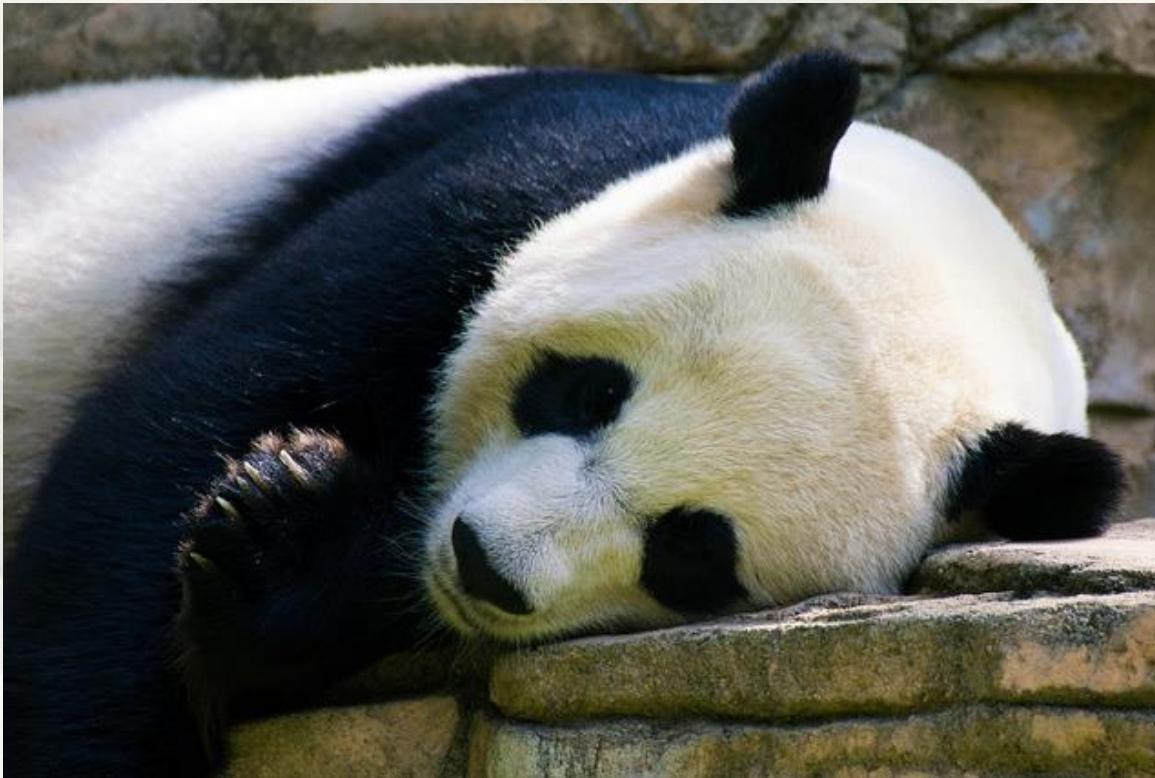
```
org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.\  
EarthRpcProviderModule
```

```
public class EarthRpcProviderModule extends \  
org.opendaylight.yang.gen.v1.urn.opendaylight.earth.rpcprovider.impl.rev150303.AbstractEarthRpcProviderModule  
{  
    //...snip...  
    @Override  
    public void customValidation() {  
    }  
    @Override  
    public java.lang.AutoCloseable createInstance() {  
        throw new java.lang.UnsupportedOperationException();  
    }  
}
```

Plugin的RPC, Notification和Datastore的实现是通过覆盖重写此method来完成的。具体如何覆盖重写会在后面具体讲解。

# Coding

- \* 终于到了写Coding的部分了。。。



图片来自于Microsoft Office 2010 PowerPoint Clip Art Search，如有版权问题请联系Microsoft。

# Coding要做的两件基本的事

- \* 在src/main/java/中创建Java Package实现具体的Plugin。
  - \* 本例子中实现具体的RPC Provider服务。

```
[root@odl java]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java  
[root@odl java]# ]
```

- \* 在Java Bindings中注册Plugin的RPC, Notification和DataStore的服务实现。
  - \* 本例子中，通过从Config Subsystem取得的org.opendaylight.controller.sal.binding.api.RpcProviderRegistry在以下Java binding中注册RPC实现。

```
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303.\  
EarthRpcProviderModule
```

```
[root@odl rev150303]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java/org/opendaylight/yang/gen/v  
1/urn/opendaylight/sdnap/earth/rpcprovider/impl/rev150303  
[root@odl rev150303]# ls | fgrep -v Factory  
EarthRpcProviderModule.java  
[root@odl rev150303]# ]
```

# EarthRpcProvider服务实现

- \* 在implementation/src/main/java/下创建Java Package目录。

```
package org.opendaylight.sdnapp;
```

```
[root@odl sdnapp]# pwd  
/root/sdnapp/rpcprovider/implementation/src/main/java/org/opendaylight/sdnapp  
[root@odl sdnapp]#
```

- \* 在implementation/src/main/java/sdn下创建EarthRpcProviderImpl.java实现EarthRpcProvider服务。

```
[root@odl sdnapp]# pwd  
/root/sdnapp/rpcprovider/implementation/src/main/java/org/opendaylight/sdnapp  
[root@odl sdnapp]# ls  
EarthRpcProviderImpl.java  
[root@odl sdnapp]#
```

# EarthRpcProviderImpl.java Code

- \* 编写EarthRpcProviderImpl.java代码，这里只对部分进行讲解。

```
[root@odl sdnalp]# pwd  
/root/sdnalp/rpcprovider/implementation/src/main/java/org/opendaylight/sdnalp  
[root@odl sdnalp]# vim EarthRpcProviderImpl.java
```

- \* 一些需要的import。

```
package org.opendaylight.sdnalp;  
import java.util.concurrent.Future;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import com.google.common.base.Preconditions;  
import com.google.common.util.concurrent.Futures;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.EarthRpcproviderInput;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.EarthRpcproviderOutputBuilder;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.EarthRpcproviderOutput;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.EarthRpcproviderService;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.KeywordType;  
import org.opendaylight.yang.gen.v1.urn.ietf.params.xml.yang.ietf.inet.types.rev100924.Uri;  
import org.opendaylight.yangtools.yang.common.RpcResult;  
import org.opendaylight.yangtools.yang.common.RpcResultBuilder;  
import org.opendaylight.yangtools.yang.common.RpcError.ErrorType;
```



下一页slide继续。

# EarthRpcProviderImpl.java Code

```
public class EarthRpcProviderImpl implements EarthRpcproviderService {
```

```
//create logger.  
private static final Logger _logger = LoggerFactory  
.getLogger(EarthRpcProviderImpl.class);
```

生成logger。

```
//properties.  
private String _contentsSwitchFlag;
```

宣言config属性。

```
private final Uri _sdnapUrl;  
private final String _sdnapQq;  
private final String _sdnapCity;  
private final Long _sdnapEmployees;
```

宣言sdnap的profile属性。

```
private final Uri _sdnlabUrl;  
private final String _sdnlabQq;  
private final String _sdnlabCity;  
private final Long _sdnlabEmployees;
```

宣言sdnlab的profile属性。

```
//default branch control values.  
private final String URL_CONTENTS_SWITCH_FLAG = new String("url");  
private final String QQ_CONTENTS_SWITCH_FLAG = new String("qq");  
private final String KEYWORD_SDNAP = new String("sdnap");  
private final String KEYWORD_SDNLAB = new String("sdnlab");
```

宣言并定义程序控制的final属性。



下一页slide继续。

# EarthRpcProviderImpl.java Code

```
//profile values, they should be stored in DB usually.  
private final String SDNAP_URL = new String("http://www.sdnnap.com/");  
private final String SDNAP_QQ = new String("279796875");  
private final String SDNAP_CITY = new String("Peking");  
private final Long SDNAP_EMPLOYEES = new Long(1);  
  
private final String SDNLAB_URL = new String("http://www.sdnlab.com/");  
private final String SDNLAB_QQ = new String("194240432");  
private final String SDNLAB_CITY = new String("Nanking");  
private final Long SDNLAB_EMPLOYEES = new Long(2);  
  
public EarthRpcProviderImpl() {  
    //set default values.  
    this._contentsSwitchFlag = URL_CONTENTS_SWITCH_FLAG;  
  
    this._sdnapUrl = new Uri(SDNAP_URL);  
    this._sdnapQq = SDNAP_QQ;  
    this._sdnapCity = SDNAP_CITY;  
    this._sdnapEmployees = SDNAP_EMPLOYEES;  
    this._sdnlabUrl = new Uri(SDNLAB_URL);  
    this._sdnlabQq = SDNLAB_QQ;  
    this._sdnlabCity = SDNLAB_CITY;  
    this._sdnlabEmployees = SDNLAB_EMPLOYEES;  
}
```

宣言并定义sdnap的profile的final属性。Profile的值一般情况下应该储存在DB中。这里使用final属性来代替DB。

宣言并定义sdnlab的profile的final属性。

在constructor中进行初始化处理。

代入属性默认值。



下一页slide继续。

# EarthRpcProviderImpl.java Code

```
//_contentsSwitchFlag setter.  
public void setContentsSwitchFlag(String contentsSwitchFlag) {  
    this._contentsSwitchFlag = contentsSwitchFlag;  
}  
  
@Override  
public Future<RpcResult<EarthRpcproviderOutput>> earthRpcprovider(EarthRpcproviderInput input) {  
    this._logger.info("EarthRpcproviderImpl is closed.", this);  
  
    ////check input, if input is null throw NullPointerException.  
    Preconditions.checkNotNull(input, "input can not be null, throw NullPointerException.");  
  
    KeywordType keywordInput = null;  
    RpcResultBuilder<EarthRpcproviderOutput> earthRpcproviderBuilder = null;  
    EarthRpcproviderOutput output = null;  
    EarthRpcproviderOutputBuilder outputBuilder = null;
```

设定config属性的setter method。

重写EarthRpcproviderService的earthRpcprovider(EarthRpcproviderInput input)

NullCheck很重要。

会利用到的变量的宣言和初始化。



下一页slide继续。

# EarthRpcProviderImpl.java Code

```
//check keyword, if keyword is null return app error.  
if ((keywordInput = input.getKeyword()) == null) {  
    earthRpcproviderBuilder = RpcResultBuilder.<EarthRpcproviderOutput>failed()  
        .withError(ErrorType.APPLICATION, "Invalid input value",  
            "Argument can not be null.");  
  
//check keyword, if keyword is invalid return app error.  
}else if (!(keywordInput.getValue().equals(KEYWORD_SDNAP)) &&  
    !(keywordInput.getValue().equals(KEYWORD_SDNLAB))){  
    earthRpcproviderBuilder = RpcResultBuilder.<EarthRpcproviderOutput>failed()  
        .withError(ErrorType.APPLICATION, "Invalid input value",  
            "only sdnap or sdnlab is acceptable as a keyword.");  
  
//set profiles into outputBuilder.  
}else {  
    outputBuilder = new EarthRpcproviderOutputBuilder();  
  
//set sdnap profile into outputBuilder.  
if (keywordInput.getValue().equals(KEYWORD_SDNAP)) {  
  
outputBuilder  
    .setUrl(this._sdnapUrl)  
    .setQq(this._sdnapQq)  
    .setCity(this._sdnapCity)  
    .setEmployees(this._sdnapEmployees);
```

得到输入的keyword并代入keywordInput。  
Check是否是Null，如果是null，创建App Error的RpcResultBuilder。

Check输入是否是"sdnap"或者"sdnlab"，如果都不是，创建App Error的RpcResultBuilder。

生成输出Builder的Instance。

当keyword是KEYWORD\_SDNAP ("sdnap") 的时候，设定以下profile。

使用sdnap的profile来创建输出。

# EarthRpcProviderImpl.java Code

```
//set sdnlab profile into outputBuilder.  
}else if (keywordInput.getValue().equals(KEYWORD_SDNLAB)) {  
  
    outputBuilder  
        .setUrl(this._sdnlabUrl)  
        .setQq(this._sdnlabQq)  
        .setCity(this._sdnlabCity)  
        .setEmployees(this._sdnlabEmployees);  
  
//set null into outputBuilder if input is mismatched.  
}else {  
  
    outputBuilder  
        .setUrl(null)  
        .setQq(null)  
        .setCity(null)  
        .setEmployees(null);  
}
```

当 keyword 是 KEYWORD\_SDNLAB ("sdnlab") 的时候，设定以下 profile。

使用 sdnap 的 profile 来创建输出。

当 keyword 是其他的时候不进行输出。



下一页 slide 继续。

# EarthRpcProviderImpl.java Code

```
//set Url as null if flag is string "qq".
if (this._contentsSwitchFlag.equals(QQ_CONTENTS_SWITCH_FLAG)) {
    outputBuilder
        .setUrl(null);

//set Qq as null if flag is not string "qq".
}else {
    outputBuilder
        .setQq(null);
}

//If output is not null, build output.
if ((output = outputBuilder.build()) != null) {
    earthRpcproviderBuilder = RpcResultBuilder.success(output);

//if null throw app error.
}else {
    earthRpcproviderBuilder = RpcResultBuilder.<EarthRpcproviderOutput>failed()
        .withError(ErrorType.APPLICATION, "Invalid output value",
                  "Output is null.");
}

return Futures.immediateFuture(earthRpcproviderBuilder.build());
}
```

当config属性是QQ\_CONTENTS\_SWITCH\_FLAG ("qq") 的时候，不输出Url。

当config属性是非QQ\_CONTENTS\_SWITCH\_FLAG ("qq") 的时候，不输出Qq。

创建output。  
如果output不是null时，创建success的RpcResultBuilder。

如果output是null时，  
创建App Error的  
RpcResultBuilder。

创建立即执行并返回  
结果的Guava Future。

# 注册RPC服务

- \* 在EarthRpcProviderModule.java中注册RPC实现。

```
[root@odl rev150303]# pwd  
/root/sdnap/rpcprovider/implementation/src/main/java/org/opendaylight/yang/gen/v  
1/urn/opendaylight/sdnap/earth/rpcprovider/impl/rev150303  
[root@odl rev150303]# vim EarthRpcProviderModule.java
```

- \* 添加将要使用的import

```
package org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303;  
import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.RpcRegistration;  
import org.opendaylight.sdnap.EarthRpcProviderImpl;  
import org.opendaylight.yang.gen.v1.urn.opendaylight.rpcprovider.rev150301.EarthRpcproviderService;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;
```

# EarthRpcProviderModule.java Code

- \* 粗字部分是手动添加内容。
- \* 其他部分是自动生成的代码。

```
public class EarthRpcProviderModule extends \\`
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303\\`
.AbstractEarthRpcProviderModule {  

\`  

//create logger.  

private final static Logger _logger = LoggerFactory.getLogger(EarthRpcProviderModule.class);  

\`  

public EarthRpcProviderModule(org.opendaylight.controller.config.api.ModuleIdentifier identifier,  

org.opendaylight.controller.config.api.DependencyResolver dependencyResolver) {  

super(identifier, dependencyResolver);  

}  

public EarthRpcProviderModule(org.opendaylight.controller.config.api.ModuleIdentifier identifier,\\`
org.opendaylight.controller.config.api.DependencyResolver dependencyResolver,\\`
org.opendaylight.yang.gen.v1.urn.opendaylight.sdnap.earth.rpcprovider.impl.rev150303
\`.EarthRpcProviderModule oldModule, java.lang.AutoCloseable oldInstance) {  

super(identifier, dependencyResolver, oldModule, oldInstance);  

}  

@Override  

public void customValidation() {  

// add custom validation form module attributes here.  

}
```

生成logger。

# EarthRpcProviderModule.java Code

```
@Override  
public java.lang.AutoCloseable createInstance() {  
    _logger.info("EarthRpcProvider is called.");  
  
    EarthRpcProviderImpl earthRpcProviderImpl = new EarthRpcProviderImpl();  
    earthRpcProviderImpl.setContentsSwitchFlag(getContentsSwitchFlag());  
  
    final RpcRegistration<EarthRpcproviderService> earthRpcproviderService =  
    getRpcRegistryDependency().addRpclImplementation(EarthRpcproviderService.class, earthRpcProviderImpl);  
  
    final class CloseResources implements AutoCloseable {  
        @Override  
        public void close() throws Exception {  
            earthRpcproviderService.close();  
            _logger.info("EarthRpcProvider was closed.", this);  
        }  
    }  
  
    AutoCloseable ret = new CloseResources();  
    return ret;  
}
```

生成 EarthRpcProviderImpl 服务的instance。

通过JMX Wrapper API (getContentsSwitchFlag) 从 config subsystem 取得 ContentsSwitchFlag 的配置变量值。并通过 earthRpcProviderImpl 实例的 setter method (setContentsSwitchFlag) 把配置变量值设给实例的 \_contentsSwitchFlag.

通过 Config subsystem 取得 RpcProviderRegistry 的 getRpcRegistryDependency()。生成 model module 的 EarthRpcproviderService Java bindings 的实例。通过 addRpclImplementation() 注册 Plugin 的 RPC 服务实现到 earthRpcproviderService 实例中。

服务关闭时的 logging 等的终了处理。

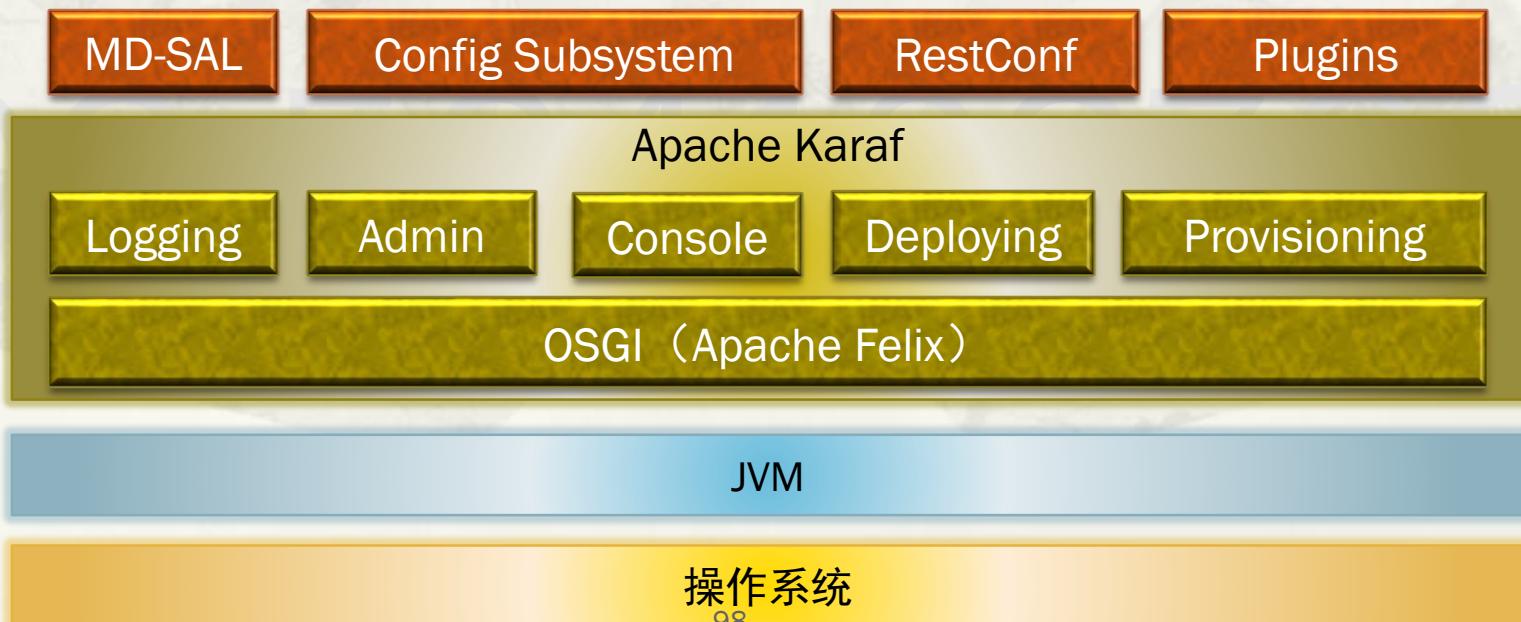
# Steps

---

- \* 配置rpcprovider的Maven pom.xml
- \* 制作Model Module
- \* 制作Configuration Module
- \* 制作Implementation Module
- \* 制作Karaf Module

# Karaf Module

- \* Karaf module会把Plugin制作成Karaf Feature，然后打包成可以导入到Apache karaf的kar文件。
  - \* Apache Karaf
    - \* 基于OSGI的运行环境，做为OSGI应用的管理容器提供各种管理utility。
  - \* Karaf Feature
    - \* 由多个Java运行文件组成的Karaf的功能管理单位。
- \* ODL Helium中，所有应用包括MD-SAL都由Apache Karaf统一管理。



# 配置Karaf Module的pom.xml

- \* 在sdnap/rpcprovider/karaf下创建karaf的pom.xml

```
[root@odl karaf]# pwd  
/root/sdnanp/rpcprovider/karaf  
[root@odl karaf]# ls  
pom.xml
```

- \* 配置Implementation的pom.xml

```
<modelVersion>4.0.0</modelVersion>  
  
<parent>  
  <groupId>org.opendaylight.sdnanp</groupId>  
  <artifactId>rpcprovider-top</artifactId>  
  <version>1.0.0-Helium-SR2</version>  
  <relativePath>..</relativePath>  
</parent>  
  
<groupId>org.opendaylight.sdnanp</groupId>  
<artifactId>karaf</artifactId>  
<version>1.0.0-Helium-SR2</version>  
<name>${project.artifactId}</name>  
  
<packaging>kar</packaging>
```

继承rpcprovider plugin的最上层pom.xml。  
<relativePath>指定其相对路径为上一层目录。

设定当前的karaf module的信息。

指定以kar形式打包。

# 配置Karaf Module的pom.xml

- \* 配置编译信息。
  - \* 把各个module生成的制品打包，制作Kar文件。

```
<build>
  <resources>

    <resource>
      <filtering>true</filtering>
      <directory>src/main/features</directory>
    </resource>

  </resources>
  <plugins>
    <plugin>

      <groupId>org.apache.karaf.tooling</groupId>
      <artifactId>karaf-maven-plugin</artifactId>
      <version>${karaf.version}</version>
      <extensions>true</extensions>

    </plugin>
  </plugins>
</build>
```

把编译时的动态置换文本功能设为有效。  
(把\${}形式的文本变量置换成它的值)

指定karaf-maven-plugin的信息。  
kar文件生成需要使用karaf-maven-plugin来完成。



编译信息的配置还没有结束，下一页slide继续。

# 配置Karaf Module的pom.xml

```
<executions>
  <execution>
    <id>features-create-kar</id>
    <goals><goal>features-create-kar</goal></goals>
    <configuration>
      <featuresFile>
        ${featureFilePath}
      </featuresFile>
    </configuration>
  </execution>
</executions>
</plugin>
```

指定目标位生成kar文件。

指定feature文件的路径。karaf-maven-plugin会从feature配置文件中读取feature信息来制作kar文件。



编译信息的配置还没有结束，下一页slide继续。

# 配置Implementation Module的pom.xml

```
<plugin>  
    <groupId>org.apache.maven.plugins</groupId>  
    <artifactId>maven-resources-plugin</artifactId>  
    <version>${maven.resources.version}</version>  
  
    <executions>  
        <execution>  
            <id>filter</id>  
            <goals><goal>resources</goal></goals>  
            <phase>generate-resources</phase>  
        </execution>  
    </executions>  
</plugin>
```

指定maven-resources-plugin信息。  
文本的动态置换需要由maven-resources-plugin来完成。

在资源生成时动态置换文本，生成资源。



编译信息的配置还没有结束，下一页slide继续。

# 配置Implementation Module的pom.xml

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>${mojo.build.helper.version}</version>

    <executions>
        <execution>
            <id>attach-artifacts</id>
            <goals><goal>attach-artifact</goal></goals>
            <phase>package</phase>
        </execution>
    </executions>

```

指定maven-resources-plugin信息。  
打包时注入追加制品要由maven-resources-plugin 完成。

设定目标为attach-artifact，执行阶段  
为package。  
(打包时生成feature文件和kar文件。)



编译信息的配置还没有结束，下一页slide继续。

# 配置Implementation Module的pom.xml

```
<configuration>
  <artifacts>
    <artifact>
      <file>${featureFilePath}</file>
      <type>xml</type>
      <classifier>features</classifier>
    </artifact>

    <artifact>
      <file>
        ${karFilePath}
      </file>
      <type>kar</type>
      <classifier>features</classifier>
    </artifact>
  </artifacts>
</configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

制定生成制品为\${featureFilePath}，类型为xml。

制定生成制品为\${karFilePath}，类型为kar。

# 配置Implementation Module的pom.xml

- \* 配置依赖信息。

```
<dependencies>
    <dependency>
        <groupId>org.opendaylight.sdnapp</groupId>
        <artifactId>model</artifactId>
        <version>${project.version}</version>
    </dependency>
```

指定model依赖信息。  
生成kar文件时会使用。

```
    <dependency>
        <groupId>org.opendaylight.sdnapp</groupId>
        <artifactId>implementation</artifactId>
        <version>${project.version}</version>
    </dependency>
```

指定implementation依赖信息。  
生成kar文件时会使用。

```
    <dependency>
        <groupId>org.opendaylight.sdnapp</groupId>
        <artifactId>configuration</artifactId>
        <version>${project.version}</version>
        <classifier>config</classifier>
        <type>xml</type>
    </dependency>
</dependencies>
</project>
```

指定configuration依赖信息。  
生成kar文件时会使用。

# 制作Feature配置文件

- \* 在karaf/src/main/features下创建features.xml配置文件

```
[root@odl features]# pwd  
/root/sdnap/rpcprovider/karaf/src/main/features  
[root@odl features]# ls  
features.xml
```

- \* 制作features.xml配置文件

```
[root@odl features]# pwd  
/root/sdnap/rpcprovider/karaf/src/main/features  
[root@odl features]# vim features.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>  
  
<features name="${project.artifactId}-${project.version}"  
  xmlns="http://karaf.apache.org/xmlns/features/v1.2.0"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.2.0  
  http://karaf.apache.org/xmlns/features/v1.2.0">
```

指定namespace。指定repository名为"\${project.artifactId}-\${project.version}"

# 制作Feature配置文件features.xml

```
<repository>  
    mvn:org.opendaylight.yangtools/features-yangtools/${yangtools.version}/xml/features  
</repository>  
<repository>  
    mvn:org.opendaylight.controller/features-mdsal/${mdsal.version}/xml/features  
</repository>  
<repository>  
    mvn:org.opendaylight.controller/features-restconf/${mdsal.version}/xml/features  
</repository>
```

指定其他Feature的配置文件的repository

```
<feature name="earth-rpcprovider-model" description="EARTH :: RPCPROVIDER :: MODEL"  
version="${project.version}">
```

定义"earth-rpcprovider-model" feature。描述和版本等信息会可以在karaf中通过命令显示。

```
<feature version="${yangtools.version}">odl-yangtools-models</feature>
```

指定导入bundle前需要提前启动的feature。这里为odl-yangtools-models（解决feature依赖）

```
<bundle>mvn:org.opendaylight.sdnap/model/${project.version}</bundle>  
</feature>
```

导入model生成的bundle

# 制作Feature配置文件features.xml

```
<feature name="earth-rpcprovider-impl" description="EARTH :: RPCPROVIDER :: IMPL" version="${project.version}">
    <feature version="${project.version}">earth-rpcprovider-model</feature>
    <feature version="${mdsal.version}">odl-restconf</feature>
    <bundle>mvn:org.opendaylight.sdnapi/implementation/${project.version}</bundle>
    <configfile finalname="etc/opendaylight/karaf/801-rpcprovider-config.xml">
        mvn:org.opendaylight.sdnapi/configuration/${project.version}/xml/config
    </configfile>
</feature>

<feature name="earth-rpcprovider-apidocs" description="EARTH :: RPCPROVIDER :: APIDOCS" version="${project.version}">
    <feature version="${project.version}">earth-rpcprovider-impl</feature>
    <feature version="${mdsal.version}">odl-mdsal-apidocs</feature>
</feature>
</features>
```

指定导入bundle前需要启动odl-mdsal-broker, earth-rpcprovider-model, odl-restconf。

定义"earth-rpcprovider-impl" feature。

指定plugin配置文件。用来在Feature启动时，注入配置文件信息。

指定启动"earth-rpcprovider-impl"feature后启动odl-mdsal-apidocs。

定义"earth-rpcprovider-apidocs" feature。

# Steps

---

- \* Plugin整体编译
- \* 把Plugin导入到karaf容器

# 编译Plugin制作Kar文件

- \* 在sdnap/rpcprovider/下编译rpcprovider Plugin。

```
[root@odl rpcprovider]# pwd  
/root/sdnapp/rpcprovider  
[root@odl rpcprovider]# ls  
configuration implementation karaf model pom.xml  
[root@odl rpcprovider]# mvn clean install -DskipTests
```

- \* 编译成功后显示类似如下信息。

```
[INFO] -----  
[INFO] Reactor Summary:  
[INFO]  
[INFO] rpcprovider-top ..... SUCCESS [ 0.191 s]  
[INFO] model ..... SUCCESS [ 8.824 s]  
[INFO] configuration ..... SUCCESS [ 0.547 s]  
[INFO] implementation ..... SUCCESS [ 2.848 s]  
[INFO] karaf ..... SUCCESS [ 1.068 s]  
[INFO] -----  
[INFO] BUILD SUCCESS  
[INFO] -----
```

- \* 编译后生成的kar文件在以下路径。

```
[root@odl target]# pwd  
/root/sdnapp/rpcprovider/karaf/target  
[root@odl target]# ls | fgrep .kar  
karaf-1.0.0-Helium-SR2.kar
```

# Steps

---

- \* Plugin整体编译
- \* 把Plugin导入到karaf容器

# 启动Karaf

- \* 下载ODL Helium

- \* <http://www.opendaylight.org/software/downloads>

- \* 启动Karaf

```
[root@odl bin]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/bin  
[root@odl bin]# ./karaf
```

- \* 成功启动后出现如下ODL的karaf console。

```
opendaylight-user@root>
```

 \* Karaf的后台运行方法。

- \* 在bin/目录执行./start可以后台运行karaf。
- \* 在bin/目录执行./client可以attach后台运行的karaf的console。
- \* 在bin/目录执行./stop可以终止后台运行的karaf。

# 倒入rpcprovider到Karaf

- \* 在karaf console中使用以下kar:install命令导入rpcprovider Plugin
  - \* kar:install file:<生成的Kar文件的绝对路径>

```
opendaylight-user@root>kar:install file:/root/sdnap/rpcprovider/karaf/target/karaf-1.0.0-Helium-SR2.kar  
opendaylight-user@root>[]
```

- \* 使用feature:list -i查看安装情况。

```
opendaylight-user@root>feature:list -i[]
```

- \* 找到我们制作好的rpcprovider Plugin的三个Features。Features信息如下。

earth-rpcprovider-model	1.0.0-Helium-SR2	x	karaf-1.0.0-Helium-SR2	EARTH :: RPCPROVIDER :: MODEL
earth-rpcprovider-impl	1.0.0-Helium-SR2	x	karaf-1.0.0-Helium-SR2	EARTH :: RPCPROVIDER :: IMPL
earth-rpcprovider-apidocs	1.0.0-Helium-SR2	x	karaf-1.0.0-Helium-SR2	EARTH :: RPCPROVIDER :: APIDOCs

- \* Rpcprovider Plugin所依赖的其他Features也会被自动导入，这里就不一一说明了，请自行查看
  - \* odl-yangtools, odl-mdsal, odl-config, odl-restconf等。

# Steps

---

- \* 通过APIDocs 测试Plugin。

地球某某

QQ:564103786

# RestConf APIDocs

---

- \* RestConf Plugin
  - \* 根据yang model定义的服务自动生成Rest API。
- \* APIDocs
  - \* 提供浏览由RestConf生成的Rest API的功能，方便通过Rest API测试Plugin。
  - \* 当然你也可以选择使用curl, postman, HTTPComponents等其他工具或lib来测试。

# 测试Scenario

- \* 使用默认配置文件，使用APIDocs发送Rest API， keyword的值如下：
  - \* Keyword: “sdnap”
    - \* 取得sdnap的网站和Profile信息。
  - \* Keyword: “sdnlab”
    - \* 取得sdnlab的网站和Profile信息。
  - \* Keyword: “sdnabc”
    - \* 返回keyword是invalid时的信息。
- \* 把配置文件中的Contents-Switch-Flag的值设为qq，使用APIDocs发送Rest API， keyword的值如下：
  - \* Keyword: “sdnap”
    - \* 取得sdnap的QQ群和Profile信息。
  - \* Keyword: “sdnlab”
    - \* 取得sdnlab的QQ群和Profile信息。
  - \* Keyword: “abc”
    - \* 返回keyword是invalid时的信息。

# 使用APIDocs

- \* 打开浏览器，输入以下URL进入APIDocs页面。
  - \* `http://<你的ODL控制器的IP>:8181/apidoc/explorer/`
- \* 找到我们制作的rpcprovider Plugin的Rest API。
  - \* Plugin: earth-rpcprovider(2015-03-01)
  - \* RestAPI: /operations/earth-rpcprovider:earth-rpcprovider

**earth-rpcprovider(2015-03-01)**

POST /operations/earth-rpcprovider:earth-rpcprovider

Show/Hide | List Operations | Expand Operations | Raw

**Response Class**

Model | Model Schema

```
(earth-rpcprovider)output {  
}
```

Response Content Type application/json ▾

**Parameters**

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	<input type="text"/>		body	Model   Model Schema
				(earth-rpcprovider)input { }

Parameter content type: application/json ▾

Try it out!

# 取得sdnap网站信息

- \* 制作keyword是"sdnap"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	{"input":{"keyword":"sdnap"}}		body	Model   Model Schema (earth-rpcprovider)input { }

Parameter content type: application/json ▾

Try it out!

[Hide Response](#)

## Request URL

<http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider>

## Response Body

```
{  
    "output": {  
        "url": "http://www.sdnap.com/",  
        "city": "Peking",  
        "employees": 1  
    }  
}
```

得到sdnap的网站和profile信息。



点击[Try it out!]执行时会  
要求输入ODL用户和密码。

- \* ODL Helium的默认值  
是admin/admin

## Response Code

200

118

# 取得sdnlab网站信息

- \* 制作keyword是"sdnlab"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	<pre>{"input":{"keyword":"sdnlab"}}</pre>		body	Model <code>(earth-rpcprovider)input {</code>

Parameter content type: application/json ▾

[Try it out!](#) [Hide Response](#)

## Request URL

```
http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider
```

## Response Body

```
{  
    "output": {  
        "url": "http://www.sdnlab.com/",  
        "city": "Nanking",  
        "employees": 2  
    }  
}
```

得到sdnlab的网站和profile信息。

## Response Code

# Keyword是invalid的时候

- \* 制作keyword是"sdnabc"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	{"input":{"keyword":"sdnabc"}}		body	Model (earth-rpcprovider)input { }

Parameter content type: application/json

Try it out! Hide Response

Request URL

http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider

Response Body

```
{"errors": { "error": [ { "error-type": "application", "error-tag": "operation-failed", "error-message": "only sdnabc or sdnlab is acceptable as a keyword.", "error-info": { "severity": "error" } } ] }}
```

得到“only sdnabc or sdnlab is acceptable as a keyword.”的错误信息。

Response Code

# 手动修改配置文件

- \* 首先在Karaf Console中使用shutdown命令终止Karaf的运行。

```
opendaylight-user@root>shutdown -h now  
Confirm: halt instance root (yes/no): yes
```

- \* 修改rpcprovider Plugin的配置文件。

- \* ODL Helium的MD-SAL Plugin的配置文件储存在以下路径。

- \* etc/opendaylight/karaf

```
[root@odl karaf]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/etc/opendaylight/karaf  
[root@odl karaf]# ls  
00-netty.xml 01-mdsal.xml 10-rest-connector.xml 801-rpcprovider-config.xml
```

- \* 打开801-rpcprovider-config.xml，把<contents-switch-flag>的值改为qq。

- \* 修改前: <contents-switch-flag>url</contents-switch-flag>

- \* 修改后: <contents-switch-flag>qq</contents-switch-flag>

```
[root@odl karaf]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/etc/opendaylight/karaf  
[root@odl karaf]# vi 801-rpcprovider-config.xml □
```



- \* 配置文件的修改可以通过使用JMX Wrapper API或者他的Rest API进行修改。相应内容可能会在以后DataStore中讲解（如果还写继续写这个教程的话）。

# 手动修改配置文件

- \* 首先在Karaf Console中使用shutdown命令终止Karaf的运行。

```
opendaylight-user@root>shutdown -h now  
Confirm: halt instance root (yes/no): yes
```

- \* 修改rpcprovider Plugin的配置文件。

- \* ODL Helium SR2的MD-SAL Plugin的配置文件储存在etc/opendaylight/karaf路径。

```
[root@odl karaf]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/etc/opendaylight/karaf  
[root@odl karaf]# ls  
00-netty.xml 01-mdsal.xml 10-rest-connector.xml 801-rpcprovider-config.xml
```

- \* 打开801-rpcprovider-config.xml，把<contents-switch-flag>的值改为qq。

- \* 修改前: <contents-switch-flag>url</contents-switch-flag>
  - \* 修改后: <contents-switch-flag>qq</contents-switch-flag>

```
[root@odl karaf]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/etc/opendaylight/karaf  
[root@odl karaf]# vim 801-rpcprovider-config.xml □
```

- \* 重新运行Karaf。

```
[root@odl bin]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/bin  
[root@odl bin]# ./karaf □
```

- \* 配置文件的修改可以通过使用JMX Wrapper API或者他的Rest API进行修改。相应内容可能会在以后DataStore中讲解（如果还写继续写这个教程的话）。

# 关于Karaf

- \* Karaf的shutdown的Plugin终止处理在现阶段（2015年3月8日截至）还无法正常工作。
- \* 当Karaf非正常结束，Karaf中的Plugin出现错误等情况下，即使重新运行karaf也无法保证Plugin可以正常运行。
- \* 需要通过以下手段清除掉Karaf中关于Plugin的缓存，然后重新安装。
- \* 清除缓存：

```
[root@odl opendaylight]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/system/org/opendaylight  
[root@odl opendaylight]# rm -rf sdnap/[]
```

```
[root@odl distribution-karaf-0.2.2-Helium-SR2]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2  
[root@odl distribution-karaf-0.2.2-Helium-SR2]# rm -rf data/[]
```

```
[root@odl karaf]# pwd  
/root/mydata/distribution-karaf-0.2.2-Helium-SR2/etc/opendaylight/karaf  
[root@odl karaf]# rm -rf 801-rpcprovider-config.xml []
```

- \* 重新安装：

```
opendaylight-user@root>kar:install file:/root/sdnap/rpcprovider/karaf/target/kar  
af-1.0.0-Helium-SR2.kar  
opendaylight-user@root>[]
```

# 取得sdnap的QQ群信息

- \* 制作keyword是"sdnap"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	{"input":{"keyword":"sdnap"}}		body	Model   Model Schema (earth-rpcprovider)input { }

Parameter content type: application/json ▾

[Try it out!](#) [Hide Response](#)

Request URL

```
http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider
```

Response Body

```
{  
    "output": {  
        "qq": "279796875",  
        "city": "Peking",  
        "employees": 1  
    }  
}
```

得到sdnap的QQ群和profile信息。

Response Code

# 取得sdnlab的QQ群信息

- \* 制作keyword是"sdnlab"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter

Parameter	Value	Description	Parameter Type	Data Type
(earth-rpcprovider)input	{"input":{"keyword":"sdnlab"}}		body	Model   Model Schema (earth-rpcprovider)input { }

Parameter content type: application/json ▾

[Try it out!](#) [Hide Response](#)

Request URL

```
http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider
```

Response Body

```
{  
    "output": {  
        "qq": "194240432",  
        "city": "Nanking",  
        "employees": 2  
    }  
}
```

得到sdnlab的QQ群和profile信息。

Response Code

```
200
```

125

# Keyword是invalid的时候

- \* 制作keyword是"sdnabc"的Json Input, 并写入到Parameter中, 点击[Try it out!]

Parameter Value Description Parameter Type Data Type

(earth-rpcprovider)input	{"input":{"keyword":"sdnabc"}}		body	Model Model Schema (earth-rpcprovider)input { }
--------------------------	--------------------------------	--	------	---

Parameter content type: application/json ▾

Try it out! Hide Response

Request URL

http://157.7.239.39:8181/restconf/operations/earth-rpcprovider:earth-rpcprovider

Response Body

```
{  
  "errors": {  
    "error": [  
      {  
        "error-type": "application",  
        "error-tag": "operation-failed",  
        "error-message": "only sdnabc or sdnlab is acceptable as a keyword.",  
        "error-info": {  
          "severity": "error"  
        }  
      }  
    ]  
  }  
}
```

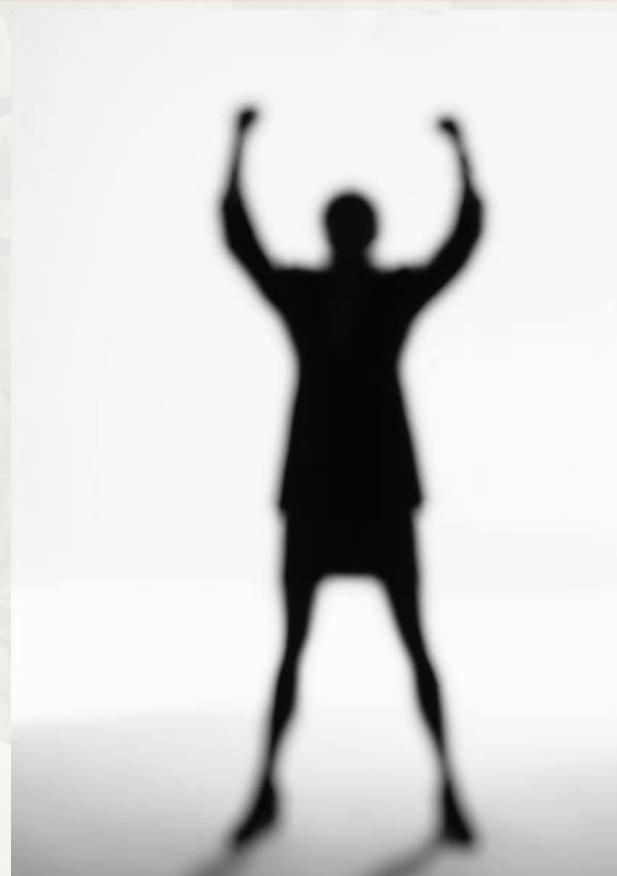
得到“only sdnabc or sdnlab is acceptable as a keyword.”的错误信息。

Response Code

500

# Mission Accomplished

看起来一切符合Spec要求并且工作正常，RPC Provider例子的讲解到此结束。



# 关于其他例子的Schedule。

- \* 第一个例子写的详细、冗长了些，其他例子即使写也不会这么详细写了。

其他例子的Schedule：未定