# 什么是SDN&Opendaylgiht

SDN（软件定义网络）的目的是使网络可编程，软件可以定义网络。手段是网络分离设备转发面与控制面，集中控制，并开放转发面和控制面之间的网络协议。

OpenDaylight项目是一个针对SDN的开源平台。使用开放协议来提供中心化可编程控制以及网络设备监控。

类似操作系统提供计算机设备的编程接口（POSIX）, ODL 提供控制和管理网络设备的接口。

* Interfaces, IP subnets and routing protocols are configured through **management plane protocols**, ranging from CLI to [NETCONF](https://blog.ipspace.net/2012/06/netconf-expect-on-steroids.html) and the latest buzzword – northbound [RESTful API](https://blog.ipspace.net/2012/08/why-is-restful-api-better-than-snmp.html);
* Router runs **control plane**routing protocols (OSPF, EIGRP, BGP …) to discover adjacent devices and the overall network topology (or reachability information in case of distance/path vector protocols);
* Router inserts the results of the control-plane protocols into [Routing Information Base (RIB) and Forwarding Information Base (FIB)](https://blog.ipspace.net/2010/09/ribs-and-fibs.html). **Data plane**software or ASICs uses FIB structures to forward the transit traffic.
* **Management plane**protocols like SNMP can be used to monitor the device operation, its performance, interface counters …

The management plane is pretty straightforward, so let’s focus on a few intricacies of the control and data planes.

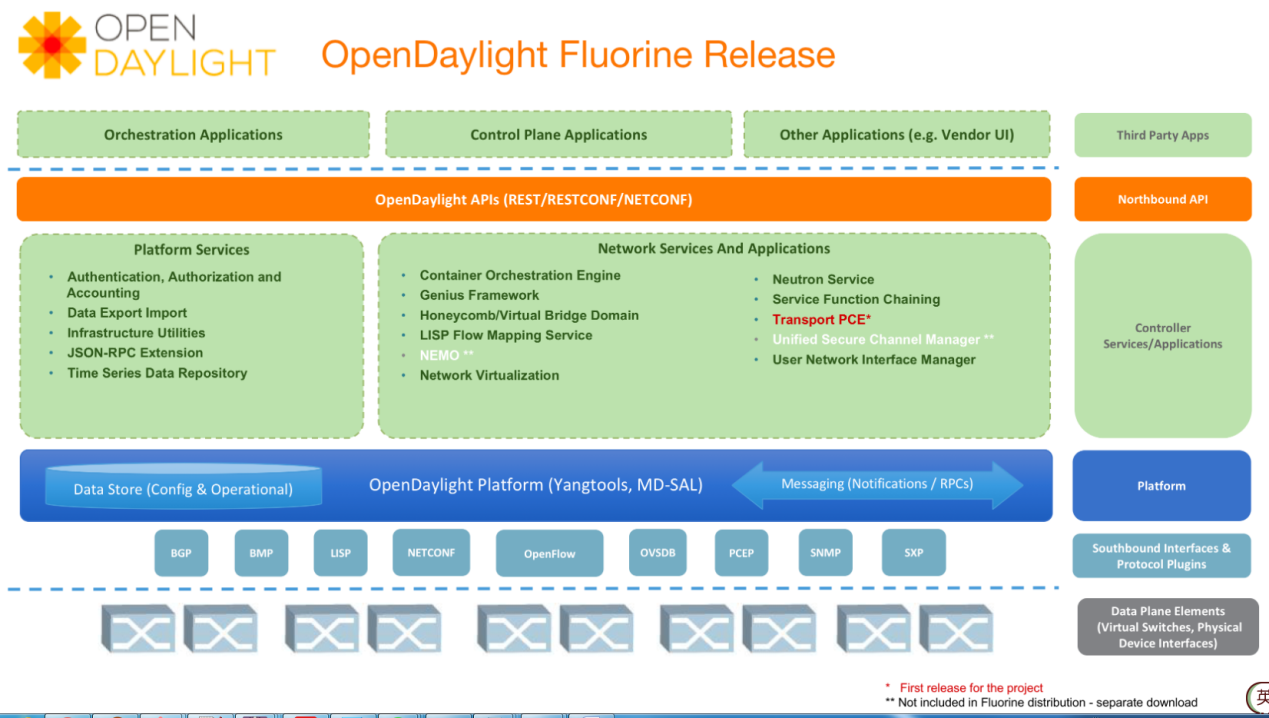
We usually have routing protocols in mind when talking about**Control plane protocols**,but in reality the control plane protocols perform numerous other functions including:

* Interface state management (PPP, LACP);
* Connectivity management (BFD, CFM);
* Adjacent device discovery (hello mechanisms present in most routing protocols, ES-IS, ARP, IPv6 ND, uPNP SSDP);
* Topology or reachability information exchange (IP/IPv6 routing protocols, IS-IS in TRILL/SPB, STP);
* Service provisioning (RSVP for IntServ or MPLS/TE, uPNP SOAP calls);

Data plane should be focused on forwarding packets but is commonly burdened by other activities:

* NAT session creation and NAT table maintenance;
* Neighbor address gleaning (example: [dynamic MAC address learning in bridging](https://blog.ipspace.net/2010/07/bridging-and-routing-is-there.html), [IPv6 SAVI](https://blog.ipspace.net/2013/03/ipv6-source-address-validation.html));
* Netflow Accounting (sFlow is cheap compared to Netflow);
* ACL logging;
* Error signaling (ICMP).

# Opendaylgiht架构



Ref: <https://www.opendaylight.org/what-we-do/current-release/fluorine>

OpenDaylight developers have formed more than 50 projects to address ways to extend network functionality.

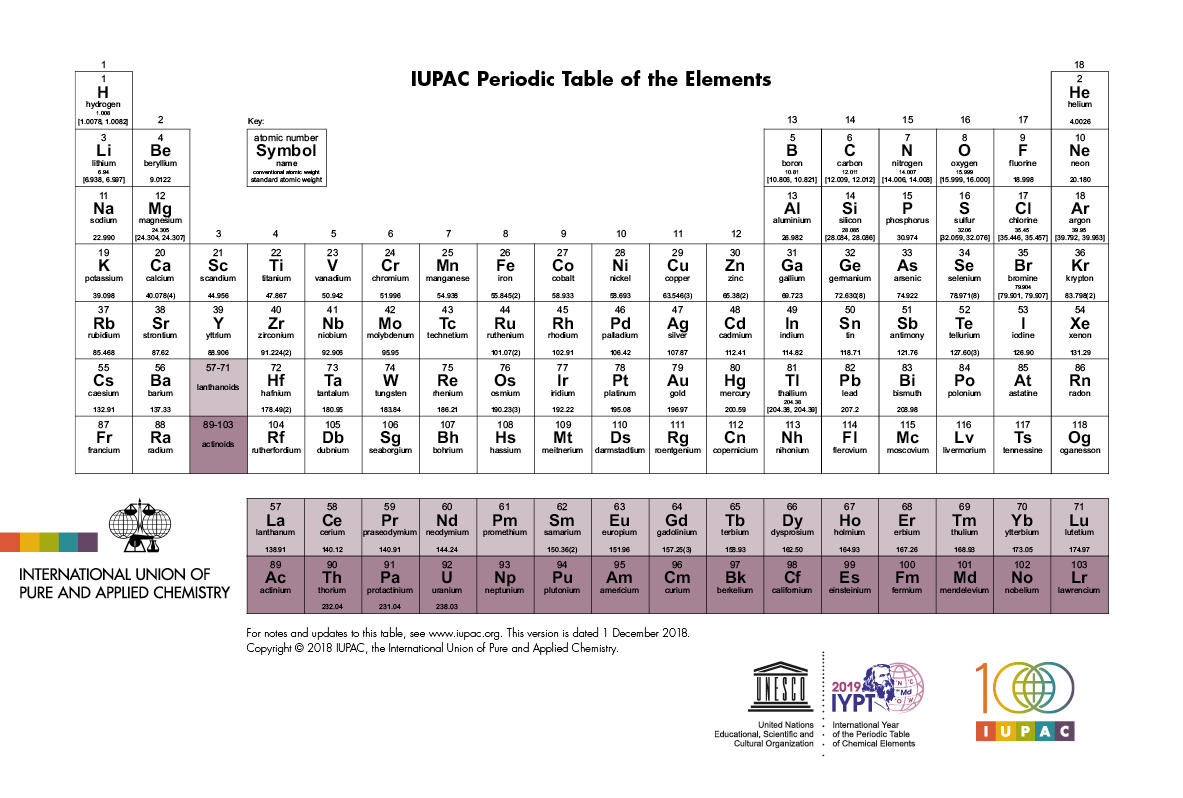
MDSAL是ODL的核心， Model-Driven Service Abstraction Layer (MD-SAL) is the OpenDaylight framework that allows developers to create new Karaf features in the form of services and protocol drivers and connects them to one another. You can think of the MD-SAL as having the following two components:

1. A shared datastore that maintains the following tree-based structures:
   1. The Config Datastore, which maintains a representation of the **desired network state.**
   2. The Operational Datastore, which is a representation of the **actual network state** based on data from the managed network elements.
2. A message bus that provides a way for the various services and protocol drivers to notify and communicate with one another.

Ref: <https://docs.opendaylight.org/en/stable-fluorine/getting-started-guide/concepts_and_tools.html>

## Opendaylight版本演绎

根据化学周期表来分配版本号，通常半年左右演进一个版本。



**配置及依赖注入系统**

Boron之前的版本主要使用YANG模型配合xml来实现**配置的管理和依赖注入**。之后主要是用OSGi的blueprint来达到该目的。

**版本号**

SR为 Service Release的缩写，在积累一堆修改后形成的发布包，类似操作系统发布的sp（service apck）。 数字越大越稳定。通常几个月便会发布一个SR版本。在Carbon版前之前的版本号会携带数字和元素名。后续仅使用数字表达。

如1.5.2-Carbon-SR2, 1.8.2

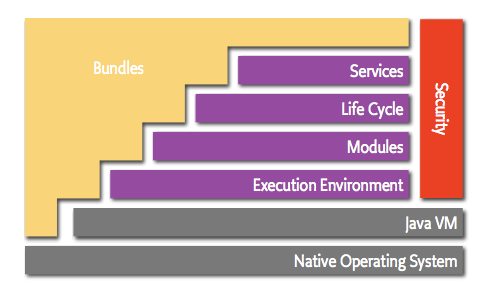
https://github.com/opendaylight/mdsalOpendaylight&Karaf&OSGi

# OSGi

OSGi的名称来源于其开源组织的名称Open Services Gateway initiative。OSGi 联盟成立于 1999 年，是一个非盈利的国际组织，旨在建立一个开放的服务规范，为通过网络向设备提供服务建立开放的标准，是开放业务网关的发起者。**起初 OSGi 技术只是关注于嵌入式领域，诸如机顶盒、服务网关、手机等应用环境**，2003 年，BMW 就基于 OSGi V3.0 规范构建其智能汽车影音系统。随着时间的发展，OSGi 的诸多优秀特性，如动态性，模块性，可扩展性，被更多的开发者所逐渐认识和欣赏，从而应用到桌面程序的设计和开发中来。Eclipse 从 V3.0 M5 版本开始，设计了 Equinox 内核，使用 OSGi 技术帮助其进行类载入，大大提升了启动速度，实际上，Equinox 就是 OSGi Framework 的一个实现。与此同时，OSGi 技术的影响也延伸到了 Java 社区，JSR-232 说明 OSGi 技术已经初开始被 Java ME 家族所认知，而 JSR-291 更是将 OSGi 技术的触角扩展到了 Java SE 和 Java EE 的范畴。OSGi 联盟的成员数量已经从最开始的几个增长到现在的 100 多个，很多世界著名的 IT 公司都加入到了这个阵营中来，如 IBM，Oracle，SAP，Red Hat，SpringSource 等等，它们的很多产品都宣布支持或使用了 OSGi 技术，如 WebSphere，Weblogic，JBOSS，Spring DM 等等，这从一个侧面说明了 OSGi 技术在企业级市场大有可为。因此，OSGi 的范畴已经不是其原来的字面意义 (Open Service Gateway initiative) 所能涵盖，**如今，OSGi 是一个企业级，动态的模块化系统规范（specification）。**

      目前基于OSGi的框架大概有4个：

* Knopflerfish,
* Apache Felix,
* Equinox,
* Spring DM。



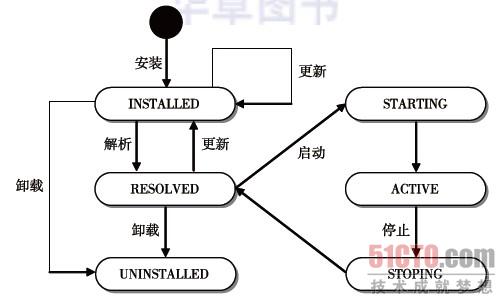
* Bundles – Bundles are the OSGi components made by the developers.
* Services – The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
* Life-Cycle – The API to install, start, stop, update, and uninstall bundles.
* Modules – The layer that defines how a bundle can import and export code.
* Security – The layer that handles the security aspects.
* Execution Environment – Defines what methods and classes are available in a specific platform.

There are four open source implementations of the framework (Apache Felix, Eclipse Concierge, Eclipse Equinox, and Knopflerfish)

## 什么是Module?

是否能和jar包画等号？Modules are a new way of grouping code and data. Contrary to [Jar files](https://en.wikipedia.org/wiki/JAR_(file_format)), modules explicitly declare which modules they depend on, and what packages they export.

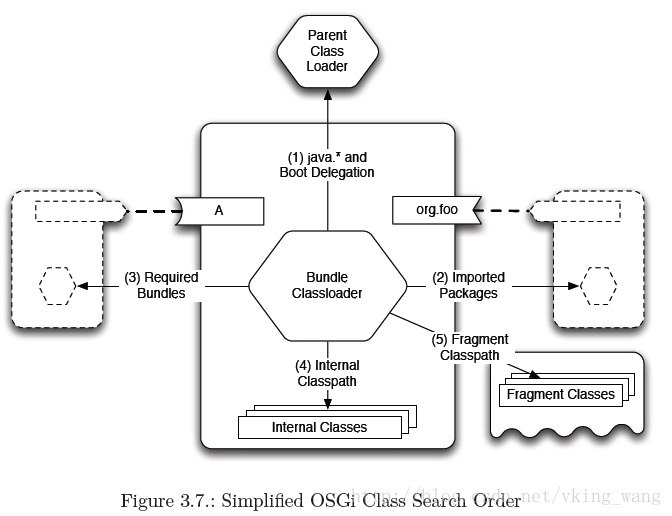
## 什么是动态？



1. INSTALLED，已安装状态。Bundle处于已安装状态就意味着它已经通过OSGi框架的有效性校验并产生了Bundle ID，但这时还未对它定义的依赖关系进行解析处理。
2. RESOLVED，已解析状态。Bundle处于已解析状态说明OSGi框架已经根据元数据信息中描述的依赖关系成功地在类名空间中找到它所有的依赖包，这时它导出的Package就可以被其他Bundle导入使用。
3. STARTING，启动中状态。Bundle处于启动中状态说明它的BundleActivator的start()方法已经被调用，但是还没执行结束。如果start()方法正常执行结束，Bundle将自动转换到ACTIVE状态； 否则，如果start()方法抛出了异常，Bundle将退回到RESOLVED状态。
4. STOPPING，停止中状态。Bundle处于停止中状态说明它的BundleActivator的stop()方法已经被调用，但是还没执行结束。无论stop()是正常结束还是抛出了异常，在这个方法退出之后，Bundle的状态都将转为RESOLVED。
5. ACTIVE，Bundle处于激活状态，说明BundleActivator的start()方法已经执行完毕，如果没有其他动作，Bundle将继续维持ACTIVE状态。

Graceperiod

## 类加载机制



## OSGi规范的演进

* OSGi Release 1 (R1)：2000年5月发布。
* OSGi Release 2 (R2)：2001年10月发布。
* OSGi Release 3 (R3)：2003年3月发布
* OSGi Release 4(R4)

1. OSGi R4.1 2007年5月发布
2. OSGi R4.2 2009年9月发布

企业级服务在这个版本中首次出现，例如远程服务规范和Blueprint容器（提供依赖注入和反转控制的容器，类似于Spring的功能）规范。JNDI、JPA和JDBC等服务规范，将Java EE的服务引入到OSGi容器中来

Web Applications规范

1. OSGi R4.3 2011年4月和2012年3月
   1. 支持新版本的JDK
   2. Bundle Wiring API子规范，该规范引入了Capabilities和Requirements的概念。在此之前，OSGi中的依赖单元要么是某个Package，要么是整个Bundle（分别使用Import-Package和Require-Bundle标识来描述），这种粒度的依赖单元能够满足代码上的依赖需要，却无法描述某些非代码的依赖特性，例如说明一个功能要依赖某个Java虚拟机版本、依赖某种架构的操作系统、依赖某些资源文件或依赖一定数量的CPU或内存等。以前虽然可以使用Bundle-Required Execution Environment标识来描述部分执行环境的特性（如指明JDK版本是Java SE 6），但是有一些特性不是在执行环境中天然存在的，是由某个Bundle安装后带来的，有了Require-Capability和Provide-Capability标识之后，才可以精确描述这类依赖关系。
2. OSGi Release 5 (R5): June 2012
   1. Core and Enterprise: June 2012
3. OSGi Release 6 (R6): June 2015
   1. Core: June 2015
4. OSGi Release 7 (R7): April 2018
   1. Core and Compendium: April 2018

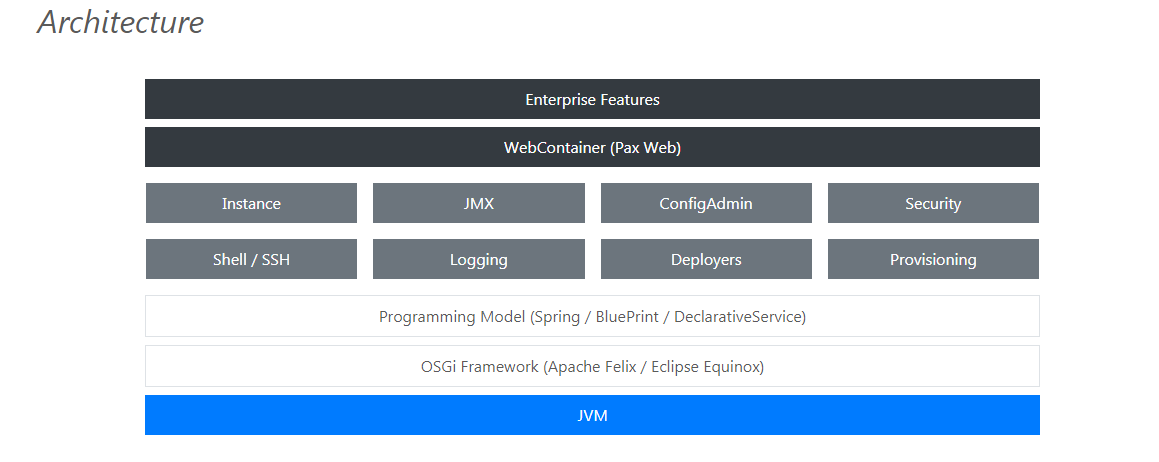
### Related RFCs and Java specifications

* [RFC 2608](https://tools.ietf.org/html/rfc2608) ([Service Location Protocol](https://en.wikipedia.org/wiki/Service_Location_Protocol))
* Sun [Jini](https://en.wikipedia.org/wiki/Jini" \o "Jini)
* Sun JCP [JSR-8](http://www.jcp.org/en/jsr/detail?id=8) (Open Services Gateway Specification)
* Sun JCP [JSR-232](http://www.jcp.org/en/jsr/detail?id=232) (Mobile Operational Management)
* Sun JCP [JSR-246](http://www.jcp.org/en/jsr/detail?id=246) (Device Management API)
* Sun JCP [JSR-249](http://www.jcp.org/en/jsr/detail?id=249) (Mobile Service Architecture for CDC)
* **Sun JCP**[**JSR-277**](http://www.jcp.org/en/jsr/detail?id=277)**(Java Module System)**
* **Sun JCP**[**JSR-291**](http://www.jcp.org/en/jsr/detail?id=291)**(Dynamic Component Support for Java SE - AKA OSGi 4.1)**
* **Sun JCP**[**JSR-294**](http://www.jcp.org/en/jsr/detail?id=294)**(Improved Modularity Support in the Java Programming Language)**

JAVA9 中引入了模块化系统，实现了整个平台的模块化。实现方式与OSGi不同，使用module-info文件来定义依赖关系。

## Karaf

一个轻量级的OSGI多态容器。Blueprint，feature，bundle，动态模块化这些概念都属于OSGI。



## 安全性

# MD-SAL

Model-Driven Service Abstract Layer

## MD-SAL作用是什么？

使用YANG模型来建模应用接口和数据定义。

## 什么是YANG模型

数据建模语言，用于建模netconf协议状态数据、配置数据、RPC和通知。

<https://tools.ietf.org/html/rfc6020>

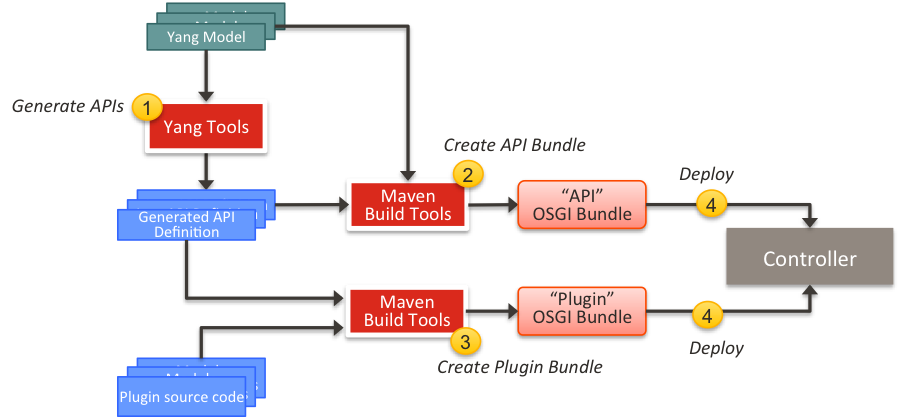
<https://tools.ietf.org/html/rfc7950>

在具备Netconf协议的前提下，如果了了解了一个netconf设备的的YANG模型，是否就知道了该如何和该设备交互？

类似的， SAL和YANG

## YANG模型在ODL开发中起什么作用

1. 接口和数据定义的建模语言
2. 代码自动生成

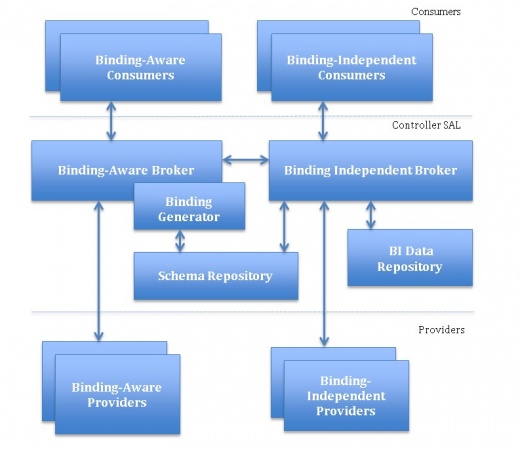


## YANG模型与代码的映射关系

编译时，YANG模型用于代码生成：

https://wiki.opendaylight.org/view/YANG\_Tools:YANG\_to\_Java\_Mapping#Container

## MD-SAL 架构



The subsystems in the above figure are as follows:

* **Provider** – a component that exposes functionality to applications and other providers (plugins) through its northbound API. A provider can be a Consumer of other Providers. There are two types of Providers:
  + **Binding Independent Providers**: their functionality is exposed in the binding-independent Data DOM format
  + **Binding Aware Providers**: their functionality is exposed in a format compiled against one or more generated binding interfaces
* **Consumer** – a component that consumes functionality provided by one or more Providers. There are two types of Consumers:
  + **Binding Independent Consumers**– the functionality is consumed in the binding-independent Data DOM format
  + **Binding Aware Consumers** – the functionality is consumed via one or more generated binding interfaces
* **Binding-Independent Broker** - the core component of the model-driven SAL. It routes RPCs, notifications and data changes between various Providers and Consumers.
* **Binding-Aware Broker** – provides programmatic APIs and Java language support to both Consumers (such as controller applications or plugins) and Providers. It is a façade / proxy built on top of the Binding Independent Broker that simplifies access to data and services provided by Binding-Independent Providers and Binding-Aware providers.
* **BI Data Repository** – is a binding-independent infrastructure component of SAL that is responsible for storage of configuration and transient data
* **Binding Schema Repository** – is infrastructure component, responsible for storing specifications of YANG–Java relationships and mapping between language-binding APIs to binding-independent API calls.
* **Binding Generator** – a SAL infrastructure component, which generates implementations of binding interfaces and data mappers to the binding-independent format.

运行时，基于YANG模型动态生成BA实现。BI 数据格式校验。

## 功能模块/应用/组件间通信模式

MD-SAL provides several messaging patterns using broker derived from basic concepts, which are intended to transfer YANG modeled data between applications to provide data-centric integration between applications instead of API-centric integration.

相关概念：

|  |  |
| --- | --- |
| **Data Tree:** | All state-related data are modeled and represented as data tree, with possibility to address any element / subtree   * **Operational Data Tree** - Reported state of the system, published by the providers using MD-SAL. Represents a feedback loop for applications to observe state of the network / system. * **Configuration Data Tree** - Intended state of the system or network, populated by consumers, which expresses their intention. |
| **Notification:** | Asynchronous transient event which may be consumed by subscribers and they may act upon it. |
| **RPC:** | asynchronous request-reply message pair, when request is triggered by consumer, send to the provider, which in future replies with reply message.  **Note**  In MD-SAL terminology, the term ‘RPC’ is used to define the input and output for a procedure (function) that is to be provided by a provider, and mediated by the MD-SAL, that means it may not result in remote call. |
|  |  |

提供三种接口风格：

## 基于MD-SAL的开发流程

**基于MD-SAL开发的优缺点**

## OSGi模块间通信

OSGi的通信机制是否可用？

<reference id="dataBroker" interface="org.opendaylight.controller.md.sal.binding.api.DataBroker" />

# ODL开发常用工具及接口

## 开发环境

1. Maven 3.5.2 or later
2. Java 8-compliant JDK
3. 设置Maven Settings.xml

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

<!-- vi: set et smarttab sw=2 tabstop=2: -->

<!--

Copyright (c) 2014, 2015 Cisco Systems, Inc. and others. All rights reserved.

This program and the accompanying materials are made available under the

terms of the Eclipse Public License v1.0 which accompanies this distribution,

and is available at http://www.eclipse.org/legal/epl-v10.html

-->

<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0 http://maven.apache.org/xsd/settings-1.0.0.xsd">

<profiles>

<profile>

<id>opendaylight-release</id>

<repositories>

<repository>

<id>opendaylight-mirror</id>

<name>opendaylight-mirror</name>

<url>http://172.18.10.215:8081/nexus/content/groups/public/</url>

<releases>

<enabled>true</enabled>

<updatePolicy>never</updatePolicy>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>opendaylight-mirror</id>

<name>opendaylight-mirror</name>

<url>http://172.18.10.215:8081/nexus/content/groups/public/</url>

<releases>

<enabled>true</enabled>

<updatePolicy>never</updatePolicy>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</profile>

<profile>

<id>opendaylight-snapshots</id>

<repositories>

<repository>

<id>opendaylight-snapshot</id>

<name>opendaylight-snapshot</name>

<url>http://172.18.10.215:8081/nexus/content/groups/snapshot/</url>

<releases>

<enabled>false</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>opendaylight-snapshot</id>

<name>opendaylight-snapshot</name>

<url>http://172.18.10.215:8081/nexus/content/groups/snapshot/</url>

<releases>

<enabled>false</enabled>

</releases>

<snapshots>

<enabled>true</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</profile>

</profiles>

<activeProfiles>

<activeProfile>opendaylight-release</activeProfile>

<activeProfile>opendaylight-snapshots</activeProfile>

</activeProfiles>

</settings>

Snashoshot：优先从远程仓库下载依赖，所以一般在开发模式下，我们可以频繁的发布Snashoshot版本，以便让其它项目能实时的使用到最新的功能做联调。

Release: 优先从本地仓库获取，用于正式发布。

## Maven

1. **优秀的构建工具**
2. **依赖管理工具**
3. **项目信息管理工具**

遵循约定优于配置的原则。因为灵活性带来混乱。

### 基本概念

1. 仓库Repository、 Artifiact
2. 生命周期LifeCycle、 阶段（phase）
3. 坐标：(groupid artifactid version)
4. **classifier：分类标识**
5. Scope：依赖的作用范围，有test， runtime，**provided**， compile

6 dependencyManagement和dependency

### Scope详解

1. **Compile（参与编译，不打包）**

默认就是compile，什么都不配置也就是意味着compile。compile表示被依赖项目需要参与当前项目的编译，当然后续的测试，运行周期也参与其中，是一个比较强的依赖。打包的时候通常需要包含进去。

1. **Test （不参与编译，不打包）**

scope为test表示依赖项目仅仅参与测试相关的工作，包括测试代码的编译，执行。比较典型的如junit。

1. **Runtime（不参与编译，但打包）**

runntime表示被依赖项目无需参与项目的编译，不过后期的测试和运行周期需要其参与。与compile相比，跳过编译而已，说实话在终端的项目（非开源，企业内部系统）中，和compile区别不是很大。比较常见的如JSR×××的实现，对应的API jar是compile的，具体实现是runtime的，compile只需要知道接口就足够了。oracle jdbc驱动架包就是一个很好的例子，一般scope为runntime。另外runntime的依赖通常和optional搭配使用，optional为true。我可以用A实现，也可以用B实现。

1. **Provided （参与编译，不打包）**

provided意味着打包的时候可以不用包进去，别的设施(Web Container)会提供。事实上该依赖理论上可以参与编译，测试，运行等周期。相当于compile，但是在打包阶段做了exclude的动作。

Kar包的生成规则： 仅依赖于feautre的bundle构成。单个feature可以构造一个kar包。或者一个featre repository里的多个feature。

### 构建时依赖来源

## 常用开发接口

[https://wiki.opendaylight.org/view/OpenDaylight\_Controller:MD-SAL:Toaster\_Step-By-Step](https://wiki.opendaylight.org/view/OpenDaylight_Controller:MD-SAL:Toaster_Step-By-Step#Wire_the_KitchenServiceImpl_for_notifications)

需了解与以下问题：

1. 如何注册与发布RPC Service
2. 如何监听数据树的变化
3. 如何注册监听通知与发布通知
4. 如何访问数据树
   1. 事务的隔离性（RR）基于乐观锁机制实现。类似git。
   2. merge不产生冲突。
5. 如何添加配置
6. 如何获取挂载点，访问挂载点数据。
7. 如何使用事务链
8. 如何将BA数据与BI数据转换

阅读Toaster的例子，应

1. 关注线程模型
2. 关注锁的机制
3. 关注YANG模型的作用
4. 关注数据树的如何使用（监听器持有状态）

关于挂载的含义：

# F版本Feature生成规则

F版本的feature解析产生了变化，可基于feature项目的Pom生成feature.xml文件。

Feature生成规则示例

针对上述变化，依赖的书写会要求更严格，应了解如下规则：

1. 无论pom的feature是什么scope，都会参与featurex.ml生成的依赖解析，意味着feature所包含的依赖，不会在目标feautre的定义中出现。
2. 无论pom中的feature是什么顺序，都会优先启动。
3. pom的依赖解析遵循广度优先原则。这意味着，如果解析依赖中出现了不同的版本，则后解析到的版本不会出现在feature中。运行时产生版本依赖错误。
4. 若src/main/feature.xml存在，则会合入该文件内容。
5. 对于配置文件的书写，必须借助src/main/feature.xml，且配置文件下需存在feature。**可以解决编译期多版本依赖冲突的问题**

**如何解决启动慢，内存消耗大的问题，**重点在于依赖能够按照规范添加。避免bundle被重复refresh。

1. 直接依赖必须显示添加
2. 未使用依赖应删除
   1. 1,2使用mvn dependency:analyze分析， 部分通过反射访问的依赖，会误认为未使用该依赖。
3. 版本号集中控制 （对速度没影响，仅方便管理）
4. 在feature的pom.xml中添加包含所清除目标依赖的feature。
5. 应明确依赖scope。例如环境提供的依赖，应标注为provided(4或5 至少选一种方式，或者两者兼用。 优先使用第5种)
6. 依赖顺序上应保证被依赖的bundle在feature.xml中处于靠前位置。

理想情况，应产生一个最小化的feature.xml。如下TRM为例：

|  |
| --- |
| <features xmlns=*"http://karaf.apache.org/xmlns/features/v1.4.0"* name=*"odl-trm-core"*>  <repository>mvn:com.utstar.trm/odl-trm-model/1.0.1/xml/features</repository>  <repository>mvn:com.utstar.trm/odl-trm-common/1.0.1/xml/features</repository>  <feature name=*"odl-trm-core"* description=*"odl-trm-core"* version=*"1.0.1"*>  <details>OpenDaylight is leading the transformation to Open Software Defined Networking (SDN). For more information, please see https://www.opendaylight.org</details>  <feature version=*"1.0.1"* prerequisite=*"false"* dependency=*"false"*>odl-trm-model</feature>  <feature version=*"1.0.1"* prerequisite=*"false"* dependency=*"false"*>odl-trm-common</feature>  <feature prerequisite=*"true"* dependency=*"false"*>wrap</feature>  <bundle>mvn:com.utstar.trm/alarm/1.0.1</bundle>  <bundle>mvn:com.utstar.trm/capability-discovery/1.0.1</bundle>  <bundle>mvn:com.utstar.trm/topology-discovery/1.0.1</bundle>  <bundle>mvn:com.utstar.trm/resource/1.0.1</bundle>  <bundle>mvn:com.utstar.trm/service-discovery/1.0.1</bundle>  </feature>  </features> |

# 开发规范

[**https://github.com/zhanghuafei/UT-ODL-Coding-Tutorial**](https://github.com/zhanghuafei/UT-ODL-Coding-Tutorial)

# 自动部署

Watch

Databroker的使用

**feature:uninstall不支持**