OpenDayLight Framework and Architecture

2015.07.28
王昱傑 <YuChiehWang@itri.org.tw>
網路通訊服務技術部, ICL/ITRI
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
Opendaylight

Platinum

Gold

Silver
Opendaylight Controller Edition

- **Hydrogen (Release Date: 2014/02/04)**
  - Base Edition
  - Virtualization Edition
  - Service Provider Edition

- **Helium (Release Date: 2014/09/29)**
  - SR1 (Release Date: 2014/11/10)
  - SR2 (Release Date: 2015/01/30)
  - SR3 (Release Date: 2015/03/19)

- **Lithium (Release Date: 2015/06/29)**
  - 尚未有 stable release
Hydrogen: Base Edition
Virtualization Edition
Service Provider Edition
Helium

The information contained herein is the exclusive property of ITRI and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of ITRI.
Lithium
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
Opendaylight Developing Environment

Project Build Up
JAVA + Maven

compile

OSGi Bundle

run

Runtime OSGi Framework

Hydrogen: Equinox
Helium, Lithium: Karaf

The information contained herein is the exclusive property of ITRI and shall not be distributed, reproduced, or disclosed in whole or in part without prior written permission of ITRI.
OSGi Concept

- Stands for Open Services Gateway initiative
  - is a specification introduced by OSGi Alliance

- Forces programmers to decouple each module

- Advantages
  - Bundles can join/leave the platform dynamically at anytime
  - Manage the dependencies so that there is no more unexpected “ClassNotFoundException” be thrown during runtime

- Disadvantages
  - Need to consider the situation that required services are not available
  - Need to manage lots of dependencies (import/export/services)
OSGi Architecture
Bundle

- A .jar file is called a bundle in OSGi.
- A bundle is a basic unit in OSGi framework
  - Contains a MANIFEST.MF file
- A bundle can include many packages
  - ex. api & implementation
- A bundle can interact with other bundles
  - Import/Export
  - Service
MANIFEST.MF (1/2)

- in .jar file: META-INF/MANIFEST.MF
  - OSGi framework identifies a bundle through its MANIFEST.MF file

- Manifest-Version
  - The version of this manifest file (not very important)

- Bundle-Name
  - Name of this bundle (for human understanding, not very important)

- Bundle-Activator
  - Tells OSGi framework what to do when a bundle start/stop

http://wiki.osgi.org/wiki/Category:Manifest_Header
MANIFEST.MF (2/2)

- **Bundle-SymbolicName**
  - Used by OSGi framework to identify a bundle, must be unique

- **Bundle-Version**
  - The version of this bundle (used by Import-Package/Export-Package)

- **Import-Package**
  - Tells OSGi framework to require a package from another bundle

- **Export-Package**
  - Tells OSGi framework to export a package to let other bundles use it
Import and Export Function (1/2)

- Manifest.mf

```
Manifest-Version: 1.0
Bundle-Name: HelloWorld
Bundle-Activator: HelloWorld
Bundle-SymbolicName: HelloWorld
Bundle-Version: 1.0.0
Import-Package: org.osgi.framework
```

- Import-Package: import the file needed from the desired package
Import and Export Function (2/2)

- Export-Package

![MoviesInterface.mf - 記事本](image)

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Movies Interface
Bundle-SymbolicName: MoviesInterface
Bundle-Version: 1.0.0
Export-Package: osgitut.movies;version="1.0.0"
```

- Requirements for other bundles to use this interface
  - This bundle Export-Package
  - Other bundles Import-Package
  - This bundle is being installed on OSGi framework

- Error message
  - Missing imported package osgitut.movies;version="1.0.0"
Interact with OSGi Framework

- OSGi framework calls a bundle’s BundleActivator method when the bundle start/stop
  - A bundle gets information and interacts with other bundles through BundleContext object which provided by OSGi framework → it’s the only way to interact with other bundles

```java
import org.osgi.framework.*;

public class HelloActivator implements BundleActivator {
    public void start(BundleContext context) {
        System.out.println("Hello EclipseZone Readers!");
    }

    public void stop(BundleContext context) {
        System.out.println("Goodbye EclipseZone Readers!");
    }
}
```
Example: Karaf (1/5)

- **Startup command (for Opendaylight Helium)**
  - ./bin/karaf

```
shock@shock-VirtualBox:~/helium/distribution-karaf-0.2.3-Helium-SR3$ ./bin/karaf

OpenDaylight

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.
```
Example: Karaf (2/5)

- see bundle list: list

```bash
opendaylight-user@root> list
START LEVEL 100 , List Threshold: -1
ID | State | Lvl | Version | Name
---|-------|-----|---------|------------------
 0 | Active| 0   | 3.8.2.v20130124-134944 | OSGi System Bundle
 1 | Active| 5   | 1.4.0   | OPS4J Base - Lang
 2 | Active| 5   | 2.2.0.20130927-173417 | bndlib
 3 | Active| 5   | 1.7.0   | OPS4J Pax Swissbox :: Bnd Utils
 4 | Active| 5   | 1.6.0   | OPS4J Pax Url - Maven Commons
 5 | Active| 5   | 1.6.0   | OPS4J Pax Url - aether:
 6 | Active| 5   | 1.6.0   | OPS4J Pax Url - wrap:
 7 | Active| 8   | 1.7.2   | OPS4J Pax Logging - API
 8 | Active| 8   | 1.7.2   | OPS4J Pax Logging - Service
 9 | Active| 10  | 3.0.1   | Apache Karaf :: Service :: Guard
```
Example: Karaf (3/5)

- Bundle control command
  - install file:name.jar
  - uninstall [bundle ID]
  - start [bundle ID]
  - stop [bundle ID]
  - refresh [bundle ID]
  - diag [bundle ID]

- Bundle status:
  - INSTALLED – 安装完成, 但因为dependency未满足而不可执行
  - RESOLVED – 安装完成且dependency已满足, 但未执行
  - ACTIVE – 執行中
Example: Karaf (4/5)

- features
  - 一個feature可能包含0到多個bundle
  - feature之間也相互有dependency的關係
Example: Karaf (5/5)

- **features command**
  - install [feature_name]
  - uninstall [feature_name]
  - repo-list
  - repo-add [feature_name]
  - repo-remove [feature_name]

- 使用者可自行選擇要開啓的功能
  - feature使用xml格式來記錄

```xml
<feature name='odl-openflowplugin-southbound' description='OpenDaylight :: Openflow Plugin :: SouthBound'>
<feature version='1.1.3-Helium-SR3'>odl-mdsal-broker</feature>
<feature version='1.1.3-Helium-SR3'>odl-flow-model</feature>
<feature version='0.5.3-Helium-SR3'>odl-openflowjava-protocol</feature>
<bundle>mvn:org.opendaylight.openflowplugin/openflowplugin/0.0.6-Helium-SR3</bundle>
<bundle>mvn:org.opendaylight.openflowplugin-api/0.0.6-Helium-SR3</bundle>
<bundle>mvn:org.opendaylight.extension-api/0.0.6-Helium-SR3</bundle>
<bundle>mvn:org.opendaylight.controller/lblldp/0.8.4-Helium-SR3</bundle>
<bundle>mvn:org.opendaylight.controller.md/topology-lldp-discovery/1.1.3-Helium-SR3</bundle>
```
Type of Dependencies

- **Package dependencies**
  - managed before a bundle can be started
  - managed by OSGi framework
  - using import & export

- **Service dependencies**
  - managed during runtime
  - must be dynamic
  - A bundle must be able to register & unregister at any time
Service

- An object registered with the service registry under one or more interfaces together with properties. This object can be discovered and used by bundles.

- Basic Service
  - The basic functionality of OSGi framework, lack of flexibility

- Dynamic Service
  - which reveals the true power of OSGi
  - A bundle can register/unregister its service at any time
OSGi Service pattern

- API+Consumer+Provider
  - provider/consumer 之間無法直接看到彼此，而是透過 API 來互動
  - 達到 decouple 的效果
Basic Service (1/3)

- **API**
  - only provide interface
  - 不需要 Activator, 只要安装到 OSGi framework 上面
  - Consumer/Provider 皆需要 Import API package

```java
package org.equinoxosgi.toast.dev.gps;

public interface IGps {
    public abstract int getHeading();
    public abstract int getLatitude();
    public abstract int getLongitude();
    public abstract int getSpeed();
}
```
Basic Service (2/3)

- **Service Provider**
  - Call: `context.registerService()` method to register a service to OSGi framework

```java
private ServiceRegistration registration;

public void start(BundleContext context) {
    FakeGps gps = new FakeGps();
    registration = context.registerService(IGps.class.getName(), gps, null);
}

public void stop(BundleContext context) {
    registration.unregister();
}
```
Basic Service (3/3)

- Service Consumer:

```java
public void start(BundleContext context) throws Exception {
    System.out.println("Launching");
    monitor = new EmergencyMonitor();
    gpsRef = context.getServiceReference(IGps.class.getName());
    airbagRef = context.getServiceReference(IAirbag.class.getName());
    if (gpsRef == null || airbagRef == null) {
        System.err.println("Unable to acquire GPS or airbag!");
        return;
    }
    gps = (IGps) context.getService(gpsRef);
    airbag = (IAirbag) context.getService(airbagRef);
    if (gps == null || airbag == null) {
        System.err.println("Unable to acquire GPS or airbag!");
        return;
    }
```
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
Maven

- Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

- Maven - as its heart – a plugin execution framework; all work is done by plugins.
  - Maven is a Java tool
Maven Project Architecture

- Create a new maven project

```sh
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DgroupId=com.mycompany.app \
  -DartifactId=my-app
```

```
my-app
|-- pom.xml
 `-- src
    |-- main
    |   |-- java
    |       |-- com
    |           |-- mycompany
    |               |-- app
    |                   `-- App.java
    `-- test
       |-- java
          |-- com
             |-- mycompany
                |-- app
                   `-- AppTest.java
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.mycompany.app</groupId>
    <artifactId>my-app</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>Maven Quick Start Archetype</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
pom.xml (2/2)

- Project Object Model (POM)
- Configuration file of maven project
  - Developers’ information
  - Project information
  - Project dependencies
  - URL of where project lives
  - Organization and licensese

- In fact, in the Maven world, a project need not contain any code at all, merely a pom.xml.

http://maven.apache.org/pom.html
pom structure (1/2)

- **parent**
  - 繼承自哪個pom.xml

```xml
<parent>
  <groupId>org.opendaylight.controller</groupId>
  <artifactId>commons.opendaylight</artifactId>
  <version>1.4.1</version>
  <relativePath>../commons.opendaylight</relativePath>
</parent>
```

- **project information**
  - <artifactId>hosttracker.implementation</artifactId>
    <version>0.5.1</version>
    <packaging>bundle</packaging>
pom structure (2/2)

dependency management

```xml
<dependencies>
  <dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>topologymanager</artifactId>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>switchmanager</artifactId>
  </dependency>
  <dependency>
    <groupId>org.opendaylight.controller</groupId>
    <artifactId>clustering.services</artifactId>
  </dependency>
```

...
Maven Dependency

- in pom.xml

```xml
<dependencies>
    <dependency>
        <groupId>org.opendaylight.controller</groupId>
        <artifactId>clustering.services</artifactId>
    </dependency>
    <dependency>
        <groupId>org.opendaylight.controller</groupId>
        <artifactId>sal.connection</artifactId>
    </dependency>
</dependencies>
```

- 在 java code 中 import 其他 project 内的 package 的时候必须加入相对应的 dependency
  - maven 不允许 circular dependency
  - if A depends on B & C, B depends on C, A 可以只 depends on B 就通过 compile → dependency 是串联的
Maven Phases (1/2)

- **validate**
  - validate the project is correct and all necessary information is available.
  - command: mvn xxx

- **compile**
  - Maven 3.0 uses javax.tools.JavaCompiler as the default java compiler

- **test**
  - test the compiled source code using a suitable unit testing framework.
  - 會執行test裡面的每一個test.java檔案

- **package**
  - take the compiled code and package it in its distributable format, such as JAR.
Maven Phases (2/2)

- **install**
  - install the package into the local repository, for use as a dependency in other projects locally
  - ~/.m2/repository/

- **clean**
  - cleans up artifacts created by prior builds
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
SAL (1/2)
SAL (2/2)

Service Abstraction Layer

-  The purpose is to let northbound application & southbound protocol plugin interfaces be separate.
-  northbound does not need to know the switch's version and status to give instructions.

AD-SAL

-  API Driven
-  Uses fixed general API implementation, providing standardized API.

MD-SAL

-  Model Driven
-  Uses yang tools to generate a unified format API, with service consumer/provider implementation.
-  Structurally similar to OSGi.
使用規範好的general API來做溝通

- 無法動態加入新的使用者自訂API
- 想要加入新的API時，需要修改SAL內部結構
AD-SAL API 命名機制

■ 由 SAL implement:
  ◆ IListenxxx: 使 bundle 可以監聽某事件之變化
  ◆ IxxxService: 使 bundle 可以從 sal 取得當前某類別的資料 & 進行操作
  ◆ IPluginOutxxx: 使 protocol plugin 可以呼叫 sal 的方法

■ 由 southbound bundle implement:
  ◆ IPluginInxxx: 使 sal 可以呼叫 protocol plugin 的方法

■ Example
  ◆ northbound application add flow:
    use IFlowProgrammerService →
    call IPluginInFlowProgrammerService
  ◆ southbound delete flow:
    call IPluginOutFlowProgrammerService
# AD-SAL APIs Example

<table>
<thead>
<tr>
<th>API type</th>
<th>Southbound implements</th>
<th>AD-SAL implements</th>
<th>Northbound uses/implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataPacketService</td>
<td>IPluginInDataPacketService</td>
<td>IPluginOutDataPacketService</td>
<td>IDataPacketService/IListenDataPacket</td>
</tr>
<tr>
<td>ReadService</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>InventoryService</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>TopologyService</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
使用 yang tools 產生 java APIs

- 架構上類似 OSGi
- 可以讓使用者動態加入自訂 APIs，不須修改 SAL 內部架構
- 不區分 southbound/northbound，只定義 provider/consumer
Overview

■ 透過 BindingAwareBroker 提供 MD-SAL 服務 (也藉此區分非 MD-SAL 的其他部分)
■ 將 service 的種類 abstract 成三種
  ◆ RPC type service
  ◆ Data type service
  ◆ Notification type service
MD-SAL Service

- **RPC Service**
  - remote procedure call
  - provider提供API的實作, consumer呼叫provider來執行功能

- **Data Service**
  - 存取MD-SAL內部的資料庫
  - 整個controller裡面使用同一份
  - 採用(key, value)方式實作

- **Notification Service**
  - 當某件事發生時, provider傳送xxx notification (type可自行定義)
  - 有向MD-SAL註冊的consumer可收到此notification, 再執行相對應的動作
Data Type Specification

- **YANG (RFC 6020)**
  - A data modeling language for the Network Configuration Protocol (NETCONF).

- **NETCONF (RFC 6241)**
  - Provides a mechanism to install, manipulate and delete the configuration of network devices.

- **RESTCONF (Internet-Draft)**
  - Provides a programmatic interface for accessing data defined in YANG, using the datastores defined in NETCONF.
  - Opendaylight MD-SAL使用此protocol做為REST介面。
API generated from Yang Models
Yang File Example (1/3)

- **Source**: openflowplugin/model/model-flow-service

- **RPC Service**

  ```
  rpc add-flow {
    description "Adding flow to openflow device.";
    input {
      uses tr:transaction-metadata;
      leaf flow-ref {
        type types:flow-ref;
      }
      uses node-flow;
    }
    output {
      uses tr:transaction-aware;
    }
  }
  ```

- **Generated java file**

  ```java
  Future<RpcResult<AddFlowOutput>> addFlow(AddFlowInput input);
  Future<RpcResult<RemoveFlowOutput>> removeFlow(RemoveFlowInput input);
  ```
Yang File Example (2/3)

- **Notification**

```java
notification flow-added {
    status deprecated;
    uses tr:transaction-metadata;
    leaf flow-ref {
        type types:flow-ref;
    }
    uses node-flow;
    uses tr:transaction-aware;
}
```

- **Generated java file**

```java
public interface FlowAdded
    extends ChildOf<DataObject>,
    Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.flow.service.rev130819.FlowAdded>,
    TransactionMetadata,
    NodeFlow,
    TransactionAware,
    Notification
{
```
Yang File Example (3/3)

- Data Type Define

```yang
grouping flow-update {
  description "DataStore aware flow update structure suitable for rpc input (contains node-context).";
  uses "inv:node-context-ref";

  container original-flow {
    uses types:flow;
  }
  container updated-flow {
    uses types:flow;
  }
}
```

- Generated java file

```java
public interface OriginalFlow extends ChildOf<FlowUpdate>,
    Augmentable<org.opendaylight.yang.gen.v1.urn.opendaylight.flow.service.rev130819.flow.update.OriginalFlow {

  public static final QName QName = org.opendaylight.yangtools.yang.common.QName.create("urn:openDayl
```
Agenda

- Opendaylight Architecture
- Opendaylight Developing Framework
  - OSGi
  - Maven
- Opendaylight Core Functions
  - SAL
  - Core Bundles
MD-SAL Core Bundles

- **inventory-manager**
  - 負責管理switch的資訊，包括table數量、flow、port status等。
  - 寫入DataStore。

- **topology-manager**
  - 負責管理topology的資訊，包括switch之間的link資訊，並寫入DataStore。

- **statistics-manager**
  - 負責定期向底層switch發送request詢問當下狀況，包括table、flow、port statistics等資訊。
  - 將資訊回報給inventory-manager、topology-manager模組。
Inventory Data Example

```
{
  "nodes": {
    "node": [
      {
        "id": "openflow:30",
        "flow-node-inventory:software": "1.4.6",
        "flow-node-inventory:serial-number": "None",
        "flow-node-inventory:manufacturer": "Nicira Networks, Inc.",
        "flow-node-inventory:hardware": "Open vSwitch",
        "flow-node-inventory:description": "None",
        "flow-node-inventory:ip-address": "127.0.0.1",
        "flow-node-inventory:switch-features": []
      }
    ],
    "capabilities": [
      "flow-node-inventory:flow-feature-capability-flow-stats",
      "flow-node-inventory:flow-feature-capability-port-stats",
      "flow-node-inventory:flow-feature-capability-table-stats",
      "flow-node-inventory:flow-feature-capability-queue-stats",
      "flow-node-inventory:flow-feature-capability-arp-match-ip"
    ],
    "max_buffers": 256,
    "max_tables": 255
  }
}
```
Topology Data Example

- Link information

```json
{
    "network-topology": {
        "topology": [
            {
                "topology-id": "flow:1",
                "node": [
                    {
                        "node-id": "openflow:40",
                        "termination-point": [
                            {
                                "tp-id": "openflow:40:1",
                                "opendaylight-topology-inventory:invento
inventory:node/opendaylight-inventory:node[opendaylight-inventory:id='op
connector[opendaylight-inventory:id='openflow:40:1']]"
                            },
                            {
                                "tp-id": "openflow:40:2",
                                "opendaylight-topology-inventory:invento
inventory:node/opendaylight-inventory:node[opendaylight-inventory:id='op
connector[opendaylight-inventory:id='openflow:40:1']]"
                            }
                        ]
                    }
                ]
            }
        ]
    }
}
```
Q & A

- Thank you for listening!