OpenDaylight Carbon二次开发实用指南

林潇 • 17-09-29 • 2458 人围观

作者简介:林潇，同济大学SNLab（先进网络与系统联合中心），研三在读。热爱分享，有OpenDaylight开发经验，OpenDaylight ALTO项目contributor。

通过本文你将知道：

1. Maven Archetype的基本原理以及如何使用Maven Archetype生成适用于不同版本的ODL子项目。
2. 本文将着重讲解cli命令开发，以及Carbon Release中新引入的Blueprint的一些基本知识。OpenDaylight Carbon Release中模块运行的大致流程以及对于api和impl的开发可以参考ODL碳版本模块开发及流程梳理还有ODL controller官方开发指南（它对DataStore的描述相当不错）。
3. 如何将编写好的应用添加到一个正在运行的OpenDaylight控制器中。

Maven Archetype的原理与其在OpenDaylight中的应用

Archetype[1]是一个Maven项目模板的工具。它提供了一种快速生成一致的Maven项目的方法。事实上OpenDaylight的Archetype存储在controller项目中。大家可以从github或者git.opendaylight.org中访问controller项目，并找到Archetype对应的位置。在作者编写本篇教程时，使用下述URI下载的Archetype源码（建议使用git clone下载该源码，方便后面使用git checkout切换源码版本）。

```
1 https://github.com/opendaylight/controller/tree/master/opendaylight/archetypes
```

让我们看看Archetype的源文件，它与普通的文件并无很大区别，但它包含`${package}`、`${classPrefix}`等一些可被替换的参数。这些参数可以在用户利用Archetype生成自己的项目，使用`mvn archetype:generate`命令时被指定。

```
1  https://github.com/opendaylight/controller/blob/release/carbon-
   sr1/opendaylight/archetypes/opendaylight-startup/src/main/resources/archetype-
   resources/impl/src/main/java/__packageInPathFormat__/impl/__classPrefix__Provider.jav
2
3  #set( $symbol_pound = '#' )
4  #set( $symbol_dollar = '$' )
5  #set( $symbol_escape = '' )
6  /*
7   * Copyright © ${copyrightYear} ${copyright} and others.  All rights reserved.
8   *
9   * This program and the accompanying materials are made available under the
10  * terms of the Eclipse Public License v1.0 which accompanies this distribution,
11  * and is available at http://www.eclipse.org/legal/epl-v10.html
12  */
13 package ${package}.impl;
14 import org.opendaylight.controller.md.sal.binding.api.DataBroker;
15 import org.slf4j.Logger;
16 import org.slf4j.LoggerFactory;
17 public class ${classPrefix}Provider {
18     private static final Logger LOG =
```

林潇 发表于17-09-29

2 2

```
22          this.dataBroker = dataBroker;
23      }
24      /**
25       * Method called when the blueprint container is created.
26       */
27      public void init() {
28          LOG.info("${classPrefix}Provider Session Initiated");
29      }
30      /**
31       * Method called when the blueprint container is destroyed.
32       */
33      public void close() {
34          LOG.info("${classPrefix}Provider Closed");
35      }
    }
```

对于开发过之前版本的OpenDaylight用户而言，这些函数应该很眼熟，只不过原来的onSessionInitiated()变成init()。Carbon Release中使用Blueprint作为依赖注入Dependency Injection框架，关于Blueprint的细节详见第二章。

注：Maven仓库[2]分为**本地仓库**与**远端仓库**。之前的很多教程是直接从OpenDaylight Maven远端仓库（往往是一个通过一个URI指定）上拉取Archetype的artifact（Maven仓库里的货物可以被称为artifact或者project）然后生成自己的项目架构。但这个URI可能在作者撰写教程的时候存在，当OpenDaylight有了新的小版本更新后这个URI就失效了。因此本教程将说明如何**利用**OpenDaylight Archetype**源码以及**Maven**本地仓库，生成任意版本的**OpenDaylight**子项目。**

既然有仓库，那么里面一定有货物（project或者叫做artifact），而且有不止一件货物。为了区分货物，每件货物都需要有自己的ID。而这个ID**由**groupId、artifactId、version**构成。**

```
org.opendaylight.controller
opendaylight-startup-archetype
1.3.1-Carbon
```

这些属性在命令`mvn archetype:generate`中用来指定即将要生成的项目是依赖于哪个Archetype的artifact。

为了使用Archectype，需要先将Archetype安装（`maven install`）到本地Maven仓库（在Ubuntu或者macOS下默认的本地Maven仓库路径为`~/.m2`）。具体操作如下：

1. **将**Archetype**的源码编译安装到本地仓库（在本地仓库生成了一个该**Archetype**对应的**artifact**），以下命令运行在控制台（**Terminal**）。**

```
# 下载controller项目
git clone https://github.com/opendaylight/controller.git
# 进入controller目录
cd controller
# 切换到carbon-sr1版本
git checkout release/carbon-sr1
# 进入Archetype/opendaylight-startup目录
cd opendaylight/archetypes/opendaylight-startup/
# 将Archetype安装到本地仓库
mvn clean install
```

注：其他版本格式均为"release/大版本号-小版本号"或者"release/大版本号"，大版本号为某一个元素全拼，小版本号为sr1、sr2等等。

2. **使用`mvn archetype:generate`命令，从本地仓库中获取安装过的**Archetype**的**artifact**，生成自己所需的子项目。**以下命令运行在控制台（Terminal）。其中带

artifact，该信息可以在**上文执行**mvn clean install**的目录**下的pom.xml文件中得到。DgroupId和DartifactId是即将生成的项目的groupId和artifactId。关于groupId、artifactId和version的命名规范请见[3]。version可以不填写，默认的version为0.1.0-SNAPSHOT。

```
mvn archetype:generate -B
  -DarchetypeGroupId=org.opendaylight.controller
  -DarchetypeArtifactId=opendaylight-startup-archetype
  -DarchetypeVersion=1.3.1-Carbon
  -DgroupId=snlab
  -DartifactId=helloworld
  -DclassPrefix=HelloWorld
  -Dcopyright=snlab
  -DcopyrightYear=2017
```

注：前面的空格不要漏掉！

3. **观察与编译依照**OpenDaylight Carbon SR1 startup**模板生成的项目**helloworld

```
# 进入helloworld目录
cd helloworld
# 查看该目录下的内容，对于该目录下的内容详见第二章
ll
# 编译helloword，并将相应artifact安装到本地Maven仓库
mvn clean install
```

至此，你应该了解Maven Archtype生成的原理以及如何生成任意OpenDaylight版本的子项目。

Opendaylight Carbon release cli开发以及Blueprint基

OpenDaylight Carbon release的利用Archetype作为模板生成后的项目结构如下：

```
api/
artifacts/
cli/
features/
impl/
it/
karaf/
src/
```

cli目录是Carbon版本里新出现的目录，它用于方便快捷地开发Karaf cli命令。它包含api、commands和impl三个子目录。api中主要是定义Karaf命令方法签名和Javadoc。在impl中，我们可以利用Java Annotation快速的定义了命令的格式，快速获得命令解析功能，以及实现相应命令对应的输出。具体而言，cli子模块的目录结构如下：

```
├── pom.xml
└── src
    └── main
        ├── java
        │   └── snlab
        │       └── cli
        │           ├── api
        │           │   └── HelloworldCliCommands.java
        │           ├── commands
        │           │   └── HelloworldCliTestCommand.java
        │           └── impl
        │               └── HelloworldCliCommandsImpl.java
        └── resources
            └── org
                └── opendaylight
                    └── blueprint
                        └── cli-blueprint.xml
```

逐个查看各个java文件。

林潇 发表于17-09-29

```
1 Object testCommand(Object testArgument);
```

HelloworldCliTestCommand.java中定义了这个命令的格式，以及对应输出应该如何处理。例如本示例中定义了一个command，它以”test-command”开头，并且接受-tA的参数（在Karaf控制台中使用这个command的例子`opendaylight-user@root> test-command -tA 123`）。参考文献[6]中给出了更加详细的介绍。

```
1  // HelloworldCliTestCommand.java
2  /**
3   * This is an example class. The class name can be renamed to match the command
   implementation that it will invoke.
4   * Specify command details by updating the fields in the Command annotation below.
5   */
6  @Command(name = "test-command", scope = "add the scope of the command, usually
   project name", description = "add a description for the command")
7  public class HelloworldCliTestCommand extends AbstractAction {
8
9      private static final Logger LOG =
10 LoggerFactory.getLogger(HelloworldCliTestCommand.class);
11     protected final HelloworldCliCommands service;
12
13     public HelloworldCliTestCommand(final HelloworldCliCommands service) {
14         this.service = service;
15     }
16
17     /**
18      * Add the arguments required by the command.
19      * Any number of arguments can be added using the Option annotation
20      * The below argument is just an example and should be changed as per your
   requirements
21      */
22     @Option(name = "-tA",
23             aliases = { "--testArgument" },
24             description = "test command argument",
25             required = true,
26             multiValued = false)
27     private Object testArgument;
28
```

```
32          /**
33           * Invoke commannd implementation here using the service instance.
34           * Implement how you want the output of the command to be displayed.
35           * Below is just an example.
36           */
37          final String testMessage = (String) service.testCommand(testArgument);
38          return testMessage;
        }
    }
```

HelloworldCliCommandsImpl继承了HelloworldCliCommands，并实现了testCommand方法。

```
1  // HelloworldCliCommandsImpl.java
2  public class HelloworldCliCommandsImpl implements HelloworldCliCommands {
3
4      private static final Logger LOG =
5  LoggerFactory.getLogger(HelloworldCliCommandsImpl.class);
6      private final DataBroker dataBroker;
7
8      public HelloworldCliCommandsImpl(final DataBroker db) {
9          this.dataBroker = db;
10         LOG.info("HelloworldCliCommandImpl initialized");
11     }
12
13     @Override
14     public Object testCommand(Object testArgument) {
15         return "This is a test implementation of test-command";
16     }
    }
```

上面列出了代码实现的部分，下面将介绍这个模块是如何被加载的。这与helloworld/impl/是一样的，使用Blueprint做Denpendency Injection。

Blueprint是一个为OSGi容器设计的Dependency Injection系统[4, 5]。Karaf包含了Apache Aries Blueprint的实现以及它的基本特色。

当一个bundle包含一个或者多少Blueprint XML文件时，会被认为是Blueprint bundles。这些Blueprint XML位于OSGI-INF/blueprint/目录下。Blueprint使用一个extender bundle来监视其他bundle的状态。一旦extender决定一个bundle是Blueprint bundle，它就为这个bundle创建一个Blueprint Container。

Bluepinrt Container负责以下的事情：

- 解析Blueprint XML文件
- 初始化组件
- 将组件连接在一起
- 注册services
- 查找service reference

Blueprint的配置文件会被用来创建命令并将其注册到OSGi注册表中，这使得命令可以用于Karaf的控制台。cli模块的Blueprint xml文件位于cli/src/main/resources/下。reference标签让我们的子模块获取到了ODL本身最重要的模块DataBroker，通过它我们能够读写ODL中的DataStore。id为cliCommandsImpl的标签bean，告诉系统cliCommandsImpl这个模块需要从ODL中获取dataBroker。service标签告诉系统cliCommandsImpl这个bean是一个OSGi service。command-bundle标签是cli中相对最重要的，它把Karaf控制台输入的命令，前端处理类HelloworldCliTestCommands,以及对应的后端处理的类cliCommandsImpl联系到一起。

林潇 发表于17-09-29

```
3
4
5
6
7
8
9
10
11          <command></command>
12
13
14
15
16
17
```

OpenDaylight Karaf目录结构以及向运行中的Karaf载入外部Kar包

OpenDaylight Karaf的目录包含以下几个部分，具体每个部分的配置选项的含义，可以直接进入对应文件夹，查看对应文件的注释部分：

```
1 /bin：包含了开始、停止、登陆等脚本。
2 /etc：配置文件
3 /data：工作目录
4   /data/cache: OSGi框架bundle的缓存
5   /data/generated-bundles: 部署者使用的临时文件
6   /data/log: log文件
7 /deploy：热部署目录
8 /lib：包含类
9 /system：OSGi bundles仓库
```

1. **下载**OpenDaylight**控制器，本例中为**Carbon SR1。

```
1 https://www.opendaylight.org/technical-community/getting-started-for-
```

2. **解压后，进入**./distribution-karaf-0.6.1-Carbon/etc**目录，在文件**org.ops4j.pax.url.mvn.cfg**文件，添加一行：**

```
org.ops4j.pax.url.mvn.defaultRepositories=
file:${karaf.home}/${karaf.default.repository}@id=system.repository@snapshots,
file:${karaf.data}/kar@id=kar.repository@multi@snapshots
```

3. **启动**karaf，**并在**karaf**中执行以下命令，将位于**helloworld/features/target**下的**kar**包安装到正在运行的**OpenDaylight**控制器上。**

```
# 安装自己的编译生成的kar包到控制器。kar包的位置位于helloworld/features/target下
opendaylight-user@root&gt; kar:install file:/Users/shawn/Develop/alto-sdnlab/helloworld/features/target/helloworld-features-0.1.0-SNAPSHOT.kar

# 可以看到我们的项目helloworld已经被加入到正在运行的ODL Carbon SR1控制器上
opendaylight-user@root&gt; feature:list | grep hello
odl-helloworld-api                              | 0.1.0-SNAPSHOT   | x         | odl-helloworld-SNAPSHOT             | OpenDaylight :: helloworld :: api
odl-helloworld                                  | 0.1.0-SNAPSHOT   | x         | odl-helloworld-SNAPSHOT             | OpenDaylight :: helloworld
odl-helloworld-rest                             | 0.1.0-SNAPSHOT   | x         | odl-helloworld-SNAPSHOT             | OpenDaylight :: helloworld :: REST
odl-helloworld-ui                               | 0.1.0-SNAPSHOT   | x         | odl-helloworld-SNAPSHOT             | OpenDaylight :: helloworld :: UI
odl-helloworld-cli                              | 0.1.0-SNAPSHOT   | x         | odl-helloworld-SNAPSHOT             | OpenDaylight :: helloworld :: CLI

# 查看我们的项目helloworld中打印的log信息
opendaylight-user@root&gt;log:display | grep hello
......
2017-09-22 20:21:29,543 | INFO  | l for user karaf | helloworldCliCommandsImpl                  | snlab.helloworld-cli - 0.1.0.SNAPSHOT | helloworldCliCommandImpl initialized
......
```

写在最后

林潇 发表于17-09-29

我相信乐于分享能够使得大家共同进步！想要了解更多最前沿SDN技术相关的内容（相关论文、相关软件）可以访问我们实验室全体成员共同收集的awesome-sdn列表：https://github.com/snlab-freedom/awesome-sdn 。欢迎各位乐于分享的小伙伴帮助我们共同完善这份列表！

参考文献

[1] https://maven.apache.org/guides/mini/guide-creating-archetypes.html

[2] https://maven.apache.org/guides/introduction/introduction-to-repositories.html

[3] https://maven.apache.org/guides/mini/guide-naming-conventions.html

[4] https://wiki.opendaylight.org/view/Using_Blueprint

[5] http://aries.apache.org/modules/blueprint.html

[6] https://karaf.apache.org/manual/latest-2.x/developers-guide/extending-console.html

林潇 发表于17-09-29

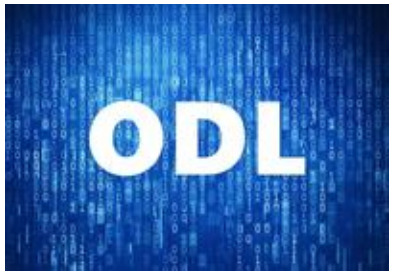
本文链接：https://www.sdnlab.com/19931.html

分享到：

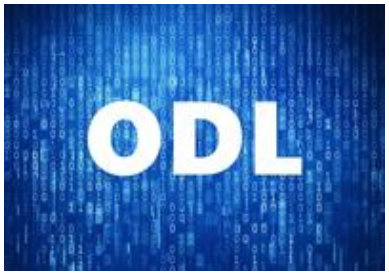
☆2 👍2

相关文章

使用Docker容器构建ODL集群



(八)ODL Openflowplugin Switch Rpc封装及流表下发源码分析



（七）ODL Openflowplugin Switch断开控制器下线源码分析

林潇 发表于17-09-29

☆2 👍2

使能容器网络，Jaguar “Sky”版本发布

（六）ODL Openflowplugin 控制器成为SLAVE过程源码分析

解构ODL：从代码到架构设计

1条评论

有话不说，憋着多难受啊!

请登录后才可以评论

评论

bedivere00 2017/12/12 09:15

相咨询一些问题，可以加个好友么

1楼 0次赞

热门技术

- Google Espresso 解耦重构 BG...

本文介绍了三种不同的思路来解决BGP出口流...

林潇 发表于17-09-29

2 2

网络测量是SDN发展的重要基础。作为网络测...

新型云基础设施项目Airship ...

AT&T正在与SK电信（SKT），Intel和OpenSta...

SD-WAN业界发展概述（文末附...

本报告不会特别评估任何一家SD-WAN公司的...

P4编程理论与实践（2）—快速...

本文的主要特色是让对P4感兴趣的大家不费...

最新评论

dsfdsf

openflow现在的最新版本是1.5吗？在哪里看它的最新资讯呢？

928917673

sdn .统一命令行。减少工作量。拍错定制不方便。 需要和sdn混合设计

宝中是圣地

wa 太感谢了！！！！！！！！！！一直卡在db.sock不存在，无法访问数据库，查了n*100+1的资料都不行 。但是通过你...

ღ_Continue梦^♛

@ycx19930209 您好，你的问题解决了吗？我现在遇到了相同的问题

林潇 发表于17-09-29

☆2 👍2

bingo4933

@咕噜噜 @咕噜噜 我是在这里：https://blog.csdn.net/chenhaifeng2016/articl做的实验，内容几...

zhouwaiqiang

@郁闷注册 在学校，有事直接发我邮箱吧857538065@qq.com

郁闷注册

请问学长现在还在学校吗，我也是北邮的学生，刚接触onos有点问题想请教下

glenn

@xxb249 a1 ping 不通 brB 的问题，试试看：...

glenn

@xxb249 a1 ping 不通 brB 的问题，试试看：ovs-ofctl add-flow brA "table=0,arp,arp_tpa=172.63.2...

glenn

a1 ping 不通 brB 的问题，试试看： ovs-ofctl add-flow brA "table=0,arp,arp_tpa=172.63.20.1,actions=ou

林潇 发表于17-09-29

2 2

林潇 发表于17-09-29