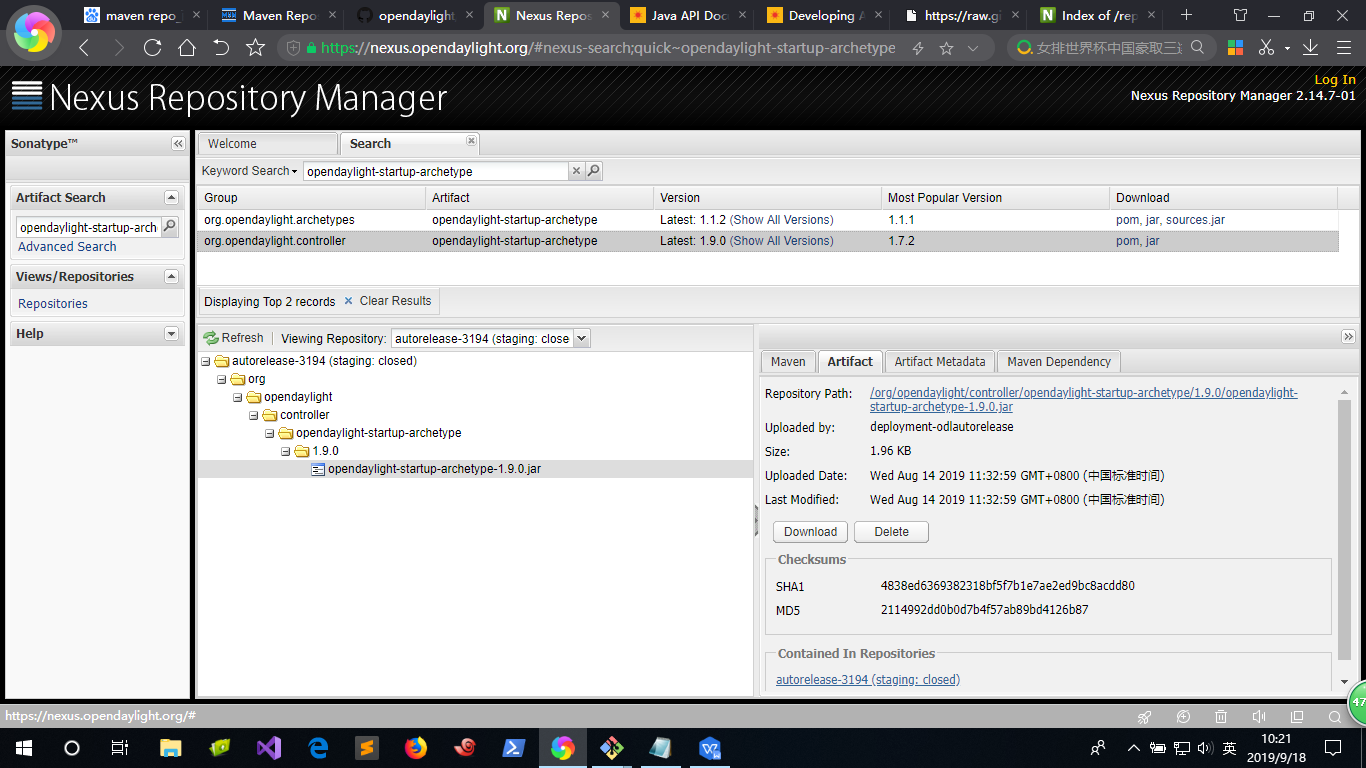
**ODL开发教程（Opendaylight）**

ODL（Opendaylight）控制器框架开发教程，ODL基于KARAF开源框架开发，而KARAF又是基于OSGI开源架构开发。

在开发ODL应用程序的时候，可以通过官方的骨架程序生成开发环境，通过生成的多层文件来开发程序。仓库网址：<https://nexus.opendaylight.org/#nexus-search>



通过MAVEN生成开发骨架：

mvn archetype:generate -DarchetypeGroupId=org.opendaylight.archetypes -DarchetypeArtifactId=opendaylight-startup-archetype -DarchetypeCatalog=remote -DarchetypeVersion=**<VERSION>**

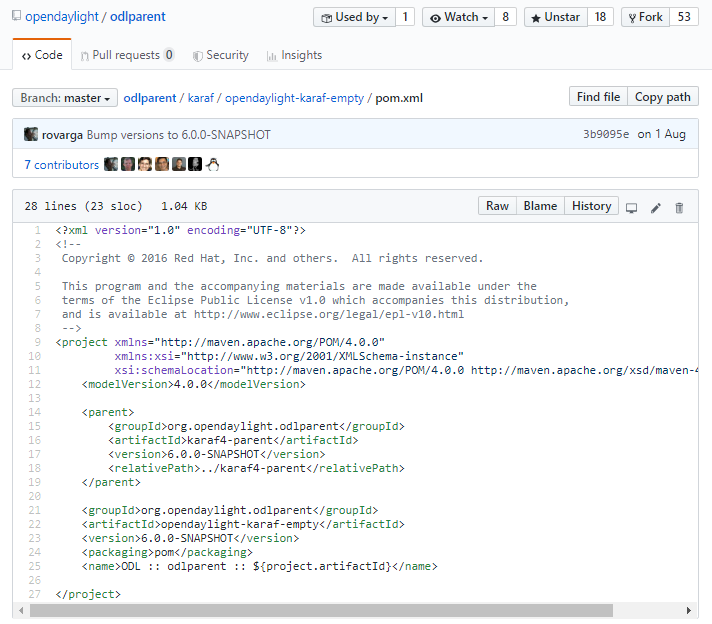
其中VERSION版本是可选择的，一般分为开发版本(SNAPSHOT)和稳定发行版本，以neon版本为例：

| **OpenDaylight Simultaneous Release** | **opendaylight-startup-archetype version** |
| --- | --- |
| Neon | 1.1.0 |
| Neon SR1 | 1.1.1 |
| Neon SR2 Development | 1.1.2-SNAPSHOT |

现在org.opendaylight.controller已经归并到org.opendaylight.archetypes，其中org.opendaylight.controller版本较高但是不包含一些开发目录，而org.opendaylight.archetypes版本低一些，但是不是对应相等关系。

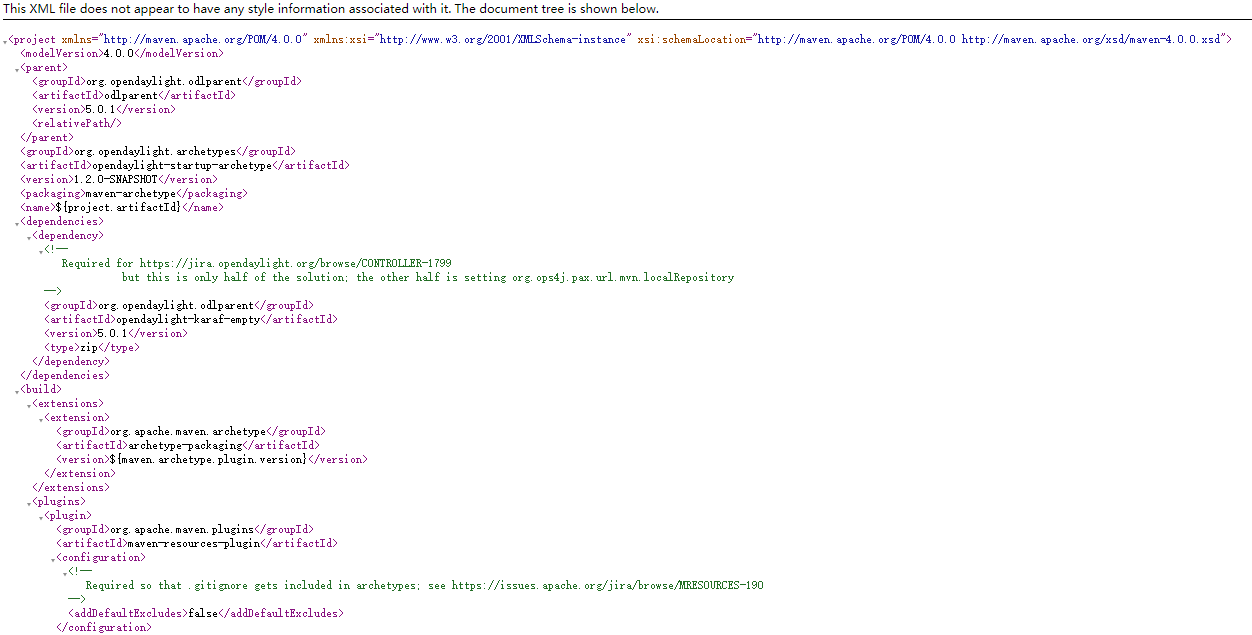
org.opendaylight.controller项目pom.xml文件：

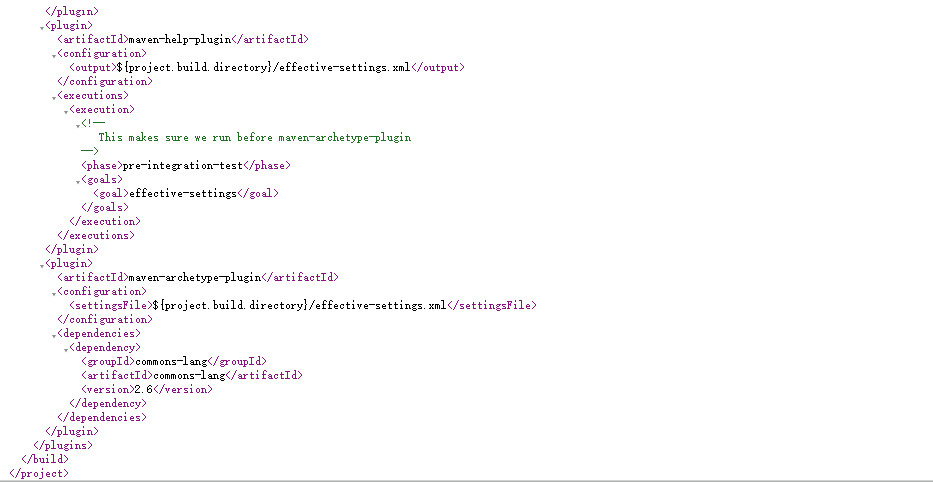
(https://github.com/opendaylight/odlparent/blob/master/karaf/opendaylight-karaf-empty/pom.xml)



org.opendaylight.archetypes项目pom.xml文件：

(https://nexus.opendaylight.org/service/local/repositories/opendaylight.snapshot/content/org/opendaylight/archetypes/opendaylight-startup-archetype/1.2.0-SNAPSHOT/opendaylight-startup-archetype-1.2.0-20190917.200053-108.pom)





<https://wiki.opendaylight.org/view/CrossProject:HouseKeeping_Best_Practices_Group:Project_layout>

示例：生成hello的骨架示例开发（以版本1.1.1为例）

## **Building an example module（编译一个示例模块）**

mvn archetype:generate -DarchetypeGroupId=org.opendaylight.archetypes -DarchetypeArtifactId=opendaylight-startup-archetype -DarchetypeCatalog=remote -DarchetypeVersion=1.1.1

## Update the properties values as follows. Ensure that the values for the groupId and the artifactId are in lower case.

Define value for property 'groupId': : org.opendaylight.example

Define value for property 'artifactId': : hello

Define value for property 'version': 1.0-SNAPSHOT: : 1.0.0-SNAPSHOT

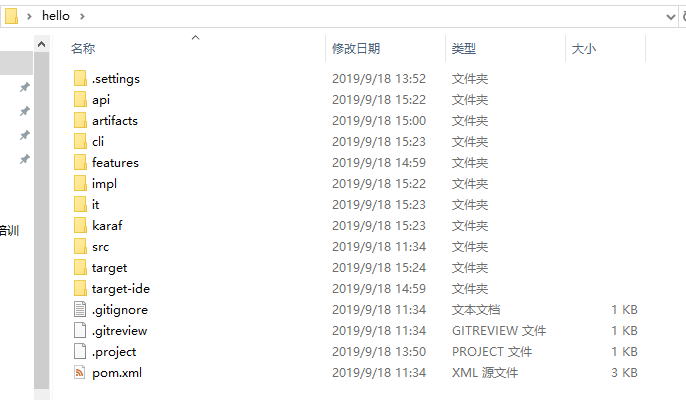
Define value for property 'package': org.opendaylight.example: :

Define value for property 'classPrefix': ${artifactId.substring(0,1).toUpperCase()}${artifactId.substring(1)}

Define value for property 'copyright': : Copyright (c) 2015 Yoyodyne, Inc.

Accept the default value of classPrefix that is,(${artifactId.substring(0,1).toUpperCase()}${artifactId.substring(1)}). The classPrefix creates a Java Class Prefix by capitalizing the first character of the artifactId.

**目录结构如下：**



**编译项目：**

mvn clean install

**启动运行：**

Start the example project for the first time.

cd karaf/target/assembly/binls./karaf

**日志查看：**

log:display | grep hello

**关闭运行：**

shutdown -f

## **Defining a Simple Hello World RPC(定义一个远程的Hello World远程调用)**

Build a hello example from the Maven archetype opendaylight-startup-archetype, same as above.

Now view the entry point to understand where the log line came from. The entry point is in the impl project:

impl/src/main/java/org/opendaylight/hello/impl/HelloProvider.java

Add any new things that you are doing in your implementation by using the HelloProvider.onSessionInitiate method. It’s analogous to an Activator.

**@Override**

Public void onSessionInitiated(ProviderContext session) {

LOG.info("HelloProvider Session Initiated");

}

## **Add a simple HelloWorld RPC API**

Navigate to the file.

Editapi/src/main/yang/hello.yang

Edit this file as follows. In the following example, we are adding the code in a YANG module to define the hello-world RPC:

module hello {

yang-version 1;

namespace "urn:opendaylight:params:xml:ns:yang:hello";

prefix "hello";

revision "2015-01-05" {

description "Initial revision of hello model";

}

rpc hello-world {

input {

leaf name {

type string;

}

}

output {

leaf greeting {

type string;

}

}

}}

Return to the hello/api directory and build your API as follows.

cd ../../../ mvn clean install

## **Implement the HelloWorld RPC API**

Define the HelloService, which is invoked through the hello-world API.

cd ../impl/src/main/java/org/opendaylight/hello/impl/

Create a new file called HelloWorldImpl.java and add in the code below.

package org.opendaylight.hello.impl;

import java.util.concurrent.Future;

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloService;

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldInput;

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldOutput;

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloWorldOutputBuilder;

import org.opendaylight.yangtools.yang.common.RpcResult;

import org.opendaylight.yangtools.yang.common.RpcResultBuilder;

public **class** HelloWorldImpl implements HelloService {

@Override

public Future<RpcResult<HelloWorldOutput>> helloWorld(HelloWorldInput input) {

HelloWorldOutputBuilder helloBuilder = new HelloWorldOutputBuilder();

helloBuilder.setGreeting("Hello " + input.getName());

return RpcResultBuilder.success(helloBuilder.build()).buildFuture();

}}

The HelloProvider.java file is in the current directory. Register the RPC that you created in the hello.yang file in the HelloProvider.java file. You can either edit the HelloProvider.java to match what is below or you can simple replace it with the code below.

/\*

\* Copyright(c) Yoyodyne, 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

\*/package org.opendaylight.hello.impl;

import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.ProviderContext;

import org.opendaylight.controller.sal.binding.api.BindingAwareBroker.RpcRegistration;

import org.opendaylight.controller.sal.binding.api.BindingAwareProvider;

import org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.hello.rev150105.HelloService;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;

public class HelloProvider implements BindingAwareProvider, AutoCloseable {

private static final Logger LOG = LoggerFactory.getLogger(HelloProvider.class);

private RpcRegistration<HelloService> helloService;

@Override

public void onSessionInitiated(ProviderContext session) {

LOG.info("HelloProvider Session Initiated");

helloService = session.addRpcImplementation(HelloService.class, new HelloWorldImpl());

}

@Override

public void close() throws Exception {

LOG.info("HelloProvider Closed");

if (helloService != null) {

helloService.close();

}

}}

Optionally, you can also build the Java classes which will register the new RPC. This is useful to test the edits you have made to HelloProvider.java and HelloWorldImpl.java.

cd ../../../../../../../mvn clean install

Return to the top level directory

cd ../

Build the entire hello again, which will pickup the changes you have made and build them into your project:

mvn clean install

## **Execute the hello project for the first time**

Run karaf

cd ../karaf/target/assembly/bin./karaf

Wait for the project to load completely. Then view the log to see the loaded Hello Module:

log:display | grep Hello

## **Test the hello-world RPC via REST**

There are a lot of ways to test your RPC. Following are some examples.

1. Using the API Explorer through HTTP
2. Using a browser REST client

### **Using the API Explorer through HTTP**

Navigate to [apidoc UI](http://localhost:8181/apidoc/explorer/index.html) with your web browser.

NOTE: In the URL mentioned above, Change localhost to the IP/Host name to reflect your development machine’s network address.

Select

hello(2015-01-05)

Select

POST /operations/hello:hello-world

Provide the required value.

{"hello:input": { "name":"Your Name"}}

Click the button.

Enter the username and password, by default the credentials are admin/admin.

In the response body you should see.

"output": {

"greeting":"Hello Your Name"

}}

### **Using a browser REST client**

For example, use the following information in the Firefox plugin RESTClient <https://github.com/chao/RESTClient>

POST: http://192.168.1.43:8181/restconf/operations/hello:hello-world

Header:

application/json

Body:

{"input": {

"name": "Andrew"

}}

## **Troubleshooting**

If you get a response code 501 while attempting to POST /operations/hello:hello-world, check the file: HelloProvider.java and make sure the helloService member is being set. By not invoking “session.addRpcImplementation()” the REST API will be unable to map /operations/hello:hello-world url to HelloWorldImpl.