

**DCFabric SDN控制器**

**开发文档**

**文档编号：**DCFabricSDN-Controller-DEV

**文档名称：**开源DCFabricSDN控制器开发文档

**版本信息：**V1.0.0

**建立日期：**2015年05月26日

**修改日期：**2017年05月06日

目录

[文档修订记录 3](#_Toc420418577)

[文档审批信息 4](#_Toc420418578)

[简介 5](#_Toc420418579)

[1.1 编写目的 5](#_Toc420418580)

[1.2 读者对象 5](#_Toc420418581)

[1.3 文档约定 5](#_Toc420418582)

[1.4 属于定义 5](#_Toc420418583)

[DCFabric-Controller开发 6](#_Toc420418584)

[一、 环境搭建 6](#_Toc420418585)

[二、 Makefile编写 8](#_Toc420418586)

[1． 增加新的.h目录 8](#_Toc420418587)

[2． 增加新的.c目录 8](#_Toc420418588)

[三、 开发实例 10](#_Toc420418589)

[1． 增加新的packet\_in消息处理 10](#_Toc420418590)

[2． 如何下发流表 11](#_Toc420418591)

[3． 如何响应Topo变化事件 13](#_Toc420418592)

# 文档修订记录

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **版本编号** | **修订原因** | **简要说明**  **（变更内容和变更范围）** | **日期** | **变更人** | **批准日期** | **批准人** |
| V1.0.0 | 创建 | 首次创建 | 2015-05-26 | 赵良智 |  |  |
| V1.0.1 | 修改 |  | 2017-05-8 | 杨才远 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**修订原因：**创建；增加；修改；删除

# 文档审批信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **审批人** | **角色** | **审批日期** | **签字** | **备注** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 简介

## 编写目的

本文旨在辅助开发人员通过内部二次开发SDN控制器，外部应用则可以通过北向接口来开发，具体参考北向接口文档。

开发分为两个部分，一个是Controller，另一个是WebApp。

Controller是控制器核心部分，基于OpenFlow协议的SDN控制器，对外提供控制REST接口。控制器采用标准C开发，基于Linux C/C++环境下编译。

WebApp是控制器界面，通过REST接口来和控制器进行交互。WebApp采用Java开发，不需要考虑编译环境。

## 读者对象

本文档预期读者包括：

* 数据中心SDN控制器开发项目管理和开发相关人员
* 系统需求组、系统架构组、系统设计组

## 文档约定

开发和编译环境均基于Linux 64位操作系统，调用的库函数均为Linux标准C/C++。本文上下文所描述的开发均基于此开发环境。其它编译环境，不保证能正确控制器软件能编译。

自主增加的功能模块必须按照约定俗成的模式，使用新的文件夹来隔离控制器的基本模块，从而偶合影响。

## 属于定义

|  |  |  |
| --- | --- | --- |
| **术语和缩写** | **全拼** | **解释** |
|  |  |  |
|  |  |  |

# DCFabric-Controller开发

## 环境搭建

* 1. 获取代码

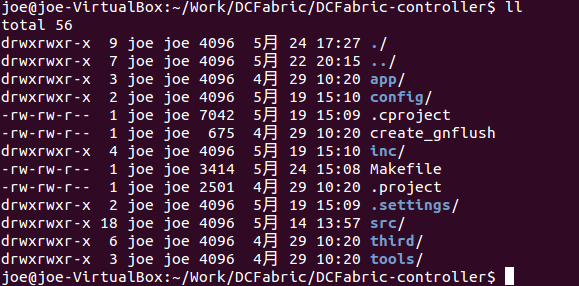
首先，从GitHub上获取到的控制器源码（注：需要有GitHub的权限才能下载）：

|  |
| --- |
| git clone https://github.com/China863SDN/DCFabric.git |



* 1. 查看目录

进入git下载目录，DCFabric，进入控制器源码目录DCFabric-controller，目录和文件结构和对应描述如下图示：



* **app：**拓展应用目录，开发人员可以在此目录实现DCFabric的拓展，暂为空；
* **inc：**控制器头文件目录，包含.h文件；
* **src：**控制器源文件目录，包含所有.c文件；
* **tools：**对接OpenStack工具
* **third：**第三方工具，包括以下目录:
* **redis：**hbase数据库相关头文件；
* **install-tools：**安装时需要用到的文件，如数据库建表脚本；
* **json-lib：**第三方json开发工具；
* **zookeeper：zookeeper头文件及静态库文件；**
  1. 编译DCFabric-controller源码

1. 进入到“DCFabric-controller”目录

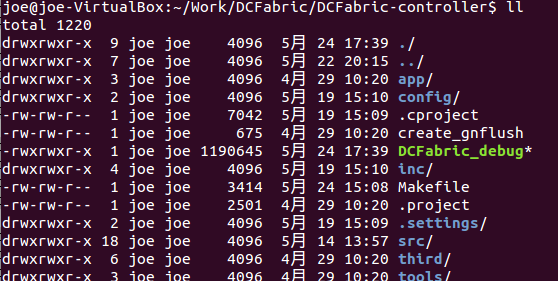


1. 执行编译

编译默认生成的是debug版本。如需生成完整版本执行如需命令编译：

|  |
| --- |
| make TYPE=release |

编译生成“DCFabric”可执行文件，如果是debug版本则名称为“DCFabric\_debug”：



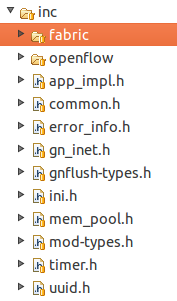
1. 至此版本编译完成。

## Makefile编写

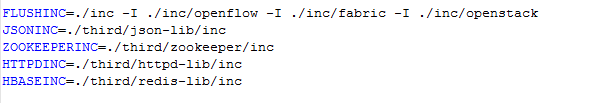
项目的makefile文件是生成控制器软件的编译脚本。增加新的源码文件时，makefile可以自动识别。

### 增加新的.h目录

一般来说，新增自定义模块最好把.h和.c分开放置，新的模块的.h文件都收录在一个文件夹中，文件夹位于inc目录下，这样便于模块分割查找。（以fabric模块为例）



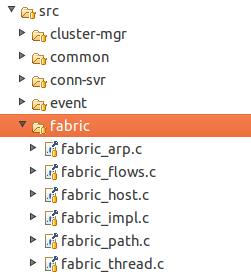
那么在Makefile文件中，必须加上新增的.h文件目录在FLUSHINC变量中：



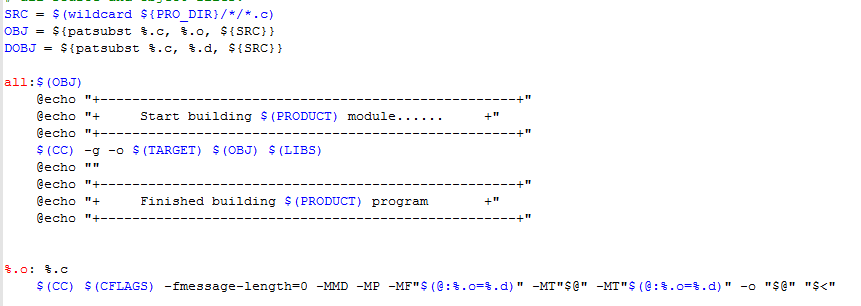
注：增加新的目录，必须make clean来重新编译，保证新的.h目录找到。

### 增加新的.c目录

同.h文件一样，一般自定义模块需要新建目录，放在src目录下，该模块的.c文件都收录在一个目录下。（以fabric模块为例）



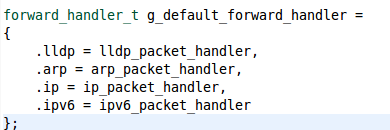
Makefile会自动识别src目录下的.c文件，并将.o文件同步生成到.c所在的文件夹目录



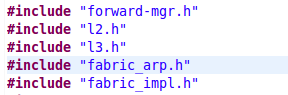
## 开发实例

### 增加新的packet\_in消息处理

packet\_in消息处理是在forward-mgr目录下的forward-mgr.c文件中处理的。当前版本已经对数据包做了类型封装处理，当前支持lldp、arp、ip、ipv6四种packet\_in消息。



要想新增加packet\_in消息处理，就必须导入相应模块的.h，引入对应的.h处理函数



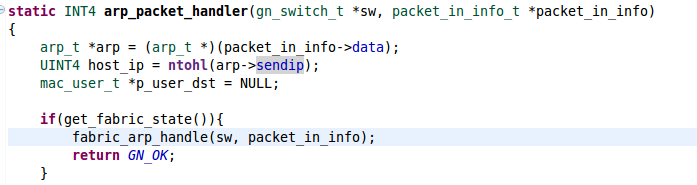
* 1. **arp处理函数**

函数名：arp\_packet\_handler

返回值：INT4

本函数是处理packet\_in上报的arp类型的数据包。如果要增加arp处理函数，就必须把自己的arp处理函数放入到这个函数内部，一般参数同主函数参数一致。

如果本模块arp处理结束还想交给其他模块的话，处理后不需要加返回值。

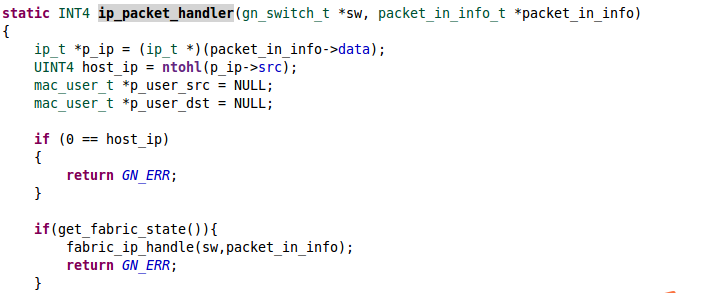


* 1. **ip处理函数**

函数名：ip\_packet\_handler

返回值：INT4

本函数是处理packet\_in上报的ip类型的数据包。要增加ip处理函数，就必须把自己的ip处理函数放入到这个函数内部，一般参数同主函数参数一致。本模块ip处理结束还想交给其他模块的话，处理后不需要加返回值。

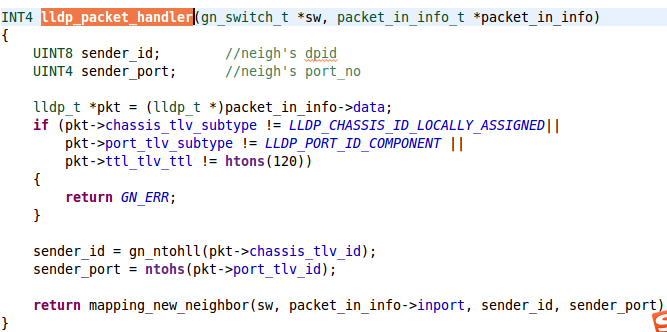


* 1. **lldp处理函数**

函数名：lldp\_packet\_handler

返回值：INT4

本函数是处理packet\_in上报的lldp类型的数据包。目的是为了发现相邻交换机。

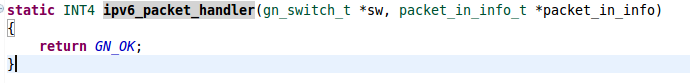


* 1. **ipv6处理函数**

函数名：ipv6\_packet\_handler

返回值：INT4

本函数是处理packet\_in上报的ipv6类型的数据包。暂时为空。



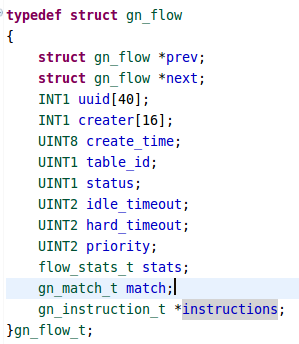
### 如何下发流表

当前，下发流表模块封装在flow-mgr.h中，通过导入.h文件中的函数来下发流表。对于自定义模块下发流表，建议自定义封装下发流表函数。

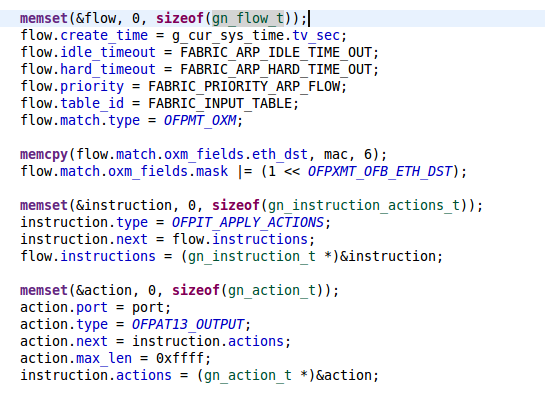
流表下发分为两个阶段，一个是流表组装阶段，一个是流表请求消息下发阶段。

* 1. **组装流表**

流表的定义在mod-types.h中

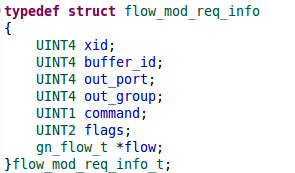


生成流表对象，填入如idle\_time，hard\_time，priority，table\_id等等流表信息。

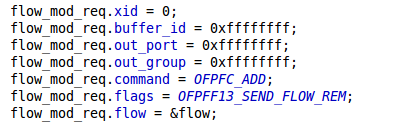


* 1. **生成流表消息**

流表消息定义在gnflush-types.h中



生成流表消息对象，主要是把流表对象放入到消息对象中。



* 1. **下发流表**

调用switch对象中的下发流表函数，完成流表下发动作

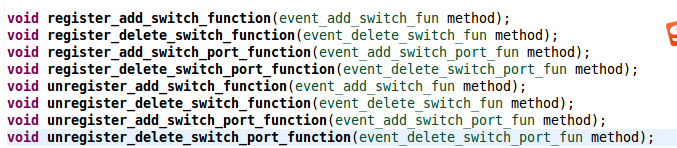


### 如何响应Topo变化事件

当Topo结构发生变化时候，自定义的模块可能要处理，当前系统提供了四个Topo变化的事件处理，分别是

* 增加交换机
* 删除交换机
* 增加交换机连线link消息
* 移除交换机连线link消息
  1. **注册Topo变化事件处理函数**

自定义模块要想处理这些消息，首先要把自己的处理函数注册到event模块中。注册函数在event\_service.h文件中提供。当自定义模块不需要处理Topo变化事件或者模块关闭时候，可以调用注销函数来把清除注册的函数。



* 注册函数都是register\_xxx\_function来命名
* 注销函数都是unregister\_xxx\_function来命名

处理函数必须符合事件处理函数规范，如果不符合规范，不清楚会出现何种异常情况。请使用者注意。

* 1. **增加交换机事件**

本事件是当新的交换机接入controller时候触发。函数格式为



参数为：

gn\_switch\_t\*\* swList //增加的sw对象数组

UINT4 num //增加的sw对象的数量

返回值为void

**例子：**

**注册**



**事件处理函数**



* 1. **移除交换机事件**

本事件是当已有交换机断开和controller时候触发。函数格式为



参数为：

gn\_switch\_t\*\* swList //删除的sw对象数组

UINT4 num //删除的sw对象的数量

返回值为void

**例子：**

**注册**



**事件处理函数**



* 1. **增加交换机连线link消息**

本事件是交换机增加新的link信息时候触发。函数格式为



参数为：

p\_event\_sw\_port sw\_portList //增加的link的端口信息数组

UINT4 num //增加的link的端口的数量

返回值为void

**例子：**

**注册**



**事件处理函数**



* 1. **移除交换机连线link消息**

本事件是交换机移除已有的link信息时候触发。函数格式为



参数为：

p\_event\_sw\_port sw\_portList //移除的link的端口信息数组

UINT4 num //移除的link的端口的数量

返回值为void

**例子：**

**注册**



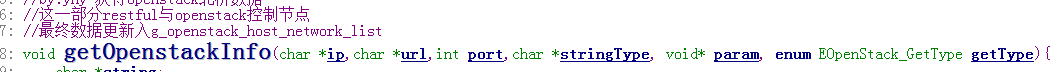
**事件处理函数**

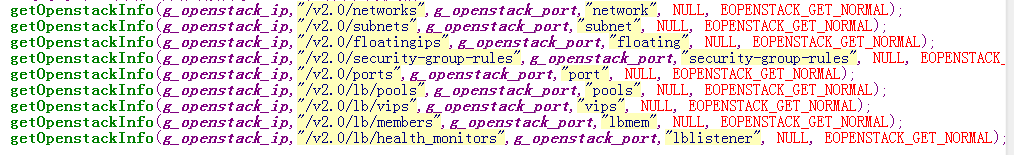


### 如何同步openstack变化

当openstack页面内容发生变化时，控制器定时主动连接openstack服务器，发送数据请求包，并按照openstack API V2.0标准格式解析响应报文。

**4.1 发送请求**

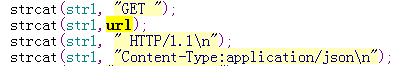




参数:

ip: openstack服务器ip地址

url: 向openstack服务器请求服务类型，填充在发送字段里



port：openstack服务器监听端口

stringType：标识向服务器发起的请求服务类型

Param： 空

getType：区分标识，仅用于区分端口转发和其他业务

**4.2 响应解析**





按照openstck API对应的格式，解析json数据，提取待用信息，构建新的数据结构保存，或者刷新原有数据。



