1. **起步** 
   1. **开发需要的知识**

本项目采用python Flask为基础进行后台开发，前端框架主要采用了Angularjs和Bootstrap。要进行本项目开发，需要具有HTML/CSS，Javascript，Python语言的知识，同时也需要了解Angularjs和Bootstrap以及Python的Flask框架

* 1. **源码目录结构说明**

|+api/ API的具体实现

|+base/ 后端基本的模块，如用户验证，表单验证等

|+doc/ 相关开发文档

|+log/ 开发时产生的日志

|~static/ Falsk静态文件目录

| `~switch/ WEB前端页面文件夹

| |+css/ 所有CSS所在目录

| |+images/ 所有图片资源

| |~js/ js文件目录，大部分为angular控制器

| | |+home/ 交换机WEB页面的控制器

| | |+i18n/ 为页面提供多国语言支持

| | |-base64.js javascript的base64编码实现

| | |-components.js 自定义angular 组件

| | |-config.js (核心文件)前端的配置文件

| | |-directives.js 自定义angular指令

| | |-filter.js 自定义angular过滤器

| | |-header.js 页头的控制器

| | |-login.js 登录页面的控制器

| | |-nav.js 页面左侧导航栏的控制器

| | |-services.js 自定义angular服务

| | `-url.js 自定义angular路由

| |~templates/ HTML模板目录

| | |+home/ 结果类似于js/home，存放大部分页面

| | |-footer.html 页脚

| | |-header.html 页头

| | |-login.html 登录页面

| | `-nav.html 导航栏页面

| |+third/ 第三方前端库，如angular，bootstrap

| `-index.html 交换机WEB的首页

|+urls/ 后台API的路由

|+vcl/ virtual command line，主要为了方便开发，vcl

| 为本地和远程执行cmd提供了相同的接口

|-index.py 用于普通PC上启动WEB服务器

|-pkg\_web.sh 编译镜像时自动打包WEB目录的脚本

|-run\_server.py 用于在交换机上启动WEB服务器

注意：在开发过程中考虑到CGI通信缓慢，我们并没有吧Flask部署到任何服务器(如nginx)，这可能会带来一些稳定性上的问题。在以后的版本中这个问题会得到改善。

* 1. **整个WEB是如何工作的**
     1. **启动服务器**

在vcl/vcmd.py里，ins = Remote() if config.is\_dev else Local()会根据is\_dev判断ins是属于Remote还是Local的实例。is\_dev在vcl/config.py里定义，在vcl/\_\_init\_\_.py导入时被赋值。其中Remote()和Local()对外提供了同样的接口，他们的作用是执行shell命令。这样通过vcmd里的接口就可以在PC上或者交换机上执行shell命令，达到远程开发的目的，省去需要在交换机上编写代码的麻烦。

在base/log.py里，会判断base/config.py里的log\_en，如果为True则创建对应的log文件。需要注意的是，在交换机上启动WEB服务器时，log\_en会自动设置为False，否则log太多占用资源。

* + 1. **访问首页**

在WEB后台：base/web\_app.py里有这样一条路由：

@app.route('/')

@auth.login\_required

def index():

return redirect(url\_for('static', filename='switch/index.html'))

访问**/**时，匹配到该条路由，进入auth.login\_required装饰器，该装饰器会检测用户是否登录，如果没有登录则重定向**/static/switch/index.html#/home/login**，如果用户已经登录，则进入index()函数，重定向到**/static/switch/index.html**。

在switch/js/config.js文件里，定义好了每条前端路由所对应的控制器和html模板，然后在switch/js/url.js文件添加到angular路由系统中，对于所有不存在的前端路由，会重定向到首页**/static/switch/index.html#/home/index**

注意：在之后的介绍中，因为该项目为SPA(single page application)，为了书写简洁，省略掉/static/switch/index.html，前端页面统一写为#/home/index, #/home/login之类的形式，另外前端目录一致写为/switch/…

* + 1. **登录**

登录验证相关文件为base/auth.py,本项目中所有需要登录验证的路由前需要添加auth.login\_required装饰器，并且采用token方式验证用户。

用户未登录时，首先需要**POST：/api/token/**，提交usernmae和password，获取token，token为一个随机的md5字符串。从获取token开始，若用户未操作间隔大于10分钟(默认10分钟，可在base/config.py里配置)，则token自动失效，只能重新获取。

对于前端请求auth.login\_required的url，分为两种情况，一种为GET方式，另外一种为PUT/POST/DELETE方式，对于**GET**方式，因为参数只能在url中，所以token也需要定义在url中提交到后台，如**/api/vlan/?token=\*\*\*\*\*\***,对于PUT/POST/DELETE，token可以放到表单当中一起提交，当然，提交的表单必须为json。

如{token: \*\*\*\*\*\*\*, ……}。

如果token没有验证通过，会返回error\_auth = true,否则为false。

* + 1. **API**

对于后台，所有API相关的路由在/urls/\*.py里定义，API的路由均以**/api**开头，比如**/api/vlan**。匹配到相应API路由后，会调用/api/\*.py里的函数处理请求并返回相应的json。

对于前端，API的请求采用AJAX方式，在angular里名叫$http的服务封装了AJAX的整套操作。但是，大多数API需要用户登录才能操作，使用$http需要我们自己添加token和处理验证错误的操作。所以在/switch/js/services.js里我们提供了两个关于http的服务：SecretHttp和SecretApi，SecretHttp在$http的基础上自动加上了token(token存放在localStroage中)，SecretApi在SecretHttp的基础上封装了错误处理，使得我们在开发的采用SecretApi只需要关心API请求成功之后该做什么。

既然SecretApi做好了错误处理，那么后台必须有个错误的返回标准，这个标准是：所有请求必须返回三个参数：error, err\_code, err\_reason

error为bool值

err\_code 为整数，0代表没有错误，1以上代表相应的错误代码(即使本项目中暂时没用到)

err\_reason为字符串，这是对错误的一个详细说明，优点是前端可以通过err\_reason直接套用错误信息，缺点是不方便做国际化，因为err\_reason只能携带一种语言。

* + 1. **访问页面**

在访问首页中提到过，前端路由都在/switch/js/config.js中定义。要访问任意定义好的页面，比如拿系统管理>文件管理举个例子，只要链接到**/static/switch/index.html#/home/sys\_mgr/file\_mgr**即可。

对于文件管理这个路由，控制器位于/switch/js/home/sys\_mgr/file\_mgr.js，html模板位于/switch/templates/home/sys\_mgr/file\_mgr.html，可以看出来，文件路径和路由结构很相似。

1. **开发** 
   1. **如何开发一个新页面**
      1. **结构**

一个新页面的开发分为两个部分，第一部分是页面的开发，即前端，另一部分是API的开发，即后台。为了达到解耦的效果，前后端采用json交互。接下来将演示开发一个简单的查看文件功能，以此来熟悉开发流程。

* + 1. **设计API接口**

查看文件，采用的是HTTP的**GET**方式，除此之外，常用的还有**POST，DELETE，PUT**，可以分别对应于新增，删除，修改。这里我们只采用GET方式获取文件内容。

另外，获取文件还需要有个路径，这个路径可以放在URL当中，所以API接口设计成这样：**GET: /api/file /<path\_base64>/**

其中形如<xxx>的是Falsk框架的路由参数，不了解的参考Falsk文档。至于base64，因为路径有特殊字符，直接放进URL太敏感，所以转换为base64即可。

* + 1. **添加Flask路由**

到url文件夹下新增filemgr.py文件，添加如下内容

from flask import jsonify

import base, api

@base.web\_app.app.route('/api/file/<path\_base64>/', methods=['GET'])

@base.web\_app.auth.login\_required

def file\_R(path\_base64):

resp = api.filemgr.get(path\_base64)

return jsonify(resp)

以上代码的作用为对于**/api/file/<path\_base64>/**匹配的请求交给file\_R函数处理，在该函数里调用了api下的filemgr模块里的get函数，并且吧返回的值包装为json回应给客户端。

注意login\_required装饰器，这说明了以上url是需要登录才能访问的

别忘了在url/\_\_init\_\_.py里导入filemgr.py

* + 1. **编写API**

到api文件夹下添加filemgr.py文件，添加内容

from vcl import vcmd

import base64

def get(path\_base64):

\_err\_reason = [

'', # err\_code: 0

'cmd error', # err\_code: 1

]

obj = {'content': '', 'error': False, 'err\_code': 0, 'err\_reason': ''}

path = base64.decodestring(path\_base64)

status, lines = vcmd.get\_status\_lines('cat %s' % path)

if status != 0:

obj['error'] = True

obj['err\_code'] = 1

obj['err\_reason'] = '; '.join(lines)

return obj

obj['content'] = '\n'.join(lines)

return obj

注意error,err\_code,err\_reason，基本所有的API都会返回这三个参数。另外别忘了导入vcmd使得你在PC端开发的时候得到在交换机上执行cmd同样的结果。

这里的status的值是$0，也就是cmd执行的状态，如果cmd执行错误会返回一个非零值。如果cmd执行成功，文件内容会被放入content里。

为了方便演示，这里并没有做参数检测，实际开发的时候需要在解码base64的时候抓取异常。

别忘了在api/\_\_init\_\_.py里导入filemgr.py

* + 1. **设计前端路由**

类似于一般的URL，这给了用户一个访问某页面的链接。这里设计为

**#/home/filemgr**

* + 1. **添加html(模板)**

添加/switch/templates/home/filemgr.html,编写代码

<div class="col-md-6">

<form class="form">

<div class="form-group">

<label>file path:</label>

<input class="form-control" type="text" ng-model="filePath"/>

</div>

<button class="btn btn-primary" ng-click="showFile()">show</button>

</form>

<textarea rows="30" ng-model="fileContent" class="form-control">

</textarea>

</div>

这里使用了angular，输入框绑定到了filePath变量，textarea绑定到了fileContent变量。Show按钮绑定到了showFile()函数。

* + 1. **添加js(控制器)**

添加/switch/js/home/filemgr.js,编写代码

Public.app.controller('ctlHomeFilemgr',['$scope','SecretApi',function($scope, SecretApi) {

$scope.filePath = '';

$scope.fileContent = '';

$scope.showFile = function() {

$scope.fileContent = '';

var pathBase64 = BASE64.encoder($scope.filePath);

SecretApi({

url: '/api/file/'+pathBase64+'/',

method: 'GET',

}, function(data) {

$scope.fileContent = data.content;

});

};

}]);

本项目中只有一个angular APP，在config.js里已经赋值给了Public.app中。

这里控制器的名称为ctrlHomeFilemgr，待会在添加路由还会用到。

注意SecretApi,这个前面已经提到过了，它对错误处理做了封装，并且为每个请求添加token。

至于js中编码BASE64，在网上都能找到源码，jQuery也有对应的库。

* + 1. **引入js，添加路由**

到/switch/index.html中，引入js文件

<script type="text/javascript" src = "./js/home/filemgr.js"></script>

到/switch/js/config.js中，添加filemgr的路由，格式为：[路由，控制器，html路径]

['/home/filemgr', 'ctlHomeFilemgr', './templates/home/filemgr.html'],

* + 1. **将链接添加到导航栏**

路由添加好了，现在可以输入对应地址访问了，但是为了用户方便使用，需要吧该地址放到一个导航链接里提供用户点击。

到/switch/templates/nav.html中，添加对应链接并设定css

<li ng-class="{'active': isAcitve('/home/filemgr')}">

<a href="#/home/filemgr">

<span class="rect"></span>&nbsp;filemgr

</a>

</li>

代码中设置了三个地方，一个为css，当匹配到**/home/filemgr**路由的时候，**li**的css被设置为active

另一个为**a**标签的href，最后为该导航的名称。

最后开启服务器，访问首页，左侧导航如图：



* + 1. **测试**

例如查看/etc/hosts,结果如下



文本框显示正确， 一个页面开发完成。

1. **前端自定义API** 
   1. **自定义服务**

源码参考文件/switch/js/services.js

* + 1. **SecretHttp**

原型：function(options)

这个之前提到过很多次了，可以把它当成$http一样中，options参数和返回值与$http一模一样，只是SecretHttp默认加入了token而已。

* + 1. **SecretApi**

原型：function(options, fnSuccess, fnBefore, fnAfter)

包装SecretHttp，做了错误处理。

fnSuccess：

会在请求成功并且error=false时调用。

fnBefore：

会在请求完成(不管成功与否)时调用。

fnAfter：

会在请求完成(不管成功与否),并且完成错误显示或者执行fnSuccess之后调用。

* + 1. **Range2Array**

原型：function(sRange)

sRange：

string类型, 形如'1,3-6,10'

返回值：

array

例：Range2Array('1,3-6,10') -> [1,3,4,5,6,10]

* + 1. **Array2Range**

原型：function(arr)

arr：

array类型, 形如[1,3,4,5,6,10]

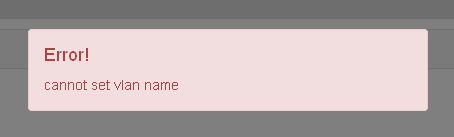
返回值：

String

例：Array2Range([1,3,4,5,6,10]) -> '1,3-6,10'

* + 1. **Alert**

类似于js里的alert，封装了一个弹出框，默认3秒消失，如图



原型：function(options)

Options：string | object

string: 只传入字符串则默认作为danger的内容

object: {

type: 'danger' | 'waring' | 'success' | 'info'，默认danger

title: string，显示的标题，默认'Error!'

content: string，要显示的内容

time: int,单位秒，设置几秒后消失，默认为3秒，如果为0，则不会自动消失。

}

返回值：{hide: function}

会返回一个对象，对象里有一个hide函数，如果之前设置time为0，那么调用hide则可以手动关闭Alert。

例：

Alert('bad request')

Alert({

type: 'success',

title: 'Success',

content: 'Delete vlan successfully'

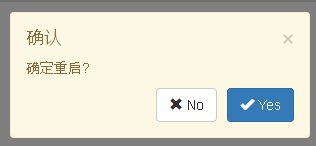
time: 5,

})

注意：Alert不可嵌套，因为页面只有一个Alert元素定义于header.html

* + 1. **Confirm**

提供一个弹框，有yes和no两个按钮给用户选择，如图



原型：function(content, fnYes, fnNo)

Content:

String，提示的内容

fnYes:

function，用户点击yes后调用该函数

fnNo:

function，用户点击no后调用该函数

例：

Confirm('Are you sure delete vlan 1?', function() {

console.log('yes');

}, function() {

console.log('no');

})

注意：Confirm不可嵌套，因为页面只有一个Confirm元素定义于header.html

* 1. **自定义过滤器**

源码参考文件/switch/js/filter.js

* + 1. **fsize**

原型：function(input)

Filesize的简写,传入一个整数，吧该数整形为形如100.712M,200B的形式。保留三位小数。

例：{{ '11670' | fsize }} -> '113.740K'

* + 1. **switchPort**

原型：function(input)

主要针对拆分口起作用。

例：{{ 'eth-0-10\_1' | swithPort }} -> 'eth-0-10/1'

* 1. **自定义组件和指令**

源码参考文件/switch/js/directives.js和/switch/js/components.js

组件和指令用法差不多，只不过组件在angular1.5版本后新加入的。

* + 1. **btnSwitch**

生成一个开关，如图：



属性：

switch-state:

bool, 可以和变量绑定

switch-text-on:

string, 开关开启时的文本，默认on

switch-text-off:

string, 开关关闭时的文本，默认off

switch-on-change:

function, 当开关状态改变时，调用该函数

switch-disabled:

bool, 设置是否禁用开关

例：

<btn-switch

switch-state="state[index]"

switch-on-change="changeState(index)"

switch-disabled="idDisable[index]">

</btn-switch>

* + 1. **pagination**

生成分页按钮，由于只是生成按钮，所以制作分页需要配合angular的limitTo过滤器使用，效果如图：



属性：

pg-item-cnt:

int, 所有条目数量

pg-item-num-each-page:

int, 每页显示的条目数量，默认为10条

pg-current:

int, 当前位于第几页

pg-max-btn-cnt:

int, 限制最多同时产生多少个分页按钮，默认9个

pg-on-change:

function, 当用户点击分页按钮或者pg-current产生变化时，调用该函数

pg-home-text:

srting, 首页按钮上的文本，默认为'home'

pg-last-text:

string, 尾页按钮上的文本，默认为'end'

例：

<pagination

pg-current="pgCurrent"

pg-item-cnt="pgItemCnt"

pg-item-num-each-page="pgItemNumEachPage"

pg-on-change="onPageChange()">

</pagination>

1. **后台自定义API** 
   1. **后台验证**

目前后台提供dict和array的验证，源码和例子参考/base/check.py

验证dict采用base.check.dictionary(dict\_data, arr)，返回值为一个元祖，如果检测通过，返回(dict, None)，否则(None, {‘err\_code’: int, ‘err\_reason’: string})，返回dict会吧参数整形，例如如果检测类型为int，但是请求的是字符串的数字，那么返回的dict中会吧字符串的数字转换为int。

其中arr中可以放多个数组，每个数组检测dict\_data中的一个值，数组的形式如

[$key, $type, $opt1, $opt\_val1, $relation1, $opt2, $opt\_val2, $relation2 ..., $err\_reason, $err\_code]

$key：dict\_data中的key

$type: 检测的类型，可以为’int’|’string’|’array’|’dict’

$err\_reason和$err\_code必须为数组最后两个元素，如果检测不通过，那么对应的$err\_reason和$err\_code会被返回。

$opt, $opt\_val, $relation可以不设置，也可以设置多个，例如下面两行是一样的效果：都是检测vlan是否为1或2或3

[['vlan', 'int', '>=', 1, 'and', '<=', 3, 'vlan error', 1]]

[['vlan', 'int', '==', 1, 'or', '==', 2, 'or', '==', 3, 'vlan error', 1]]

一个完整的例子：

dictionary({'ip': '192.168.1.1', 'vlan': 4094, 'type': 'dynamic', 'pwd': 'abcd1234'}, [

['ip', 'string', 're', r'^(\d{1,3}\.){3}\d{1,3}$', 'bad request: ip', 1],

['vlan', 'int', '>=', 2, 'and', '<', 4095, 'bad request: vlan', 1],

['type', 'string', 'in', ['static', 'dynamic'], 'bad request: type', 1],

['pwd', 'string', '>=', 6, 'and', '<=', 10, 'bad password', 1],

])

array的验证形如dict，只不过array不需要设置key，而且数组里没有嵌套数组，因为array里一般数据都是同一类型。函数原型：

array(list\_data, arr, filter=False)

例如

array(

['eth-0-1', 'eth-0-2', '\_\_fdsfa\_\_'],

['string', 're', r'^[a-zA-Z0-9\-]{3,15}$', 'bad request', 1],

True)

Array和dict不同的是，多了第三个参数filter，默认为False，即array里所有元素必须匹配，一旦有不匹配的返回错误。如果设置为True，那么并不会返回错误，而是过滤掉不匹配的元素。

* 1. **后台数据类型转换**

在python里进行数据转时，类型或数值错误就会抛异常。为了阻止异常，在base/parse.py里编写了parse\_json, parse\_int, parse\_string三个函数

其中parse\_string一般不会出异常，重点在于parse\_json和parse\_int。

parse\_json和parse\_int可以传入两个参数，第一个为要转换的数据，第二个为转换错误时的默认值。parse\_json错误默认返回{},parse\_int错误默认返回None

例：

parse\_int('1') -> 1

parse\_int('1x') -> 1

parse\_int('x') -> None

parse\_int('x', 0) -> 0

1. **FAQ** 
   1. **如何国际化**

语言的翻译在/switch/js/i18n下，其中英文可以不用单独翻译，因为英文可以作为key。

例如在/switch/js/i18n/zh\_CN.js中添加

'home': '首页'

那么在html中显示home的时候，采用

{{ 'home' | translate }}即可

* 1. **参考手册**
     1. **angularjs**

<https://angularjs.org>

* + 1. **bootstrap**

<http://getbootstrap.com>

* + 1. **flask**

<http://flask.pocoo.org>

* + 1. **underscore.js**

<http://underscorejs.org>