1.

Flask（<http://flask.pocoo.org/docs/1.0/>）是Python的另一个Web开发框架，并且是一个微型的开发框架。Flask的主旨是让Web框架的核心尽量的简单。Flask本身不像Django本身含有，表单验证，数据库抽象层等功能。这些功能Flask交给了第三方扩展插件来提供。Flask通过这些插件来让人感觉Flask本身实现了这些功能。

2.

Flask本身主要含有两个组件，Werkzeug组件以及Jinja组件。Wekzeug组件用于处理WSGI的应用，处理HTTP请求以及响应。而Jinja组件则和Django一样，用于模板(template)生成。

3.

Flask是BSD协议，意义为使用者可以自由的使用，修改源代码，也可以将修改后的代码作为开源或者专有软件再发布，在必要的时候进行二次开发。

4.

默认情况下，Flask将模板文件放在templates文件夹下，静态文件 (JS, 图片等) 放在/static文件夹下。

5.

为了在处理多个Python 项目的时候，各个python版本不互相影响。Flask鼓励使用python自身所带的virtualenv进行开发处理。Python3自带venv模块，而对于Python2来说，需要自行安装virtualenv模块。在进行项目开发以及安装Flask的时候，我们可以先进去virtualenv环境，然后在这个环境里面，安装的任何python包都只会对这个环境有影响，不会影响到外界。

6.

python – m 的作用是将一个Python模块作为Python脚本运行。Python xx.py（直接运行） 与Python -m xx.py（模块脚本方式运行）是Python启动的两种不同的加载方式. 他们的区别在于sys.path在脚本运行中的值。

Sys.path的值在直接运行时是脚本所在目录，以及脚本所在目录的目录以及Python自带路径。

Sys.path的值在模块脚本方式运行时是当前命令行所在路径，以及当前命令行所在路径的目录以及Python自带路径。

7，我们使用pip来安装Flask, pip命令自身会将Flask所需要的依赖包都安装上。Flask的其中一个依赖包是Click, Click的作用是提供了flask可以被命令行识别，并且添加自定义管理flask的命令。

8.

from flask import Flask # 引入Flask类

app = Flask(\_\_name\_\_) # 创建一个Flask类的实例app, 这个Flask实例用于处理WSGI任务

@app.route**(**'/'**) #**告诉这个函数应该处理什么URL，使用route修饰器

**def** hello\_world**(): #**函数名

**return** 'Hello, World! #返回什么HTTP内容

9.

Flask还可以从URL中获得参数，并且接下来的处理函数中该参数作为命名参数传入函数当中。

@app.route**(**'/user/<username>'**)** #默认认为是string类型

**def** show\_user\_profile**(**username**):** #传入命名参数为username的参数

*# show the user profile for that user*

**return** 'User %s' % username

@app.route**(**'/post/<int:post\_id>'**)** #传入类型为int的整数

**def** show\_post**(**post\_id**):** #传入命名参数为post\_id的参数

*# show the post with the given id, the id is an integer*

**return** 'Post %d' % post\_id

|  |  |
| --- | --- |
| string | (default) accepts any text without a slash  默认为字符串，但是不接受/ |
| int | accepts positive integers |
| float | accepts positive floating-point values |
| path | like string but also accepts slashes  和string相比，接受/ |
| uuid | accepts UUID strings |

10.

Flask在Debug Mode启动的时候，如果代码有所改变，则Flask会自动提取变化，无需再手动重启。

11.

默认情况下定义的函数只会处理HTTP的GET请求，除非加上methods参数methods=**[**'GET'**,** 'POST']，那么此函数还会响应POST请求。

@app.route**(**'/login'**,** methods=**[**'GET'**,** 'POST'**])**

12.

url\_for()函数是Flask自带的反向构建URL函数,它的目的就是构建一个URL地址。第一个函数是endpoint，也就时响应HTTP请求的函数名，然后是URL地址中的命名参数。

使用url\_for()去反向的解析出url地址的好处是：

--url\_for()可以对硬性编码更加有描述性，因为写入的是函数的名称。

--url\_for()对传入的参数的值，如果含有空格等特殊字符的时候，会自动转义成%20，而不需要硬性编码直接写成%20

--url\_for() 的时候，如果我们在view函数中改变分配的路径路由，那么url\_for()映射出的url地址也会动态改变。而不需要再一次到模板中更改硬性编码的地址。

url\_for()可以在模板中，也可以在view函数中使用。

例如：

url\_for('profile', username='John Doe') -----》 反向产生：/user/John%20Doe

同时我们也可以反向产生静态文件的URL地址：

url\_for('static', filename='style.css') --------》产生/static/style.css

13.

[render\_template()](http://flask.pocoo.org/docs/1.0/api/#flask.render_template) 方法是Flask用Jinja2引擎生成HTML文件的函数。其中指定要进行渲染的HTML文件以及传入该文件的数据。

14.

Flask的HTTP Request Object,

我们可以通过Request.form 来获取POST或者PUT请求中的数据。

我们可以通过Request.args来获取url中? key=value 的数据。

同时Flask建议使用Python中所带的get方法来访问字典中的数据，而不是直接通过中括号来访问。这样可以避免访问数据时，出现异常，用户接受到500的HTTP响应。

15.

对Flask组建的网站上传文件十分简单。首先在HTML文件form标签中，写入enctype="multipart/form-data"，然后在响应函数中使用request.files来获取文件句柄，然后调用该文件句柄的save () 方法（save方法可以指定保存在服务器上的位置以及文件名），便可以保存在服务器上。

16.

Redirect（重定向的地址）函数，用于返回一个响应，将用户重定向到另一个页面。

17.

Flask对HTTP响应的改写。首先服务器最终要给浏览器返回一个HTTP Response。我们在Flask中有时候写出的view函数时最后只是return一个字符串，并没有直接返回一个HTTP Response响应。Flask会根据一套规则，将我们从view中返回的数据进行包装，变成一个HTTP Response对象然后再返回给浏览器。这个过程时由Flask自动完成的。

同时，Flask也允许我们自行构建HTTP Response请求 （即使用make\_response函数），然后直接返回给浏览器。如果我们直接操作HTTP Response时，Flask不会再次加工而是直接返回。例如：

@app.errorhandler(404)

def not\_found(error):

resp = make\_response(render\_template('error.html'), 404) #使用make\_response函数来构建HTTP Response，并且在接下来自行改写

resp.headers['X-Something'] = 'A value'

return resp #现在返回的是HTTP Response，所以Flask不会进行自动转化。

18.

Flask从release版本0.3开始，自带有logging模块。其中有三个level, app.logger.debug

与app.logger.warning以及app.logger.error.这个logging模块不是Flask开发出来的，而是python库中标准自带的logger包。

19.

Flask-Bootstrap与Bootstrap-Flask是针对Flask的两个Bootstrap插件扩展。Bootstrap-Flask比Flask-Bootstrap更加轻量化更好。

他们的流程一般是在Flask App创建完实例以后，对其进一步的wrap,然后便可以在代码中提供额外的标签以及功能。

20.

我们在开发Flask的时候，希望Flask处于Debug模式之下，这样在网页上可以显示更多的Debug信息。

export FLASK\_ENV=development的时候，Flask自动处于开发模式下，此时 DEBUG模式会被默认激活。

export FLASK\_ENV=production的时候，Flask处于生产模式下，此时DEBUG模式不会被激活。

另外我们可以专门设置FLASK\_DEBUG=1/0去激活或者禁用DEBUG状态。

21.

我们有两种方法可以启动Flask.

第一种是直接到.py所在的目录，使用python xxx.py进行启动。

第二种是设置环境变量export flask\_app = “路径/xxxx.py”,告诉flask框架应用的.py文件在哪。然后在服务器上的任何地方进行flask run便可以启动flask框架。

22.

在Flask的url匹配上，可以进行多个url匹配都对应同一个函数。对于可能的存在或者缺失的命名参数，初值为None. 例如；

@app.route('/setupContents/<string:targetfile>')

@app.route('/setupContents/')

@app.route("/")

def index(targetfile=None):

23.Flask也可以自定义templatetags.自定义的filter可以在app上注册也可以在blueprint上注册。

例如：

@app.template\_filter()

def pretty\_date(dttm):

return dttm.strftime("%m/%d")

然后在模板中便可以使用。

或者在blueprint上注册：

my\_blueprint = Blueprint('blue\_name', \_\_name\_\_, template\_folder='templates')

@my\_blueprint.app\_template\_filter()

def pretty\_date(dttm):

return dttm.strftime("%m/%d")

24.

Flask蓝图 (Blueprint)的用途是给Flask创建一个子应用模块存在与一个子目录中。在这个蓝图子目录下，存在自己的模板以及URL匹配。最后Flask所指定的主程序app中，进行注册这个子应用，也就是蓝图就可以扩展，响应更多的URL。同时，在主程序中如果不注册使用，便相当于卸载掉了这个子应用程序。

Flask蓝图 (Blueprint)的工作方式很像Flask的APP实例，但又不是app实例。并且Flask利用蓝图来扩展App实例的功能。

25.

from flask import g

g在Flask中是一个特殊的对象，面对每个独立的http请求。在多个函数处理通过一个http请求时，可以将共享的数据放入这个特殊的对象中。

26.

Flask运行的时候只能有一个APP实例，但是有可以有很多blueprint实例。这些blueprint的实例用于挂载在APP上给与URL匹配扩充。

27.Url地址中协议和域名部分是不区分大小写的，但是路径部分则根据服务器平台而定。原因在于server的主机。windows系统服务器不区分大小写，但是在Linux以及类Linux主机上认为大小写是俩个不同的文件路径。

28.

我们在注册blueprint的时候，即是app.register\_blueprint(xxx, url\_prefix='/pages')中，有可选参数url\_prefix.这个可选参数url\_prefix如果不填，那么相当于直接扩充APP上的url匹配。如果填写，那么等于带有url\_prefix的那些url才会到这个蓝图中。

29.

***View functions, error handlers, and other functions that run during a request will have access to the***[***request***](http://flask.pocoo.org/docs/1.0/api/#flask.request)***proxy, which points to the request object for the current request.***

在一次Flask的view函数处理请求的过程中，不需要传入request对象，Flask会主动产生。并且如果这个view函数调用其它任何函数，那么在调用其他任何函数的时候也不需要显示的传request对象，其他函数在运行中可以直接访问到request对象。

30.

Flask直接使用的是Jinja2 Html模板语言，而Django的模板语言采用的是与Jinja2相似，但又不是完全是Jinja2的语言。

31.

@app.route**(**'/projects/'**)**

**def** projects **():**

**return** 'The project page'

@app.route**(**'/about'**)**

**def** about **():**

**return** 'The about page'

在Flask注册url的时候：

如果在注册的时候带有结尾的/，那么用户访问界面的时候，如果结尾带有/，则访问成功。如果用户输入的url不带有/,那么Flask会自动添加然后访问成功。

如果在注册的时候不带有结尾的/,那么用户访问界面的时候，如果结尾带有/，则访问不成功返回404错误代码。如果不带有/，则访问成功。

32.

@app.route**(**'/submitForm/', methods = [“GET”, “POST”]**)**

**def** submitForm **():**

**……………**

在Flask中，请求地址submitForm的时候，由于在路由配置中带有结尾的/， 所以我们访问/submitForm时候，Flask会帮我们自动自动添加，可以访问。 但是如果我们向/submitForm提交POST请求的时候，同样的Flask也会帮我们加入尾部的/访问的到。但是默认情况下，自动添加尾部相当于redirect, Flask将会拒绝经过redirect的POST请求。所以对于GET请求，我们访问地址时，可以添加尾部/也可以不添加。但是如果进行POST请求，必须要与路由定义一摸一样添加尾部的/

33.

r = requests.post ("http://megatron.labs.ca.alcatel-lucent.com:5000/HashGenerate/", data=json.dumps({‘content”:”AAA”}),headers={“content-type“： “application/json”})

这样向Flask发送post请求过程中，我们使用data来传递数据，headers定义了content-type为json.所以传到Flask后端的时候可以使用request.json来获取content的值。如果我们不加入这个头headers= {“content-type “： “application/json”} ， 在Flask默认情况下是不能使用request.json来获取的（request.json将是None,因为Flask并不知道这个Json格式的数据）。

res = requests.post('http://localhost:5000/api/add\_message/1234', json={"mytext":"lalala"})

这样向Flask发送post请求过程中，我们使用json关键字来发送数据，虽然没有通过headers来定义content-type为json， 但是json关键字将指定content-type为application/json.所以在后台中还是可以通过request.json来获取数据。