## 简介

本节实验开始的实验环境进行了一些优化，因此第一周保存的环境不能继续使用。若你保存了第一周的环境，请重新选择默认在线实验环境。

Flask 是 Python 社区比较知名的微框架。Flask 被设计成可以通过插件拓展，Flask 本身只维护一个核心。如果要和 Django 做对比的话，Django 更像一个大品牌的出的电脑整机，你不用操心使用什么配件，你需要什么 Django 全家桶都有。而 Flask 可以说是一个组装机了，更准确的说是一个设计精良的 CPU。这给了你很大的灵活性去选择需要的配件（插件）。

### 知识点

* Flask 简介
* 配置方法
* 注册路由
* 模板渲染
* request 对象
* session
* cookies
* 错误处理
* 插件

## 简单的例子

一个最简单的例子，创建 app.py，写入：

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def index():

return 'Hello World!'

if \_\_name\_\_ == '\_\_main\_\_':

app.run()

Flask 提供了一个管理 Flask 应用的命令行工具，首先要设置应用的环境变量：

export FLASK\_APP=app.py

export FLASK\_DEBUG=1

环境变量 FLASK\_APP 是用来指向 flask run 执行的 Flask 应用代码的路径，这里是 app.py 的路径。FLASK\_DEBUG=1 表示打开 DEBUG 信息，可以输出访问和出错信息，帮助我们解决代码中出现的问题，建议后续只要执行 flask run 都要打开 FLASK\_DEBUG。

然后就可以这样运行 Flask 应用了：

flask run

默认地，应用运行在 localhost:5000 上。打开浏览器，访问这个地址就能看到返回的 Hello World! 了。

Flask 简单示例开发视频：

## flask shell

除了 flask run 之外，还有一个常用的命令是 flask shell，这两个命令都会自动把 FLASK\_APP 环境变量中指定的代码模块进行加载，不同的是 flask run 直接进入到运行 app 的状态，而 flask shell 只加载并进入到一个 Shell 终端，在这个终端中可以执行一些代码，比如后续章节中要用到的初始化数据库，向数据库中插入一些数据等。

$ flask shell

>

## 配置

初始化一个 Flask app 后，可以通过 app.config 管理配置。app.config 存储的配置信息本质上是个字典，所以你可以用字典的方法添加或者更新配置。比如说，初始化 app 后，配置一个密钥：

app = Flask(\_\_name\_\_)

app.config.update({

'SECRET\_KEY': 'a random string'

})

所有的配置选项需要用大写，多个单词间用下划线 \_连接。大型项目中，配置通常写在一个单独的 config.py 文件中，这时候就可以用 app.config 提供的特有方法来更新 config，参数是配置文件 config.py 的路径：

app.config.from\_pyfile('path/to/config.py')

其他类似的方法：

* from\_envvar(variable\_name)：使用一个环境变量指定的配置文件更新配置
* from\_object(obj)：使用一个对象更新配置文件，dict 无效
* from\_json(filename)：使用 JSON 文件更新配置
* from\_mapping(\*mapping, \*\*kwargs)：类似前面的 update，不同的是，这个方法不强制使用大写字母

获得一个配置信息的方法是用字典的形式 app.config['SECRET\_KEY'] 这样可以获得 SECRET\_KEY 的配置值。

## 注册路由

Flask 使用 @app.route 装饰器来注册路由及其处理函数。在上面的例子中，就用主页 / 注册一个路由，访问主页 Flask 会用 index 函数去处理。

可以在路由中传入变量，格式为 <variable\_name>，比如每个用户的主页需要不同的路由，可以使用用户名作为路由的变量：

@app.route('/user/<username>')

def user\_index(username):

# 在函数中指名变量名称，这样就能获取到通过路由传入的变量值

return 'Hello {}'.format(username)

也可以指定路由变量的类型，比如说，一个博客应用的每个博文页面可以用这篇博文的 ID 作为路由变量，ID 应该是个 int 类型的值：

@app.route('/post/<int:post\_id>')

def show\_post(post\_id):

return 'Post {}'.format(post\_id)

注册的路由 return 的内容会包含在返回给用户的 HTTP response 中，这两个实例都返回字符串，所以用户使用浏览器访问这两个链接地址的时候看到的就是两个字符串显示在浏览器页面上。

## 模板渲染

在上面的例子中，处理函数返回的都是字符串，但是在真正的项目中，需要使用 HTML 编写页面，不可能把所有的内容都写到字符串中。模板引擎的作用就是你用模板引擎规定的语法编写 HTML 页面，在处理函数中指定模板，传入相应的模板变量，Flask 就能调用模板引擎自动渲染出一个完整的 HTML 页面。

Flask 默认的模板引擎是 jinja2，理论上你是可以更换其它模板引擎的，但是 jinja2 已经足够好用。

Flask 使用 render\_template 函数渲染模板，指定了一个模板名称后，Flask 会到 templates 目录下去找这个模板，然后使用传入的变量渲染模板。

如果我们用模板来改写用户主页的例子，那么处理函数可以这样写：

from flask import render\_template

@app.route('/user/<username>')

def user\_index(username):

return render\_template('user\_index.html', username=username)

然后创建 templates 目录，目录结构变成这样：

/flask-test-app

app.py

/templates

user\_index.html

在 user\_index.html 中：

<h1>Hello, {{ username }}!</h1>

在 jinja2 中，用 {{ }} 来渲染一个字符串变量。这里的 username 就是在 render\_template 的时候传入的关键字参数 username。现在访问一个用户主页，比如说：

localhost:5000/user/shiyanlou

就能看到一个用 h1 标签包裹的 Hello, shiyanlou! 了。

另外，flask 中提供了 url\_for 来根据路由的函数名称构建 URL 链接地址，提供了 redirect 来跳转到其他路由，见下面的例子：

from flask import render\_template, redirect, url\_for

@app.route('/')

def index():

return redirect(url\_for('user\_index', username='default'))

@app.route('/user/<username>')

def user\_index(username):

return render\_template('user\_index.html', username=username)

这个例子中，访问 / index 页面的时候，会自动跳转到 /user/default 页面，跳转的目标页面 URL 地址是由 url\_for 产生的，而跳转过程是由 redirect 函数进行的操作。

Flask 模板渲染示例开发视频：

## request 对象

Flask 通过 request 对象获取请求相关的数据，要使用它，要从 flask 导入：

from flask import request

从 request.headers 获取请求头的数据，可以把它当作字典来使用，比如要获取用户的 user-agent：

request.headers.get('User-Agent')

从 request.args 获取请求的参数，假设我们的应用是个博客应用，主页有分页功能，用 这个 URL 访问主页：

localhost:5000?page=2&per\_page=10

获取 ? 后面的参数就可以这样做：

page = request.args.get('page')

per\_page = request.args.get('per\_page')

除此之外，可以通过 request.form 获取表单数据，通过 request.method 获取当前请求的方法（GET 或 POST）。

## session

HTTP 协议是无状态的，每一次请求它们的关系都是相互独立的。但是在实际的应用是，我们确实有很多数据服务器需要记住，但又不适合存放在数据库中。

比如说，一个登录页面需要在用户密码输入错误 3 次后要求输入验证码，服务器就需要一个计数器纪录错误次数，但把它放到数据库也不太合适。session 就是用来为每个用户独立存放一些数据的地方。存放在 session 里的数据可以在特定用户的多个请求之间共享。

## cookies

cookies与 session 类似，只不过 cookies 是存在客户端加密信息。在 Flask 中，cookie 使用配置的 SECRET\_KEY 作为签名进行加密。

比如在上面的访问用户主页的路由中，将用户名设置为一个 cookies，这样用户在此访问时，我们就能知道他是谁了：

from flask import make\_response

@app.route('/user/<username>')

def user\_index(username):

resp = make\_response(render\_template('user\_index.html', username=username))

resp.set\_cookie('username', username)

return resp

设置 cookies后，用户访问其他页面可以从 request.cookies 获取到我们设置的 username：

from flask import request

@app.route('/')

def index():

username = request.cookies.get('username')

return 'Hello {}'.format(username)

Flask cookies 示例开发视频：

## 错误处理

使用 app.errorhandler 装饰器可以注册错误处理函数，比如对于最常见的 404 错误，我们返回一个特定的 404.html 页面。

@app.errorhandler(404)

def not\_found(error):

return render\_template('404.html'), 404

例子中也展示了使用 render\_template 的一个小知识点，就是可以在它后面指定本次返回的状态码。

在 flask 中还有一个经常用来处理错误的方法 abort()，使用 abort(404) 则能够直接进入到页面无法找到（HTTP 状态码404）的处理逻辑中。例子如下：

from flask import render\_template, abort

@app.route('/user/<username>')

def user\_index(username):

if username == 'invalid':

abort(404)

return render\_template('user\_index.html', username=username)

当 username 为 invalid 字符串的时候，即访问 /user/invalid 地址的时候，直接返回页面无法找到。

Flask 错误处理示例开发视频：

## 插件

下面列出了一些 Flask 开发中常用的插件，这些插件大多在后面的项目中会用到，因为篇幅的原因不能一一介绍。每个插件都可以在 Github 搜到，大家可以先去了解一下。

* flask\_sqlalchemy：ORM，封装了 sqlalchemy，使用更简单
* flask-login：管理用户 session，如登入、登出，session 过期管理等等
* flask\_migrate：数据库版本管理
* flask\_wtf：封装了 wtforms 表单生成与验证工具，提供了 CSRF 支持
* flask-session：flask 默认 session 基于客户端 cookie 的，这个插件方便在服务端做 session