Jinja模板

《Web应用开发》 / 任课教师：罗志一

计算机科学与技术学院

浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

概览 Overview

- Flask与HTTP（续）
  - 请求钩子，Cookie，Session，Flask上下文
- 模板引擎：Jinja2
- 大作业：问答平台

Flask与HTTP（续）

请求钩子

- 有时候我们需要对请求进行预处理和后处理，这是可以使用Flask提供的一些**请求钩子**，它们可以用来**注册**在请求处理的不同阶段执行的处理函数（或称为**回调函数**）。
- 请求钩子使用装饰器实现，通过程序实例app调用。

钩子	说明
before_first_request	注册一个函数，在处理第一个请求前运行
before_request	注册一个函数，在处理每个请求前运行
after_request	注册一个函数，如果没有未处理的异常抛出，会在每个请求结束后运行
teardown_request	注册一个函数，即使有未处理的异常抛出，会在每个请求结束后运行。如果发生异常，会传入异常对象作为参数到注册的函数中。
after_this_request	在视图函数内注册一个函数，会在这个请求结束后运行

请求钩子

- 这些钩子使用起来和app.route()装饰器基本相同，每个钩子可以注册任意多个处理函数，函数名并不是必须和钩子名称相同。例如：

```
@app.before_request
def do_something():
    pass   # 这里的代码会在每个请求处理前执行
```

来一块Cookie

- HTTP是无状态协议。也就是说，再一次请求响应结束后，服务器不会留下任何关于对方状态的信息。但是对于某些Web程序来说，客户端的某些信息又必须被记住，比如用户的登录状态，这样才可以根据用户的状态来返回不同的响应。为了解决这些问题，就有了Cookie技术。
- Cookie技术通过在请求和响应报文中添加Cookie数据来保存客户端的状态信息。

来一块Cookie

- Cookie指Web服务器为了存储某些数据（比如用户信息）而保存在浏览器上的小型文本数据。
- 浏览器会在一定时间内保存它，并在下一次向同一个服务器发送请求时附带这些数据。Cookie通常被用来进行用户会话管理（比如登录状态），保存用户的个性化信息数据（比如语言偏好，视频上次播放的位置，网站主题选项等）以及记录和收集用户浏览数据以用来分析用户行为等。

来一块Cookie

```
# get name value from query string and cookie
@app.route('/')
@app.route('/hello')
def hello():
    name = request.args.get('name')
    if name is None:
        name = request.cookies.get('name', 'Human')
    response = '<h1>Hello, %s!</h1>' % escape(name)  # escape name to avoid XSS
    # return different response according to the user's authentication status
    if 'logged_in' in session:
        response += '[Authenticated]'
    else:
        response += '[Not Authenticated]'
    return response
```

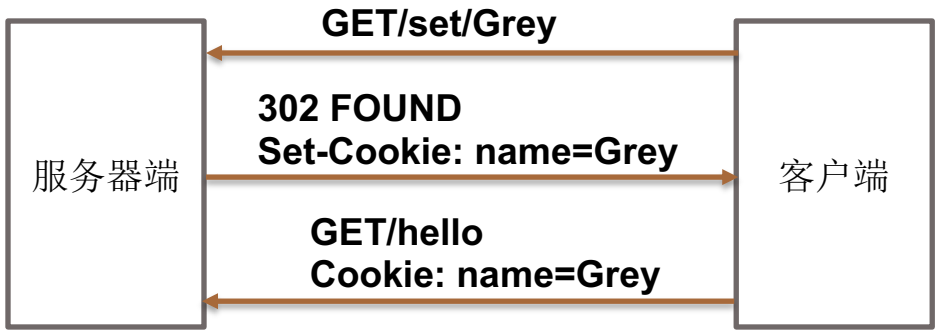
来一块Cookie

- 在Flask中，如果想要在响应中添加一个cookie，最方便的方法是使用Response类提供的set_cookie()方法。
  - 使用make_response()方法手动生成一个响应对象，传入响应主体作为参数。这个响应对象默认实例化内置的Response类。
  - 使用set_cookie()方法为响应对象设置Cookie的选项。
  - 下面的set_cookie视图会在生成的响应报文首部中创建一个Set-Cookie字段，即“Set-Cookie: name=Grey”

```
from flask import Flask, make_response
@app.route('/set/<name>')
def set_cookie(name):
    response = make_response(redirect(url_for('hello')))
    response.set_cookie('name', name)
    return response
```

来一块Cookie

- 当浏览器保存了服务器端设置的Cookie后，浏览器再次发送到该服务器的请求会自动携带设置的Cookie信息，Cookie的内容存储再请求首部的Cookie字段中。



session: 安全的Cookie

- Cookie在Web程序中发挥了很大的作用，其中最重要的功能是存储用户的认证信息。
  - 在登录表单填写用户名、密码，点击登录按钮向服务器发送一个包含认证数据的请求；服务器接受请求后会查找对应的账户，然后验证密码是否匹配，如果匹配就在返回的响应中设置一个cookie，比如：“login_user: greyli”。
  - 问题：在浏览器中手动添加和修改Cookie是很容易的事情，如果把认证信息以明文的方式存储在Cookie里，那么恶意用户就可以通过伪造cookie的内容来获得对网站的权限，冒用别人的账户。
  - 我们需要对敏感的Cookie内容进行加密。Flask提供了session对象用来将Cookie数据加密存储。

session: 安全的Cookie

- 设置程序密钥
  - session通过密钥对数据进行签名以加密数据。

1. 写到代码文件

```
app.secret_key="Drmhze6EPev0fN_81Bj-nA”
```

1. pip install python-dotenv
2. 创建.env文件，写入：

```
SECRET_KEY=Drmhze6EPev0fN_81Bj-nA
```

3. 在命令行中运行：flask run
4. import os

```
app.secret_key = os.getenv("SECRET_KEY")
```

- 模拟用户认证
  - session对象把数据存储在浏览器上一个名为session的cookie里。

```
from flask import redirect, url_for, session
# log in user
@app.route('/login')
def login():
    session['logged_in'] = True  # 写入session
    return redirect(url_for('hello'))
```

session: 安全的Cookie

- 使用session对象存储的Cookie，用户可以看到其加密后的值，但无法修改它。因为session中的内容使用密钥进行签名，一旦数据被修改签名的值也会变化。这样在读取时，就会验证失败，对应的session值也会随之失败。
- 除非用户知道密钥，否则无法对session cookie的值进行修改。

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 为了方便的获取这两种上下文环境中存储的信息，Flask提供了四个上下文全局变量：

变量名	上下文类别	说明
current_app	程序上下文	指向处理请求的当前程序实例
g	程序上下文	替代Python的全局变量用法，确保仅在当前请求中可用。用于存储全局数据，每次请求都会重设。
request	请求上下文	封装客户端发出的请求报文数据
session	请求上下文	用于记住请求之间的数据，通过签名的Cookie实现

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 上下文“全局”变量
  - 每一个视图函数都需要上下文信息。例如，在前面我们学习过Flask将请求报文封装在request对象中。
  - 动态全局：Flask会在每个请求产生后自动激活当前请求的上下文，request被临时设为全局可访问。而当每个请求结束后，Flask就销毁对应的请求上下文。
  - Flask通过本地线程技术将请求对象在特定的线程和请求中全局可访问。

request请求上下文

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 为了方便的获取这两种上下文环境中存储的信息，Flask提供了四个上下文全局变量：

变量名	上下文类别	说明
current_app	程序上下文	指向处理请求的当前程序实例
g	程序上下文	替代Python的全局变量用法，确保仅在当前请求中可用。用于存储全局数据，每次请求都会重设。
request	请求上下文	封装客户端发出的请求报文数据
session	请求上下文	用于记住请求之间的数据，通过签名的Cookie实现

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 上下文“全局”变量
  - 既然有了程序实例app对象，为什么还需要current_app变量？

current_app程序上下文

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 上下文“全局”变量
  - 既然有了程序实例app对象，为什么还需要current_app变量？
  - 在不同的视图函数中，request对象都表示和视图函数对应的请求，也就是当前请求。而程序也会有多个程序实例的情况，为了能获取对应的程序实例，而不是固定的某一个程序实例，我们就需要使用current_app变量。

current_app程序上下文

Flask上下文

- 我们可以把编程中的上下文理解为当前环境的快照。
- 有两种上下文：程序上下文和请求上下文。
- 上下文“全局”变量
  - g存储在程序上下文中，随着每一个请求的进入而激活，随着每一个请求的处理完毕而销毁，所以每次请求都会重设这个值。我们通常会使用它结合请求钩子来保存每个请求处理前所需要的全局变量，比如当前登入的用户对象，数据库连接等。

g程序上下文

Flask上下文

- 当请求进入时，Flask会自动激活请求上下文，这时我们可以使用request和session变量。另外当请求上下文被激活时，程序上下文也被自动激活。当请求处理完毕后，请求上下文和程序上下文也会自动销毁。
- 这也就意味着，我们可以在视图函数中或者在视图函数内调用的函数或方法中使用所有上下文全局变量。
- 如果在没有激活相关上下文时使用这些变量，Flask就会抛出RuntimeError异常。

模板引擎：Jinja2

模板引擎：Jinja2

- 在我们第一个“hello_world”示例中，当用户访问程序的根地址时，我们的视图函数会向客户端返回一行HTML代码。然而，一个完整的HTML页面往往需要几十行甚至上百行代码，如果都写到视图函数里，既不简洁也不利于维护。
- 正确的做法是把HTML代码存储在单独的文件中，以便让程序的业务逻辑和表现逻辑分离，即控制器和用户界面的分离。

模板引擎：Jinja2

- 在动态Web程序中，视图函数返回的HTML数据往往需要根据相应的变量（比如查询参数）动态生成。当HTML代码保存到单独的文件中使用时，我们没法在使用字符串格式化或拼接字符串的方式来在HTML中插入变量，这时我们需要使用模板引擎（template engine）。
- 借助模板引擎，我们可以在HTML文件中使用特殊的语法来标记出变量，这类包含固定内容和动态部分的可重用文件称为模板。

模板引擎：Jinja2

- 为什么要叫Jinja
  - 之所以叫Jinja，是因为日本的神社（Jinja）英文单词是temple，而模板的英文是template，两者发音很相似

模板引擎：Jinja2

- 模板渲染
- Flask默认使用的模板引擎是Jinja2，它是一个功能齐全的Python模板引擎，除了设置变量，还允许我们在模板中添加if判断，执行for迭代，调用函数等。

```
{% block body %}
  <ul>
  {% for user in users %}
    <li><a href="{{ user.url }}">{{ user.username }}</a></li>
  {% endfor %}
  </ul>
{% endblock %}
```

模板引擎：Jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>{{ user.username }}'s Watchlist</title>
</head>
<body>
<a href="{{ url_for('index') }}">&larr; Return</a>
<h2>{{ user.username }}</h2>
{% if user.bio %}
    <i>{{ user.bio }}</i>
{% else %}
    <i>This user has not provided a bio.</i>
{% endif %}
{# Below is the movie list (this is comment) #}
<h5>{{ user.username }}'s Watchlist ({{ movies|length }}):</h5>
<ul>
    {% for movie in movies %}
    <li>{{ movie.name }} - {{ movie.year }}</li>
    {% endfor %}
</ul>
</body>
</html>
```

watchlist.html

我们可以在模板中使用Python语句和表达式来操作数据的输出，但Jinja2并不支持所有Python语法。

模板引擎：Jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>{{ user.username }}'s Watchlist</title>
</head>
<body>
<a href="{{ url_for('index') }}">&larr; Return</a>
<h2>{{ user.username }}</h2>
{% if user.bio %}
    <i>{{ user.bio }}</i>
{% else %}
    <i>This user has not provided a bio.</i>
{% endif %}
{# Below is the movie list (this is comment) #}
<h5>{{ user.username }}'s Watchlist ({{ movies|length }}):</h5>
<ul>
    {% for movie in movies %}
    <li>{{ movie.name }} - {{ movie.year }}</li>
    {% endfor %}
</ul>
</body>
</html>
```

语句：比如if判断、for循环

{% … %}

模板引擎：Jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>{{ user.username }}'s Watchlist</title>
</head>
<body>
<a href="{{ url_for('index') }}">&larr; Return</a>
<h2>{{ user.username }}</h2>
{% if user.bio %}
    <i>{{ user.bio }}</i>
{% else %}
    <i>This user has not provided a bio.</i>
{% endif %}
{# Below is the movie list (this is comment) #}
<h5>{{ user.username }}'s Watchlist ({{ movies|length }}):</h5>
<ul>
    {% for movie in movies %}
    <li>{{ movie.name }} - {{ movie.year }}</li>
    {% endfor %}
</ul>
</body>
</html>
```

表达式：比如字符串、变量、函数调用等；
{{ … }}

模板引擎：Jinja2

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>{{ user.username }}'s Watchlist</title>
</head>
<body>
<a href="{{ url_for('index') }}">&larr; Return</a>
<h2>{{ user.username }}</h2>
{% if user.bio %}
    <i>{{ user.bio }}</i>
{% else %}
    <i>This user has not provided a bio.</i>
{% endif %}
{# Below is the movie list (this is comment) #}
<h5>{{ user.username }}'s Watchlist ({{ movies|length }}):</h5>
<ul>
    {% for movie in movies %}
    <li>{{ movie.name }} - {{ movie.year }}</li>
    {% endfor %}
</ul>
</body>
</html>
```

注释
{# ... #}

渲染模板

- 渲染一个模板，就是执行模板中的代码，并传入所有在模板中使用的变量，渲染后的结果就是我们要返回给客户端的HTML响应。
- 在视图函数中渲染模板时，我们并不直接使用Jinja2提供的函数，而是使用Flask提供的渲染函数render_template()。
- 先在app.py中定义user, movies变量

```
user = {
    'username': 'Grey Li',
    'bio': 'A boy who loves movies and music.',
}
movies = [
    {'name': 'My Neighbor Totoro', 'year': '1988'},
    {'name': 'Three Colours trilogy', 'year': '1993’}, …]
```

渲染模板

- 渲染一个模板，就是执行模板中的代码，并传入所有在模板中使用的变量，渲染后的结果就是我们要返回给客户端的HTML响应。
- 在视图函数中渲染模板时，我们并不直接使用Jinja2提供的函数，而是使用Flask提供的渲染函数render_template()。
- 先在app.py中定义user, movies变量

```
from flask import render_template
@app.route('/watchlist')
def watchlist():
    return render_template('watchlist.html', user=user, movies=movies)
```

大作业：问答平台

问答平台

- 开发一个问答平台网站，自行进行前端页面设计，后端技术不限，需包含如下功能：
  - 基于检索的问答功能
    - 关键词全文检索（必做）
    - 相似度匹配检索（可选）

问答平台

- 开发一个问答平台网站，自行进行前端页面设计，后端技术不限，需包含如下功能：
  - 用户登录（必做）
  - 登录之后可以发布问答
    - 发布问题（必做）
    - 回答问题（必做）
    - 记录时间、点赞数（选做）

问答平台

- 开发一个问答平台网站，自行进行前端页面设计，后端技术不限，需包含如下功能：
  - 充分利用已有数据，设计其他功能（选做）
    - 如：关键词做标签对问答分类等