Flask入门系列

项目开发中，经常要写一些小系统来辅助，比如监控系统，配置系统等等。用传统的Java写，太笨重了，连PHP都嫌麻烦。一直在寻找一个轻量级的后台框架，学习成本低，维护简单。发现Flask后，我立马被它的轻巧所吸引，它充分发挥了Python语言的优雅和轻便，连Django这样强大的框架在它面前都觉得繁琐。可以说简单就是美。这里我们不讨论到底哪个框架语言更好，只是从简单这个角度出发，Flask绝对是佼佼者。这一系列文章就会给大家展示Flask的轻巧之美。

1. Flask入门系列(一)-快速上手
2. Flask入门系列(二)-路由
3. Flask入门系列(三)-模板
4. Flask入门系列(四)-请求，响应及会话
5. Flask入门系列(五)-错误处理及消息闪现]
6. Flask入门系列(六)-数据库集成

Flask入门系列(一)–快速上手

程序员的经典学习方法，从Hello World开始。不要忘了，先安装python, pip，然后运行`pip install Flask`，环境就装好了。当然本人还是强烈建议使用virtualenv来安装环境。细节就不多说了，让我们写个Hello World吧：

```
# 导入了 Flask 类。 该类的实例将会成为我们的 WSGI 应用
from flask import Flask

# 创建一个该类的实例。第一个参数是应用模块或者包的名称
app = Flask(__name__)


# route() 装饰器来告诉 Flask 触发函数的 URL
# 函数名称被用于生成相关联的URL
@app.route('/')
def index():
    # 返回需要在用户浏览器中显示的信息
    return '<h1>Hello World</h1>'


if __name__ == '__main__':
    # 默认为 127.0.0.1:5000
    app.run()

```

个Web应用的代码就写完了，对，就是这么简单！保存为”main.py”，打开控制台，到该文件目录下，运行

```
$ python main.py
```

如果看到

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

字样后，就说明服务器启动完成。打开你的浏览器，访问`http://127.0.0.1:5000/`，一个硕大的”Hello World”映入眼帘:-)

请不要使用`flask.py`作为应用名称，这会与 Flask 本身发生冲突

简单解释下这段代码

- 首先引入了Flask包，并创建一个Web应用的实例”app”

```
from flask import Flask
app = Flask(__name__)
```

这里给的实例名称就是这个python模块名。

定义路由规则

```
@app.route('/')
```

这个函数级别的注解指明了当地址是根路径时，就调用下面的函数。可以定义多个路由规则，会在下篇文章里详细介绍。说的高大上些，这里就是MVC中的Contoller。

处理请求

```
def index():
    return '<h1>Hello World</h1>'
```

当请求的地址符合路由规则时，就会进入该函数。可以说，这里是MVC的Model层。你可以在里面获取请求的request对象，返回的内容就是response。本例中的response就是大标题”Hello World”。

启动Web服务器

```
if __name__ == '__main__':
    app.run()
```

当本文件为程序入口（也就是用python命令直接执行本文件）时，就会通过`app.run()`启动Web服务器。如果不是程序入口，那么该文件就是一个模块。Web服务器会默认监听本地的5000端口，但不支持远程访问。如果你想支持远程，需要在`run()`方法传入`host=0.0.0.0`，想改变监听端口的话，传入`port=端口号`，你还可以设置调试模式。具体例子如下：

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8888, debug=True)
```

注意，Flask自带的Web服务器主要还是给开发人员调试用的，在生产环境中，你最好是通过WSGI将Flask工程部署到类似Apache或Nginx的服务器上

```
# 开启debug ： 激活自动重载 并 可以帮助我们查找代码里面的错误
```

运行时常见错误：

错误1：

```
    self.server_name = socket.getfqdn(host)
  File "C:\Users\Lenovo\AppData\Local\Programs\Python\Python37\lib\socket.py", line 676, in getfqdn
    hostname, aliases, ipaddrs = gethostbyaddr(name)
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xc1 in position 0: invalid start byte
```

错误原因：

在控制面板中将电脑名字改为英文名字，运行就ok

错误2：

```
127.0.0.1 - - [23/Oct/2021 15:44:08] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Oct/2021 15:44:08] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [23/Oct/2021 15:44:13] code 400, message Bad request version ('\x1b\x00\x00\x1aÀ')
127.0.0.1 - - [23/Oct/2021 15:44:13] "□□□ k□  g□□□ß  ~o  □{ɗ¤¹7GÜ5[1â`Þ□;□oè¡    1f□□  □À" HTTPStatus.BAD_REQUEST -
```

错误原因：

浏览器访问时使用：https ，将https修改为http

Flask入门系列(二)–路由

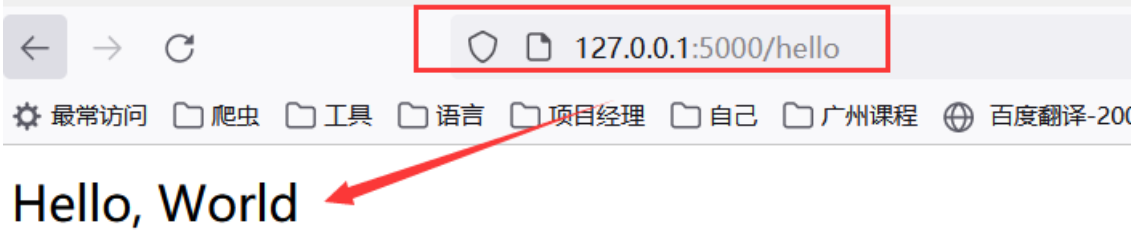
现代 web 应用都使用有意义的 URL ，这样有助于用户记忆，网页会更得到用户的青睐， 提高回头率。

使用 `route()` 装饰器来把函数绑定到 URL:

```python
@app.route('/')
def index():
    return 'Index Page'

@app.route('/hello')
def hello():
    return 'Hello, World'
```

浏览器的地址栏中输入:http://127.0.0.1:5000/

浏览器的地址栏中输入:`http://127.0.0.1:5000/hello



但是能做的不仅仅是这些！你可以动态变化 URL 的某些部分， 还可以为一个函数指定多个规则。

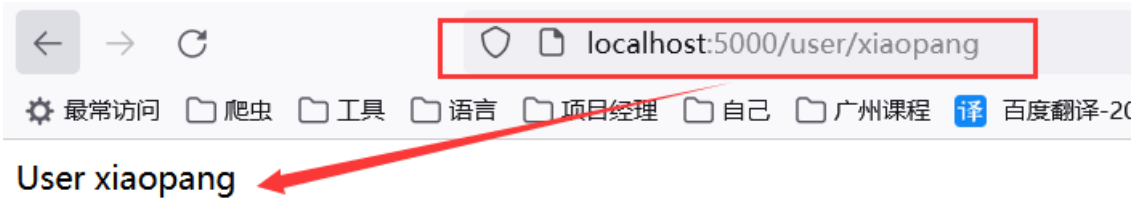
变量规则

通过把 URL 的一部分标记为 `<variable_name>` 就可以在 URL 中添加变量。标记的 部分会作为关键字参数传递给函数。通过使用 `<converter:variable_name>` ，可以 选择性的加上一个转换器，为变量指定规则。请看下面的例子:

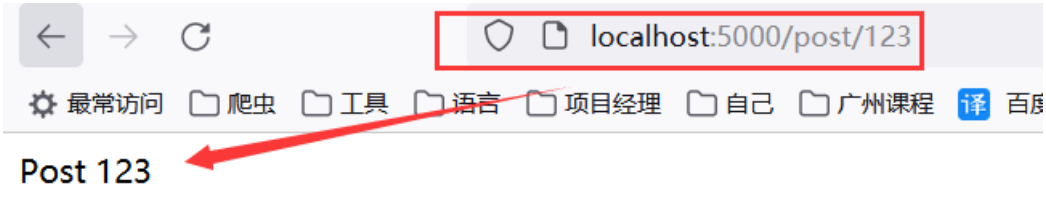
```
@app.route('/user/<username>')
def show_user_profile(username):
    # 显示该用户的用户配置文件 , escape用来转义
    return 'User %s' % escape(username)

@app.route('/post/<int:post_id>')
def show_post(post_id):
    # 显示具有给定id的帖子，id为整数
    return 'Post %d' % post_id

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    # 显示在/path之后的子路径
    return 'Subpath %s' % escape(subpath)
```

escape()是用来转义的,需要引入 from flask import escape

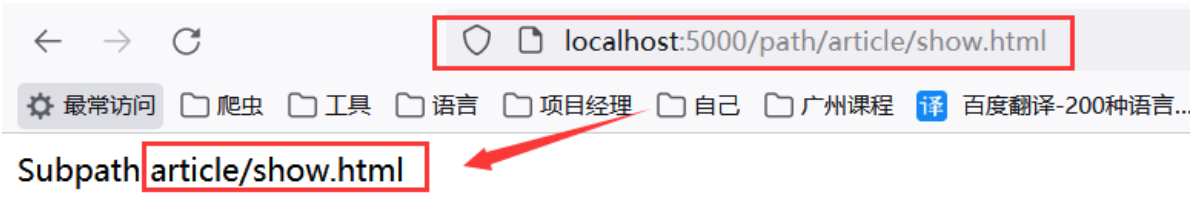
- 当你在浏览器的地址栏中输入`http://localhost:5000/user/xiaopang`，你将在页面上看到”User xiaopang”的字样。URL路径中/user/后面的参数被作为show_user_profile()`函数的`username参数传了进来。



- 试下访问`http://localhost:5000/post/xiaopang`，你会看到Not Found错误。但是试下`http://localhost:5000/post/123`，页面上就会有”Post 123”显示出来。参数类型转换器`int:`帮你控制好了传入参数的类型只能是整形。



- 试下访问`http://localhost:5000/path/article/show.html`，你将在页面上看到”Subpath article/show.html”的字样,会将path之后的路径地址全部作为show_subpath()函数的subpath参数值传递进去



目前支持的参数类型转换器有：

类型转换器	作用
缺省	字符型，但不能有斜杠
int:	整型
float:	浮点型
path:	字符型，可有斜杠

多URL的路由

一个函数上可以设施多个URL路由规则

```
@app.route('/muti')
@app.route('/muti/<name>')
def muti_route(name=None):
    if name is None:
        name = 'World'
    return '多路由 %s' % name
```

这个例子接受二种URL规则，`/muti`都不带参数，函数参数`name`值将为空，页面显示"多路由 World"；`/muti/<name>`带参数，页面会显示参数`name`的值。

- http://localhost:5000/muti/xiaowang 页面显示：多路由 xiaowang
- http://localhost:5000/muti 页面显示：多路由 World

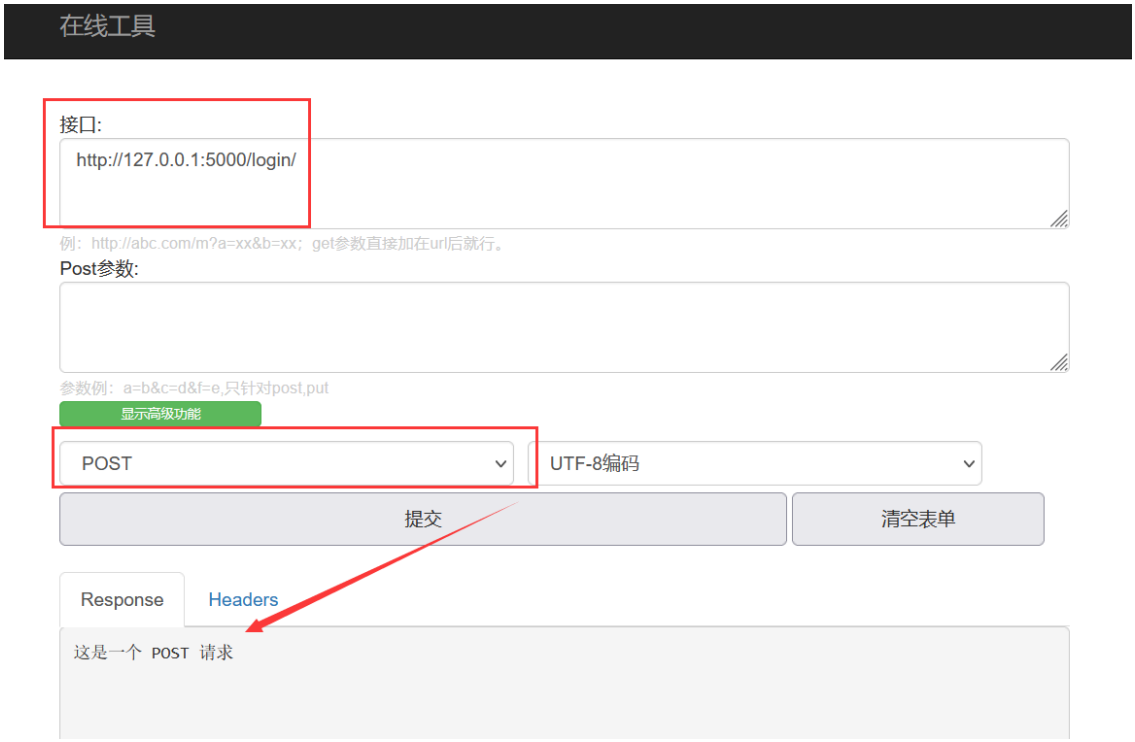
HTTP请求方法设置

HTTP请求方法常用的有Get, Post, Put, Delete。Flask路由规则也可以设置请求方法。

```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    # request返回请求的方式  需要 from flask import request
    if request.method == 'POST':
        return '这是一个 POST 请求'
    else:
        return '这是一个 GET 请求'
```

当你请求地址`http://localhost:5000/login/`，"GET"和"POST"请求会返回不同的内容

- 通过浏览器访问`http://localhost:5000/login/`，你将在页面上看到`这是一个 GET 请求`
- 可借助第三方测试软件



唯一的 URL / 重定向行为

以下两条规则的不同之处在于是否使用尾部的斜杠。：

```
@app.route('/projects/')
def projects():
    return 'The project page'

@app.route('/about')
def about():
    return 'The about page'
```

`projects` 的 URL 是中规中矩的，尾部有一个斜杠，看起来就如同一个文件夹。 访问一个没有斜杠结尾的 URL 时 Flask 会自动进行重定向，帮你在尾部加上一个斜杠。

```
当浏览器上输入:http://127.0.0.1:5000/projects
Flask会自动将http://127.0.0.1:5000/projects重定向到
http://127.0.0.1:5000/projects/
```

`about` 的 URL 没有尾部斜杠，因此其行为表现与一个文件类似。如果访问这个 URL 时添加了尾部斜杠就会得到一个 404 错误。这样可以保持 URL 唯一，并帮助 搜索引擎避免重复索引同一页面

URL 构建

`url_for()` 函数用于构建指定函数的 URL。它把函数名称作为第一个 参数。它可以接受任意个关键字参数，每个关键字参数对应 URL 中的变量。未知变量 将添加到 URL 中作为查询参数。

为什么不在把 URL 写死在模板中，而要使用反转函数 `url_for()` 动态构建?

1. 反转通常比硬编码 URL 的描述性更好。
2. 你可以只在一个地方改变 URL ，而不用到处乱找。
3. URL 创建会为你处理特殊字符的转义和 Unicode 数据，比较直观。
4. 生产的路径总是绝对路径，可以避免相对路径产生副作用。

5. 如果你的应用是放在 URL 根路径之外的地方（如在 `/myapplication` 中，不在 `/` 中），`url_for()` 会为你妥善处理。

```
url_for('login')    # 返回/login
url_for('login', id='1')    # 将id作为URL参数，返回/login?id=1
url_for('hello', name='man')    # 适配hello函数的name参数，返回/hello/man
url_for('static', filename='style.css')    # 静态文件地址，返回/static/style.css
```

静态文件

一个Web应用的静态文件包括了JS, CSS, 图片等，Flask的风格是将所有静态文件放在"static"子目录下。并且在代码或模板（下篇会介绍）中，使用 `url_for('static')` 来获取静态文件目录。如上第四个的例子就是通过 `url_for()` 函数获取"static"目录下的指定文件。如果你想改变这个静态目录的位置，你可以在创建应用时，指定 `static_folder` 参数。

```
app = Flask(__name__, static_folder='static')
```

Flask入门系列(三)-模板

在 Python 内部生成 HTML 不好玩，且相当笨拙。因为你必须自己负责 HTML 转义， 以确保应用的安全。因此， Flask 自动为你配置 `Jinja2` 模板引擎。

render_template()

使用 `render_template()` 方法可以渲染模板，你只要提供模板名称和需要 作为参数传递给模板的变量就行了。下面是一个简单的模板渲染例子:

```
from flask import render_template

@app.route('/hello/')
@app.route('/hello/<name>')
def hello(name=None):
    # 调用模板 并传 变量name
    return render_template('hello.html', name=name)
```

Flask 会在 `templates` 文件夹内寻找模板，文件路径如下:

```
/main.py
/templates
    /hello.html
```

你可以充分使用 Jinja2 模板引擎的威力。更多内容，详见官方 Jinja2 模板文档 。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='style.css') }}">
    <title>Hello from Flask</title>
</head>
```

```
<body>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}
</body>
</html>
```

这段代码是不是很像HTML？接触过其他模板引擎的朋友们肯定立马秒懂了这段代码。它就是一个HTML模板，根据`name`变量的值，显示不同的内容。变量或表达式由`{{ }}`修饰，而控制语句由`{% %}`修饰，其他的代码，就是我们常见的HTML。

让我们打开浏览器，输入`http://localhost:5000/hello/man`，页面上即显示大标题"Hello man!"。我们再看下页面源代码

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="/static/style.css">
    <title>Hello from Flask</title>
</head>
<body>

  <h1>Hello man!</h1>

</body>
</html>
```

flask之 jinja2模板内置过滤器

过滤器是通过管道符号（|）实用的，比如{{ age|abs }}是返回age的绝对值。过滤器类似函数，将参数传递给过滤器，再由过滤器根据其相应的功能返回相应的值去渲染网页。jinja2有许多内置的过滤器，下面一 一进行介绍：

```
abs(value)：返回value的绝对值

default(value,default_value)：设置默认值，如果value没有定义，则返回默认的default_value值

escape(value)/e(value)：将value中的<,>等符号转义为html符号

safe(value)：如果全局开启了转义，用safe可以将value的转义关闭

first(value)：获取value(序列)中的第一个值

last(value)：获取value(序列)中的最后一个值

length(value)：获取value的长度

format(value,**args,**kwargs)：格式化value，

 {{ '%s and %s' | format(xxx,yyy) }}

join(value,d='参数值')：将序列value用d的参数值进行拼接，拼接成一个字符串
```

```
wordcount(value)：获取value中的词的个数（以空格为分隔符）

trim(value)：截取value前后的空白字符

int(value)：将value转化为整型

float(value)：将value转化为浮点型

string(value)：将value转化为字符串类型

replace(value,old,new)：把value中的old替换成new，输出替换后的value

truncate(value,length=x,killwords=False)：截取value长度为length的字符串；
killwords为是否输出完整的单词

upper(value)：将value转化为大写

lower(value)：将value转化为小写
```

实例demo.filter.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>常见内建filter</h1>
<em>hello</em><hr>
<p>{{ '<em>hello</em>'}}</p><hr>
{#自动转义#}
<p>{{ '<em>hello</em>'|safe}}</p><hr>
{#变量首字母大写#}
<p>{{ 'hello'|capitalize}}</p><hr>
{#转化成大写#}
<p>{{ 'hello'|upper}}</p><hr>
{#转化成小写#}
<p>{{ 'HELLO'|lower}}</p><hr>
{#单词首字母大写#}
<p>{{ 'ada addada fhfghf'|title}}</p><hr>
{#格式化输出#}
<p>{{ '%s is %d'| format('李傲',18)}}</p><hr>
{#反转#}
<p>{{ 'hello'|reverse}}</p><hr>
{#渲染之前把所有html标签都去掉#}
<p>{{ '<h1>hello</h1>'|striptags}}</p><hr>
<hr>
<hr>
<h1>列表操作</h1>
<p>{{ [1,2,3,4 ]|first}}</p>
<p>{{ [1,2,3,4 ]|last}}</p>
<p>{{ [1,2,3,4 ]|length}}</p>
<p>{{ [1,2,3,4 ]|sum}}</p>
<p>{{ [1,2,3,4 ]|sort}}</p>
</body>
```

```
</html>
```

jinja2模板for循环

在`jinja2`中的`for`循环，跟`python`中的`for`循环基本上是一模一样的。也是`for...in...`的形式。并且也可以遍历所有的序列以及迭代器。但是唯一不同的是，`jinja2`中的`for`循环没有`break`和`continue`语句。

```
并且jinja2中的for循环还包含以下变量，可以用来获取当前的遍历状态
变量|描述
loop.index 当前迭代的索引（从1开始）
loop.index0 当前迭代的索引（从e开始）
1oop.first 是否是第一次迭代，返回True或 False
loop.last 是否是最后一次迭代，返回True或 False
11op.length 序列的长度
```

```python
from flask import Flask,render_template

app = Flask(__name__)
app.config['TEMPLATES_AUTO_RELOAD'] = True

@app.route('/')
def index_for():
    context = {
        'users':['捡猫1','捡猫2','捡猫3'],
        'person': {
            'username': '捡猫',
            'age': 18,
            'country': 'china'
        },
        'books':[
            {
                'name': '三国演义',
                'author':'罗贯中',
                'price': 110
            },{
                'name': '西游记',
                'author':'吴承恩',
                'price': 109
            },{
                'name': '红楼梦',
                'author':'曹雪芹',
                'price': 120
            },{
                'name': '水浒传',
                'author':'施耐庵',
                'price': 119
            }
        ]
    }
    return render_template('index_for.html',**context)


if __name__ == '__main__':
```

```
    app.run(debug=True)

```

templates/index_for.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>捡猫for循环</title>
</head>
<body>
    <ul>
        {% for user in users|reverse %}
            <li>{{ user }}</li>
        {% else %}
            <li>沒有任何值</li>
        {% endfor %}
    </ul>

    <table>
        <thead>
            <tr>
                <th>用户名</th>
                <th>年龄</th>
                <th>国家</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                {% for key in person.keys() %}
                    <td>{{ key }}</td>
                {% endfor %}
            </tr>
        </tbody>
    </table>

    <table>
        <thead>
            <tr>
                <th>序号</th>
                <th>书名</th>
                <th>作者</th>
                <th>价格</th>
                <th>总数</th>
            </tr>
        </thead>
        <tbody>
            {% for book in books %}
                {% if loop.first %}
                    <tr style="background: red;">
                {% elif loop.last %}
                    <tr style="background: pink;">
                {% else %}
                    <tr>
                {% endif %}
                    <td>{{ loop.index0 }}</td>
```

```
                    <td>{{ book.name }}</td>
                    <td>{{ book.author }}</td>
                    <td>{{ book.price }}</td>
                    <td>{{ loop.length }}</td>
                </tr>
            {% endfor %}
        </tbody>
    </table>

    <table border="1">
        <tbody>
            {% for x in range(1,10) %}
                <tr>
                    {% for y in range(1,10) if y <= x %}
                        <td>{{ y }}*{{ x }}={{ x*y }}</td>
                    {% endfor %}
                </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>

```

页面效果

- 捡猫3
- 捡猫2
- 捡猫1

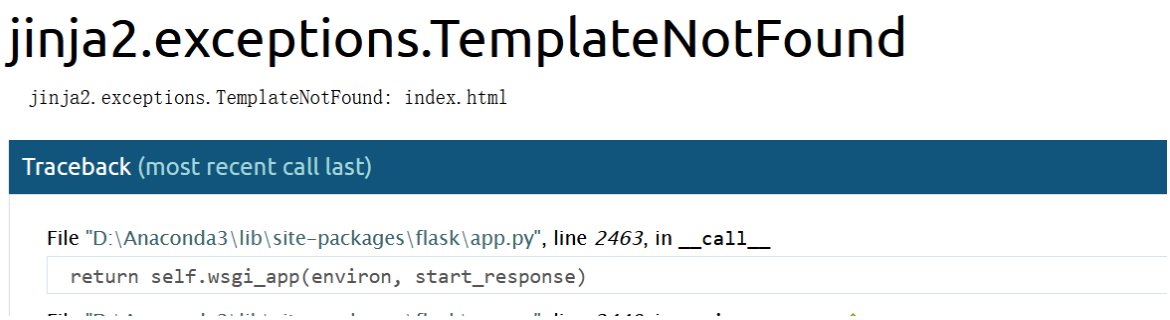
用户名	年龄	国家
username	age	country

序号	书名	作者	价格	总数
0	三国演义	罗贯中	110	4
1	西游记	吴承恩	109	4
2	红楼梦	曹雪芹	120	4
3	水浒传	施耐庵	119	4

1*1=1								
1*2=2	2*2=4							
1*3=3	2*3=6	3*3=9						
1*4=4	2*4=8	3*4=12	4*4=16					
1*5=5	2*5=10	3*5=15	4*5=20	5*5=25				
1*6=6	2*6=12	3*6=18	4*6=24	5*6=30	6*6=36			
1*7=7	2*7=14	3*7=21	4*7=28	5*7=35	6*7=42	7*7=49		
1*8=8	2*8=16	3*8=24	4*8=32	5*8=40	6*8=48	7*8=56	8*8=64	
1*9=9	2*9=18	3*9=27	4*9=36	5*9=45	6*9=54	7*9=63	8*9=72	9*9=81

常见错误

1.templates模板文件夹创建错误，或文件不存在 都会报一下错误



Flask入门系列(四)-请求，响应及会话

Flask内建对象

Flask提供的内建对象常用的有request, session, g，通过request，你还可以获取cookie对象。这些对象不但可以在请求函数中使用，在模板中也可以使用。

请求对象request

来自客户端网页的数据作为全局请求对象发送到服务器。为了处理请求数据，应该从Flask模块导入。

Request对象的重要属性如下所列：

- **Form** - 它是一个字典对象，包含表单参数及其值的键和值对。
- **args** - 解析查询字符串的内容，它是问号（？）之后的URL的一部分。
- **Cookies** - 保存Cookie名称和值的字典对象。
- **files** - 与上传文件有关的数据。
- **method** - 当前请求方法。

获取get方式提交的数据

实例：

main.py

```
from flask import Flask, render_template, request

# app类 ,对flask实例
app = Flask(__name__)


@app.route('/')
def index():
    #return "首页 <a href='/news_list/'>新闻列表页{name}
</a>".format(name=1212)
    return render_template('index.html')


@app.route('/gm02/news_list/')
def news_list():
    user_info = [
        {'id':1,"name": "苏圆圆", 'age': 18},
        {'id':2,"name": "林泽棕", 'age': 19},
```

```
        {'id':3,"name": "观景风", 'age': 20},
        {'id':4,"name": "吴武强", 'age': 12},
    ]
    # locals()以字典形式保存 函数中的所有局部变量
    #print(locals())
    return render_template('news_list.html', **locals())


@app.route('/user/detail.html')
def user_detail():
    #地址:127.0.0.1/user/detail.html?id=10&name=aaaaa
    # 参数id   值为10
    # 参数name   值为aaaaa
    # 取得get方式提交的参数 使用request.args.get() # 参数值都为字符串
    uid = int(request.args.get('uid'))
    users_info = [
        {'id': 1, "name": "苏圆圆", 'age': 18, 'other': "她喜欢吃吃吃.....
各种吃，就是不胖，就是玩！"},
        {'id': 2, "name": "林泽棕", 'age': 19, 'other': "喜欢按摩,各种按，全方位
的按,不怕花钱，就是玩！"},
        {'id': 3, "name": "观景风", 'age': 20, 'other': "喜欢上课，各种课，学习无
止境，别人都以为我在学习，我也不解释"},
        {'id': 4, "name": "吴武强", 'age': 12, 'other': "刷B站，各种刷，网罗全网
各色小姐姐."},
    ]
    res = {}
    # 循环判断参数uid和用户id是否相等。取出正确内容
    for user in users_info:
        if uid == user['id']:
            res = user
            break
    return render_template('user_detail.html', res=res)



@app.route('/news_list/detail_<int:nid>.html')
def news_detail(nid):
    #return f"新闻详细页-{nid}"
    #return render_template('news_detail.html', t_nid=nid, title="大数据2班")
    return render_template('news_detail.html', **{'t_nid': nid, 'title': "--
大数据2班--"})


if __name__ == '__main__':
    app.run('127.0.0.1', 80, debug=True)

```

模板：news_list.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title></title>
        <style>
```

```
        ul li {
            list-style: none;
        }
        </style>
    </head>
    <body>
        <table border="1">
            <tr>
                <th>编号</th>
                <th>姓名</th>
                <th>年龄</th>
                <th>查看爱好</th>
            </tr>
            <!----  循环展示学生信息----->
            {%  for user in user_info %}
            <tr>
                <td>{{ user.get('id', '') }}</td>
                <td>{{ user.get('name', '') }}</td>
                <td>{{ user['age'] }}</td>
                <td><a target="_blank" href="/user/detail.html?uid={{ user['id'] }}">查看--/user/detail.html?uid={{ user['id'] }}</a></td>
            </tr>
            {% endfor %}
        </table>

    </body>
</html>

```

页面:user_detail.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>2班学生-{{ res['name'] }}爱好页面</title>
    </head>
    <body>
        <h1>"{{ res['name'] }}"学生的爱好</h1>
        <p>
            <h3>学生:{{ res['name'] }} ,年龄:{{res.get('age')}}</h3>
            <span style="color: red;">爱好</span>:{{ res.get('other') }}
        </p>
    </body>
</html>

```

获取post方式提交的数据

引入flask包中的request对象，就可以直接在请求函数中直接使用该对象了。让我们改进下`login()`方法：

```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    # request返回请求的方式  需要 from flask import request
    if request.method == 'POST':
        # 获取页面表单提交的内容
        user_name = request.form.get('user_name')
        pwd = request.form.get('pwd')
        # 判断账号和密码是否正确
        if user_name == 'root' and pwd == '123456':
            # 登录成功后，页面返回登录成功
            return '<h1>登录成功</h1>'
            # 跳转到后台首页
            #return redirect(url_for('index_for'))
        else:
            # 登录错误，返回到登录页面，并提示错误信息
            return render_template('login.html', **{'error':'账号或密码错误！'})
    else:
        return render_template('login.html')
```

在的templates目录下，添加"login.html"文件

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <!--引入公共样式文件style.css和登录样式login.css-->
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',filename='login.css') }}">
    <title>登录页面</title>
</head>
<body>
<div class="main">
    <!-- action为提交地址，method为提交方式  -->
    <form action="/login/" method="post">
        <ul>
            <li>欢迎登录</li>
            <li><label>账 号：<input type="text" autocomplete="off" name="user_name" required></label></li>
            <li><label>密 码：<input type="password" name="pwd" required></label></li>
            <li><button>登录</button></li>
            <!-- 用户登录失败时 提示错误信息-->
            <li class="login_error">{% if error %} {{ error }}{% endif%}</li>
        </ul>
    </form>
</div>
</body>
```

```
</html>
```

在的static目录下，添加"style.css"公共样式文件和`login.css`登录样式文件

style.css

```
html, body, div, span, applet, object, iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre, a, abbr, acronym, address, big, cite, code, del, dfn, em, font, img, ins, kbd, q, s, samp, small, strike,
strong, sub, sup, tt, var, b, u, i, center, dl, dt, dd, ol, ul, li, fieldset, form, label, legend, table, caption, tbody,
tfoot, thead, tr, th, td ,textarea,input { margin:0; padding:0;  }
address,cite,dfn,em,var, i {font-style:normal;}
body {font-size: 16px; line-height: 1.5; font-family:'Microsoft Yahei','simsun','arial','tahoma';  color: #333; }
table { border-collapse:collapse; border-spacing:0; }
h1, h2, h3, h4, h5, h6, th { font-size: 100%; font-weight: normal; }
button,input,select,textarea{font-size:100%;}
fieldset,img{border:0;}
a,
img {
    -webkit-touch-callout: none
}
a,
a:active,
a:focus,
a:hover,
a:visited {text-decoration: none}
input[type=password],
input[type=text],
textarea {
    resize: none;
    outline: 0;
    -webkit-appearance: none;
    white-space: pre-wrap;
    word-wrap: break-word;
    background: #fff
}
ul, ol { list-style: none; }
:focus{ outline:none;}
.clearfix{ clear: both; content: ""; display: block; overflow: hidden }
.clear{clear: both;}
.fl{ float: left; }
.fr{float: right;}
```

login.css

```
.main{
    width: 240px;
    height: 160px;
    position: absolute;top:50%;left: 50%;
    margin-top:-100px;
    margin-left:-120px;
    text-align:center;
    border-radius: 10px;
    background-color: #888888;
}
```

```
.main ul li{
    margin-top: 10px;
}
.login_error{
    font-size: 10px;
    color: red;
    margin-top: 0px !important;
}
```

执行上面的例子，结果我就不多描述了。简单解释下，`request`中`method`变量可以获取当前请求的方法，即"GET", "POST", "DELETE", "PUT"等；`form`变量是一个字典，可以获取"Post"请求表单中的内容，在上例中，如果提交的表单中不存在`user`项，则会返回一个`KeyError`，你可以不捕获，页面会返回400错误（想避免抛出这`KeyError`，你可以用`request.form.get('user')`来替代）。而`request.args.get()`方法则可以获取"Get"请求URL中的参数，该函数的第二个参数是默认值，当URL参数不存在时，则返回默认值。`request`的详细使用可参阅Flask的官方API文档

会话对象session

会话可以用来保存当前请求的一些状态，以便于在请求之前共享信息。这个对象相当于用密钥签名加密的 cookie ，即用户可以查看你的 cookie ，但是如果没有密钥就无法修改它。 我们将上面的python代码改动下：

```
from flask import Flask, escape, request, render_template, redirect, url_for, session


# 创建一个该类的实例。第一个参数是应用模块或者包的名称
app = Flask(__name__, static_folder='static')
# 使用session，需要需要设置秘钥
app.secret_key = 'a1s@i34!3-&d'
@app.route('/')
def index():
    # 判断 'user' 是否存在于session中
    if 'user' in session:
        # 如果存在 则返回admin_index.html模板
        return render_template('admin_index.html')
    # 如果不存在 返回Flask
    return '<div>Hello %s</div>' % '<em>Flask</em>---session中无用户信息'


@app.route('/login/', methods=['GET', 'POST'])
def login():
    # request返回请求的方式  需要 from flask import request
    if request.method == 'POST':
        # 获取页面表单提交的内容
        user_name = request.form.get('user_name')
        pwd = request.form.get('pwd')
        # 判断账号和密码是否正确
        if user_name == 'root' and pwd == '123456':
            # 登录成功后，将用户名保存到session中，
            session['user'] = user_name
            #return '<h1>登录成功</h1>'
            # 跳转到后台首页
            return redirect(url_for('index'))
        else:
            # 登录错误，返回到登录页面，并提示错误信息
```

```
            return render_template('login.html', **{'error':'账号或密码错误！'})
    else:
        return render_template('login.html')
```

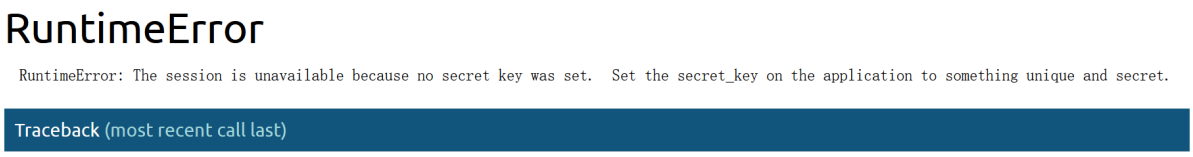
admin_index.html页面内容：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>后台首页</title>
</head>
<body>
<!-- 判断'user'是否存在session中，存在则显示session中保存的用户信息 -->
<h1>欢迎
    {% if 'user' in session %}
        {{ session['user'] }}
    {% else %}
        ---无session数据
    {% endif %}</h1>
</body>
</html>
```

第7行，在使用session之前需要需要设置秘钥

```
app.secret_key = '123456'   # 密钥要尽量复杂，最好使用一个随机数，如: a1s@i34!3-&d
```

不设置，则会报错：




```
            # 登录成功后，将用户名保存到session中，
            session['user'] = user_name
            #return '<h1>登录成功</h1>'
            # 跳转到后台首页
            return redirect(url_for('index'))
```

25到31行代,设置session内容，并跳转到指定页面

```
        # 判断账号和密码是否正确
        if user_name == 'root' and pwd == '123456':
            # 登录成功后，将用户名保存到session中，
            session['user'] = user_name
            #return '<h1>登录成功</h1>'
            # 跳转到后台首页
            return redirect(url_for('index'))
```

11行到15行：判断用户是否登录成功，如果成功登录账号信息

```
    # 判断 'user' 是否存在于session中
    if 'user' in session:
        # 如果存在 则返回admin_index.html模板
        return render_template('admin_index.html')
    # 如果不存在 返回Flask
    return '<div>Hello %s</div>' % '<em>Flask</em>'
```

对应的退出操作:

```
@app.route('/logout')
def logout():
    # 移除session中的user
    session.pop('user', None)
    # 跳转到login登录页面
    return redirect(url_for('login'))
```

对应admin_index.html新增 `退出` 按钮

```
<!-- 判断'user'是否存在session中，存在则显示session中保存的用户信息 -->
<h1>欢迎{% if 'user' in session %} {{ session['user'] }}
        <a style="font-size:10px" href="/logout">退出</a>
        {% else %}
        ---无session数据
    {% endif %}
</h1>
```

Cookies

要访问 cookies ，可以使用 `cookies` 属性。可以使用响应 对象 的 `set_cookie` 方法来设置 cookies 。请求对象的 `cookies` 属性是一个包含了客户端传输的所有 cookies 的字典。在 Flask 中，如果使用 [会话]，那么就不要直接使用 cookies ，因为 [会话]比较安全一些。

读取 cookies:

```
from flask import request

@app.route('/')
def index():
    # 注意cookies为复数
    username = request.cookies.get('username')
    # 使用 cookies.get(key) 而不是使用cookies[key]获取，因为当key不存时会报错
```

储存 cookies:

```
from flask import make_response

@app.route('/')
def index():
    resp = make_response(render_template(...))
    resp.set_cookie('username', 'the username')
    return resp
```

修改之前的登录操作

```python
@app.route('/login/', methods=['GET', 'POST'])
def login():
    # request返回请求的方式  需要 from flask import request
    if request.method == 'POST':
        # 获取页面表单提交的内容
        user_name = request.form.get('user_name')
        pwd = request.form.get('pwd')
        # 判断账号和密码是否正确
        if user_name == 'root' and pwd == '123456':
            # 登录成功后，将用户名保存到session中，
            # session['user'] = user_name
            # 跳转到后台首页
            # 使用make_response创建一个相应对象，然后再添加其他信息
            resp = make_response(redirect(url_for('index')))
            # 设置cookie
            resp.set_cookie('user', user_name)
            return resp
        else:
            # 登录错误，返回到登录页面，并提示错误信息
            return render_template('login.html', **{'error':'账号或密码错误! '})
    else:
        return render_template('login.html')
```

将11行代码修改为14到17行代码，使用make_response创建一个相应对象，然后再添加cookie信息，返回给浏览器，使浏览器保存cookie信息。

后台首页判断cookies中是否存在信息，进行不同的处理

```python
@app.route('/')
def index():
    # 判断 'user' 是否存在于session中
    #if 'user' in session:
    # 判断 'user' 是否存在于cookie中
    if request.cookies.get('user'):
        # 如果存在 则返回之前保存在session中的用户名
        return render_template('admin_index.html')
    # 如果不存在 返回Flask
    return '<div>Hello %s</div>' % '<em>Flask</em>---session中无用户信息'
```

页面admin_index.html页面修改为对cookies的判断

```
<!-- 判断'user'是否存在request.cookies中，存在则通过request.cookies.get取得用户保存的数据 -->
<h1>欢迎{% if request.cookies.get('user') %} {{ request.cookies.get('user') }}
        <a style="font-size:10px" href="/logout">退出</a>
        {% else %}
        ---无cookie数据
    {% endif %}
</h1>
```

Flask入门系列(五)-数据库操作

Flask 之 g属性

g: global
g对象解释： 就是为了保存用户一些自定义参数

1. g对象是专用用来保存用户的数据的。
2. g对象在一次请求中，全局可以调用。

经常用来保存数据库连接操作

配置连接参数

创建配置文件"config.py"，保存配置信息：

```
HOST = "localhost",  # 本地为localhost
PORT = 3306,  # 端口号
USER = "root",  # 账号
PASSWORD = "123456",  # 密码
DB = "gm03"  # 数据库名
```

在创建Flask应用时，导入配置信息

```
from flask import Flask
# 导入配置文件
import config

# 创建一个该类的实例。第一个参数是应用模块或者包的名称
app = Flask(__name__, static_folder='static')
# 将配置文件加载到app配置中 ,之后就可以只用app.config获取文件中的变量
# 需要先打入config文件
app.config.from_object('config')
```

建立和释放数据库连接

这里要用到请求的上下文装饰器

```
from Flask import g

@app.before_request
def before_request():
    # 设置g的属性 conn数据库连接
    g.conn = pymysql.connect(
            host="localhost",  # 本地为localhost
            user="root",  # 账号
            port=3306,  # 端口号
            password="123456",  # 密码
            database="gm03"  # 数据库名
        )
    # 设置g的属性 cursor数据库游标
    g.cursor = g.conn.cursor(pymysql.cursors.DictCursor)

# 每次请求关闭前被调用
@app.teardown_request
def teardown_request(exception):
    # 取出全局变量中的conn 和 cursor
    conn = getattr(g, 'conn', None)
```

```
    cursor = getattr(g, 'cursor', None)
    # 当存在时，关闭游标和连接操作
    if conn is not None and cursor is not None:
        cursor.close()
        conn.close()
```

我们在before_request()里建立数据库连接，它会在每次请求开始时被调用；

并在teardown_request()关闭它，它会在每次请求关闭前被调用。

却表gm03数据库中存在用户表，如果没有，创建user表和导入输入

```
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `user_name` varchar(20) NOT NULL,
  `pwd` varchar(20) NOT NULL,
  `reg_datetime` datetime DEFAULT NULL,
  `login_datetime` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `user_name` (`user_name`)
) ENGINE=InnoDB AUTO_INCREMENT=2 DEFAULT CHARSET=utf8;
INSERT INTO `user` VALUES (1, 'root', '123456', '2021-10-21 11:22:53',
'2021-10-21 17:11:54');
```

修改登录程序

```
@app.route('/login/', methods=['GET', 'POST'])
def login():
    # request返回请求的方式  需要 from flask import request
    if request.method == 'POST':
        # 获取页面表单提交的内容
        user_name = request.form.get('user_name')
        pwd = request.form.get('pwd')
        # 操作全局游标 改写为（execute帮我们做字符串拼接，我们无需且一定不能再为%s加引
号了
        g.cursor.execute('select * from user where user_name=%s and pwd=%s
', (user_name.strip(), pwd.strip()))
        # 对返回行数来确定是否查询到数据
        if g.cursor.rowcount:
            # 登录成功后，将用户名保存到session中，
            # session['user'] = user_name
            # 跳转到后台首页
            # 使用make_response创建一个相应对象，然后再添加其他信息
            resp = make_response(redirect(url_for('index')))
            # 设置cookie
            resp.set_cookie('user', user_name)
            return resp
        else:
            # 登录错误，返回到登录页面，并提示错误信息
            return render_template('login.html', **{'error': '账号或密码错
误！'})
    else:
        return render_template('login.html')
```

第9行使用全局g中保存的游标进行操作，第11 行对返回行数来确定是否查询到数据，：

```
        # 操作全局游标 改写为（execute帮我们做字符串拼接，我们无需且一定不能再为%s加引号了
        g.cursor.execute('select * from user where user_name=%s and pwd=%s ', (user_name.strip(), pwd.strip()))
        # 对返回行数来确定是否查询到数据
        if g.cursor.rowcount:
```