

python flask框架详解

[python flask框架详解-CSDN博客](#)

[https://www.bilibili.com/video/BV17r4y1y7jJ/?](https://www.bilibili.com/video/BV17r4y1y7jJ/?p=4&share_source=copy_web&vd_source=00de136a1bd6eae86f7ccd0c05e6332d)

[p=4&share_source=copy_web&vd_source=00de136a1bd6eae86f7ccd0c05e6332d](https://www.bilibili.com/video/BV17r4y1y7jJ/?p=4&share_source=copy_web&vd_source=00de136a1bd6eae86f7ccd0c05e6332d)

Flask是一个Python编写的Web 微框架，让我们可以使用Python语言快速实现一个网站或Web服务。通常来说，大型项目用Django，小型项目用Flask。著名的网飞（Netflix）也是使用Flask开发。

Flask是轻量框架，本身带有Werkzeug(用于路由解析)和Jinja2（用于模板渲染），同时Flask有丰富的第三方库，需要什么就安装什么，所以自身是比较小巧的。

1.安装flask

```
pip install flask
```

2.简单上手

一个最小的 Flask 应用如下:

```
from flask import Flask

#使用flask类创建一个app对象
#__name__:代表当前app.py这个模块
#1.以后出现bug，可以帮助快速定位；
#2.对于寻找模块文件，有一个相对路径
app = Flask(__name__)

#创建一个路由和视图函数的映射
@app.route('/')#路由
def hello_world():
    return 'Hello World'

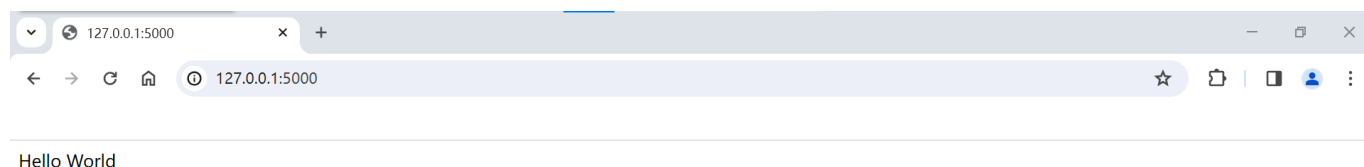
if __name__ == '__main__':
    app.run()
```

代码解析：1、首先我们导入了 Flask 类。该类的实例将会成为我们的 WSGI 应用。2、接着我们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。如果你使用一个单一模块（就像本例），那么应当使用 name，因为名称会根据这个模块是按应用方式使用还是作为一个模块导入而发生变化（可能是 'main'，也可能是实际导入的名称）。这个参数是必需的，这样 Flask 才能知道在哪里可以找到模板和静态文件等东西 3、然后我们使用 route() 装饰器来告诉 Flask 触发函数的 URL。4、函数名称被用于生成相关联的 URL。函数最后返回需要在用户浏览器中显示的信息。

运行结果：

```
* Serving Flask app 'app'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

访问：<http://127.0.0.1:5000/>



2.1 app.route()函数

```
app.route(rule, options)
```

- rule 参数表示与该函数的URL绑定。
- options 是要转发给基础Rule对象的参数列表。在上面的示例中，'/' URL与hello_world()函数绑定。因此，当在浏览器中打开web服务器的主页时，将呈现该函数的输出。最后，Flask 类的run()方法在本地开发服务器上运行应用程序。

2.2 app.run()函数

```
app.run(host, port, debug, options)
```

所有参数都是可选的

- host：要监听的主机名。默认为127.0.0.1（localhost）。设置为“0.0.0.0”以使服务器在外部可用
- port：默认值为5000

- debug: 默认为false。如果设置为true, 则提供调试信息, 可以自动重载代码并显示调试信息
- options: 要转发到底层的Werkzeug服务器。

2.3 调试模式

虽然 flask 命令可以方便地启动一个本地开发服务器, 但是每次应用代码 修改之后都需要手动重启服务器。

这样不是很方便, Flask 可以做得更好。如果你打开 调试模式, 那么服务器会在修改应用代码之后自动重启, 并且当应用出错时还会提供一个 有用的调试器。在命令行中, 如果需要打开所有开发功能 (包括调试模式), 那么要在运行服务器之前导出 FLASK_ENV 环境变量并把其设置为 development:

```
$ export FLASK_ENV=development
$ flask run
```

在代码中, 在运行或将调试参数传递给run()方法之前, 通过将application对象的debug属性设置为True来启用Debug模式。

```
app.debug = True
app.run()
# 或者
app.run(debug = True)
```

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

修改hello_world()函数:

```
* Restarting with watchdog (windowsapi)
* Debugger is active!
* Debugger PIN: 627-486-913
127.0.0.1 - - [29/Feb/2024 10:54:06] "GET / HTTP/1.1" 200 -
```

- 1.开启debug模式后, 只要修改代码后保存, 就会自动重写加载, 不需要手动重启项目;
- 2.如果开发的时候, 出现bug, 如果开启了debug模式, 在浏览器上就可以看到出错信息;

2.4 绑定IP和端口

默认情况下, Flask绑定IP为127.0.0.1, 端口为5000。我们也可以通过下面的方式自定义:

```
app.run(host='0.0.0.0', port=80, debug=True)
```

0.0.0.0代表电脑所有的IP。80是HTTP网站服务的默认端口。什么是默认？比如，我们访问网站<http://www.example.com>，其实是访问的<http://www.example.com:80>，只不过:80可以省略不写。

修改host：让其他电脑能访问到我电脑上的flask项目；

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

查看ip地址

```
>ipconfig
```

可以看到我的电脑ip地址：192.168.211.199

```
本地链接 IPv6 地址. . . . . : fe80::fcc9:3450:7d00:2d2a%18  
IPv4 地址 . . . . . : 192.168.211.199  
子网掩码 . . . . . : 255.255.255.0  
默认网关. . . . . : fe80::ecfa:93ff:fe03:792f%18  
192.168.211.174  
  
WARNING: This is a development server. Do not use it in a production deployment. Use a production  
* Running on all addresses (0.0.0.0)  
* Running on http://127.0.0.1:5000  
* Running on http://192.168.211.199:5000  
Press CTRL+C to quit  
* Restarting with watchdog (windowsapi)  
* Debugger is active!  
* Debugger PIN: 627-486-913
```

修改port端口号：如果5000端口被其他程序占用了，那么可以通过修改port来修改端口号；

3.URL与视图的映射

URL俗称网址，全称为统一资源定位系统（uniform resource locator;URL）是因特网的万维网服务程序上用于指定信息位置的表示方法。

例如：一个url的组成为：<http://www.baidu.com:443/path>，各部分为：

- 请求协议：http协议使用80端口，https用的是443端口
- 域名：www.baidu.com是域名

- 端口号
- path: 路径。url与视图其实就是path与视图，因为需要我们输入的就是一个path

以百度网址为例：



3.1 Flask中的URL和视图函数

在Flask项目中，经常可以看到如下代码结构：

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

@app.route中第一个字符串参数叫做URL，@app.route装饰的函数叫做视图函数。

3.2 URL和视图的映射

现代Web框架使用路由技术来帮助用户记住应用程序URL。可以直接访问所需的页面，而无需从主页导航。

Flask中的route()装饰器用于将URL绑定到函数。

可以在代码中看见URL和视图函数的映射关系：

```
@app.route('/')
def hello_world():
    return 'Hello World!'
```

其中装饰器@app.route添加了访问的URL规则—— '/'， '/' 代表网站的根路径，只要在浏览器中输入本网站的域名即可访问到 '/'。被装饰的函数hello_world会在浏览器访问路径 '/' 时被执行，此时hello_world函数没有做任何事情，只是简单返回了一个字符串'Hello World!'。

```
@app.route('/hello')
def hello_world():
    return 'hello world'
```

在这里，URL `/hello` 规则绑定到 `hello_world()` 函数。因此，如果用户访问 <http://localhost:5000/hello> URL，`hello_world()` 函数的输出将在浏览器中呈现。

`application` 对象的 `add_url_rule()` 函数也可用于将 URL 与函数绑定，如上例所示，使用 `route()` 装饰器的目的也由以下表示：

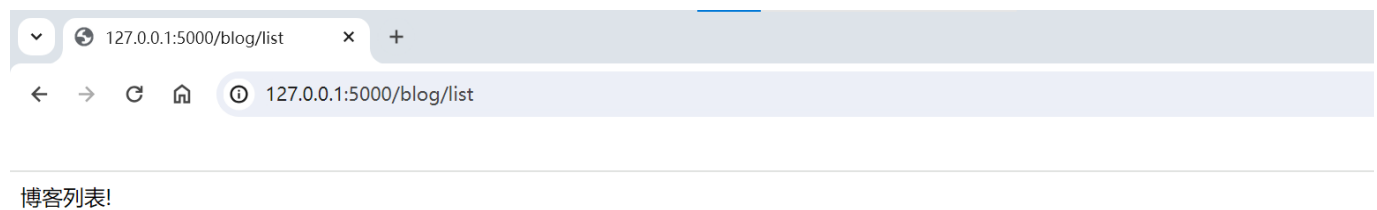
```
def hello_world():
    return 'hello world'
app.add_url_rule('/', 'hello', hello_world)
```

举例：

3.2.1 定义无参函数

```
@app.route('/blog/list')
def blog_list():
    return '博客列表!'
```

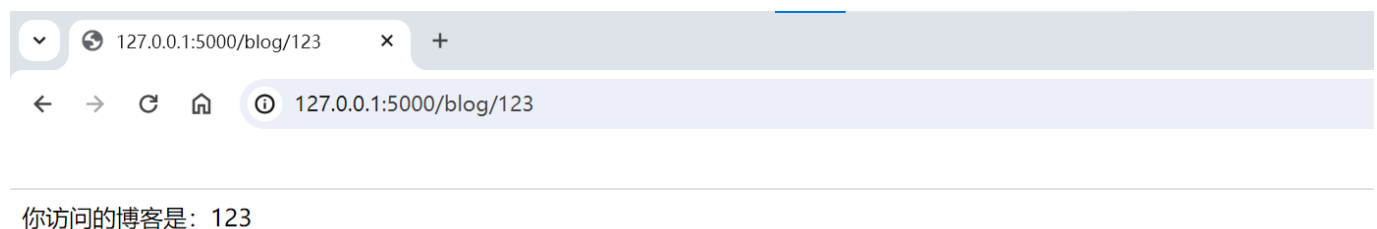
打开浏览器并输入 URL - <http://127.0.0.1:5000/blog/list>



3.2.2 定义带参数的URL

```
#带参数的url:将参数固定到path中
@app.route('/blog/<blog_id>')
def blog_detail(blog_id):
    return '你访问的博客是: %s' % blog_id
```

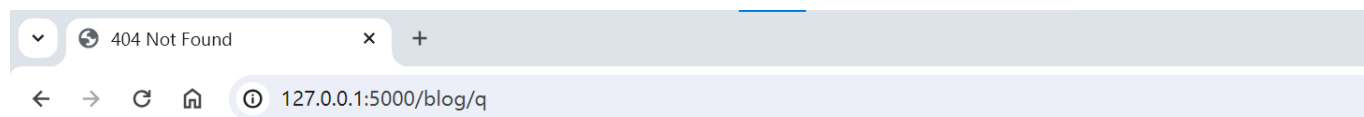
打开浏览器并输入 URL - <http://127.0.0.1:5000/blog/123>



带有类型的id

```
#带参数的url:将参数固定到path中
@app.route('/blog/<int:blog_id>')
def blog_detail(blog_id):
    return '你访问的博客是: %d' % blog_id
```

打开浏览器并输入URL - <http://127.0.0.1:5000/blog/q>



Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

除了 int 类型以外，URL 中的参数还可以指定以下类型，如表 3-1 所示。

表 3-1 URL 中参数类型

| 参数类型 | 描述 |
|--------|------------------------------------|
| string | 字符串类型。可以接受除/以外的字符。 |
| int | 整形。可以接受能通过 int()方法转换的字符。 |
| float | 浮点类型。可以接受能通过 float()方法转换的字符。 |
| path | 路径。类似 string，但是中间可以添加/。 |
| uuid | UUID 类型。UUID 是由一组 32 位数的 16 进制所构成。 |
| any | 备选值中的任何一个。 |

- any的使用：如果要实现一个获取某个分类的博客列表，但是博客分类只能是：

python, flask, django;

```
#带参数的url:将参数固定到path中
@app.route('/blog/<any(python,flask,django):category>')
def blog_list_with_category(category):
    return '你获取的博客分类是: %s' % category
```

打开浏览器并输入URL - <http://127.0.0.1:5000/blog/python>

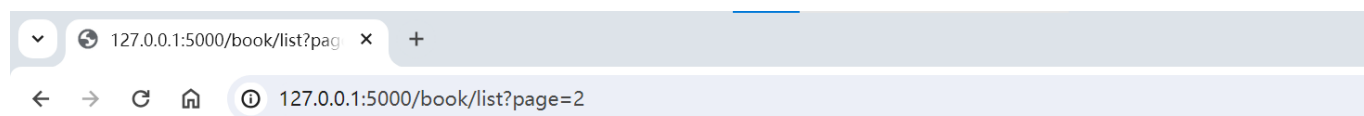


你获取的博客分类是: python

3.2.3 查询字符串传参

```
# 查询字符串传参
# /book/list: 会返回第一列的数据
# /book/list?page=2: 获取第二页的数据
@app.route('/book/list')
def book_list():
    #arguments:参数
    #request.args:类字典类型
    page=request.args.get("page",default=1,type=int)
    return f"您获取的是第{page}页的图书列表！"
```

打开浏览器并输入URL - <http://127.0.0.1:5000/book/list?page=2>



您获取的是第2页的图书列表！

3.3 唯一的 URL / 重定向行为

3.3.1 唯一的URL

以下两条规则的不同之处在于是否使用尾部的斜杠。

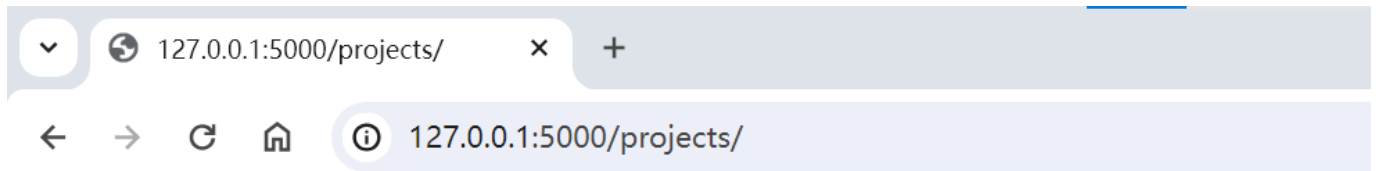
```
@app.route('/projects/')
def projects():
    return 'The project page'

@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。

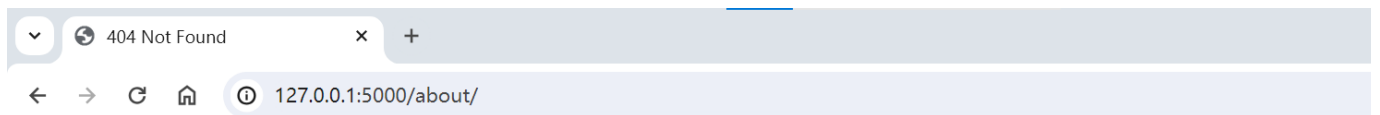
about 的 URL 没有尾部斜杠，因此其行为表现与一个文件类似。如果访问这个 URL 时添加了尾部斜杠就会得到一个 404 错误。这样可以保持 URL 唯一，并帮助搜索引擎避免重复索引同一页面。

打开浏览器并输入URL - <http://127.0.0.1:5000/projects>



The project page

打开浏览器并输入URL - <http://127.0.0.1:5000/about/>



Not Found

The requested URL was not found on the server. If you entered the URL manually please check your spelling and try again.

3.3.2 重定向行为

可以实现的功能例如：当我们访问一个需要用户先登陆才能访问的地址时，比如获取用户信息，如果用户没登陆，需重定向到登录页。

redirect函数用于重定向，实现机制很简单，就是向客户端（浏览器）发送一个重定向的HTTP报文，浏览器会去访问报文中指定的url。

url_for()函数对于动态构建特定函数的URL非常有用。一般我们通过一个URL就可以执行到某一个函数。如果反过来，我们知道一个函数，url_for函数就可以帮我们去获得这个URL。该函数接受函数的名称作为第一个参数，以及一个或多个关键字参数，每个参数对应于URL的变量部分。

```
from flask import Flask, redirect, url_for

app = Flask(__name__)

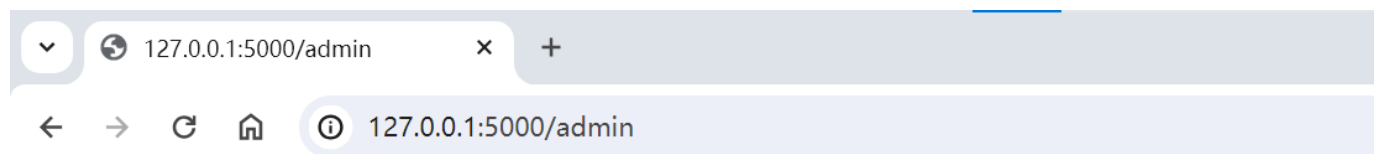
@app.route('/admin')
def hello_admin():
    return 'Hello Admin'

@app.route('/guest/<guest>')
def hello_guest(guest):
    return 'Hello %s as Guest' % guest

@app.route('/user/<name>')
def hello_user(name):
    if name == 'admin':
        return redirect(url_for('hello_admin'))
    else:
        return redirect(url_for('hello_guest', guest=name))

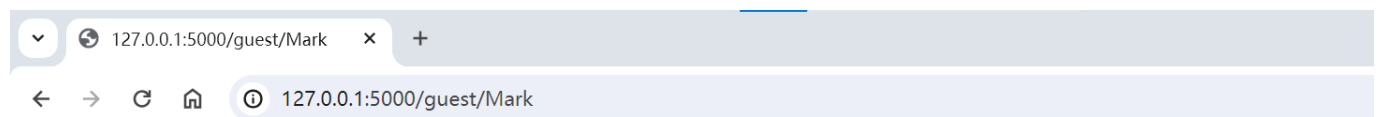
if __name__ == '__main__':
    app.run(debug=True)
```

打开浏览器并输入URL - <http://127.0.0.1:5000/user/admin>



Hello Admin

打开浏览器并输入URL - <http://127.0.0.1:5000/user/Mark>



Hello Mark as Guest

3.4 Flask HTTP方法

Web 应用使用不同的 HTTP 方法处理 URL 。当你使用 Flask 时，应当熟悉 HTTP 方法。在 @app.route() 中可以传入一个关键字参数 methods 来指定本方法支持的 HTTP 方法，默认情况下，只能使用 GET 请求。可以使用 route() 装饰器的 methods 参数来处理不同的 HTTP 方法:

| 方法 | 描述 |
|--------|--|
| GET | 以未加密的形式将数据发送到服务器，最常见的方法。 |
| HEAD | 和 GET 方法相同，但没有响应体。 |
| POST | 用于将 HTML 表单数据发送到服务器，POST 方法接收的数据不由服务器缓存。 |
| PUT | 用上传的内容替换目标资源的所有当前表示。 |
| DELETE | 删除由 URL 给出的目标资源的所有当前表示。 |

默认情况下，Flask 路由响应 GET 请求。但是，可以通过为 route() 装饰器提供方法参数来更改此首选项。

GET 请求和 POST 请求的区别:

- (1) post 传输数据大小无限制，get 请求传输数据大小有限制，最多可传递 2kb 的数据。
- (2) post 请求比 get 请求方式更安全。Get 请求方式的参数信息都会再 URL 地址栏明文显示，而 post 请求方式传递的参数信息隐藏在实体内容中，用户是看不到的。

为了演示在 URL 路由中使用 POST 方法，首先让我们创建一个 HTML 表单，并使用 POST 方法将表单数据发送到 URL。将以下脚本另存为 login.html

```
<html>
  <body>

    <form action = "http://localhost:5000/login" method = "post">
      <p>Enter Name:</p>
      <p><input type = "text" name = "nm" /></p>
      <p><input type = "submit" value = "submit" /></p>
    </form>

  </body>
</html>
```

运行以下代码:

```

from flask import Flask, redirect, url_for, request
app = Flask(__name__)

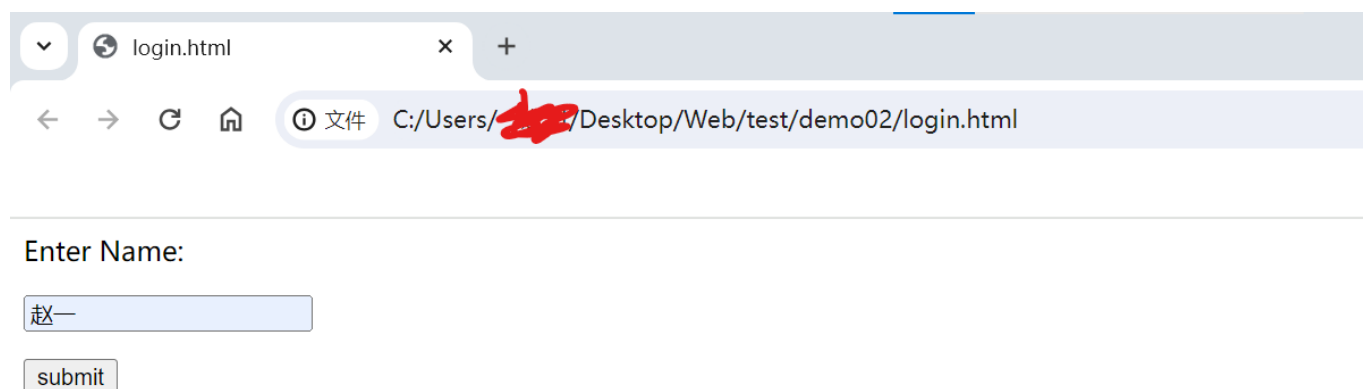
@app.route('/success/<name>')
def success(name):
    return 'welcome %s' % name

@app.route('/login', methods = ['POST', 'GET'])
def login():
    if request.method == 'POST':
        user = request.form['nm']
        return redirect(url_for('success', name = user))
    else:
        user = request.args.get('nm')
        return redirect(url_for('success', name = user))

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

```

在浏览器中打开login.html，在文本字段中输入name，然后单击提交



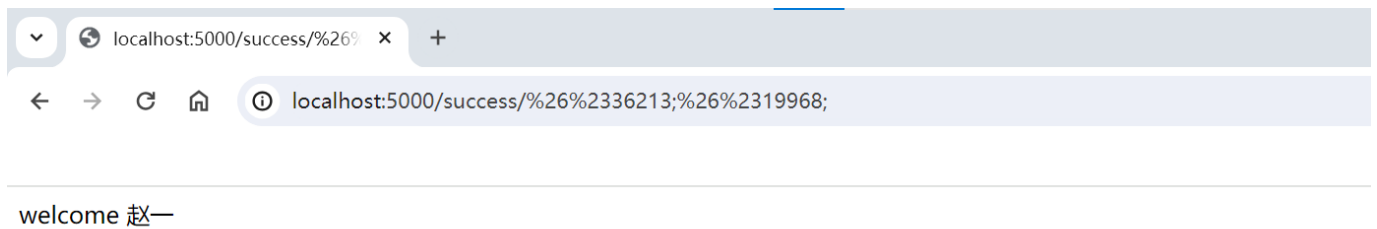
Enter Name:

表单数据将POST到表单标签的action子句中的URL。

<http://localhost/login> 映射到login()函数。由于服务器通过POST方法接收数据，因此通过以下步骤获得从表单数据获得的“nm”参数的值：

```
user = request.form['nm']
```

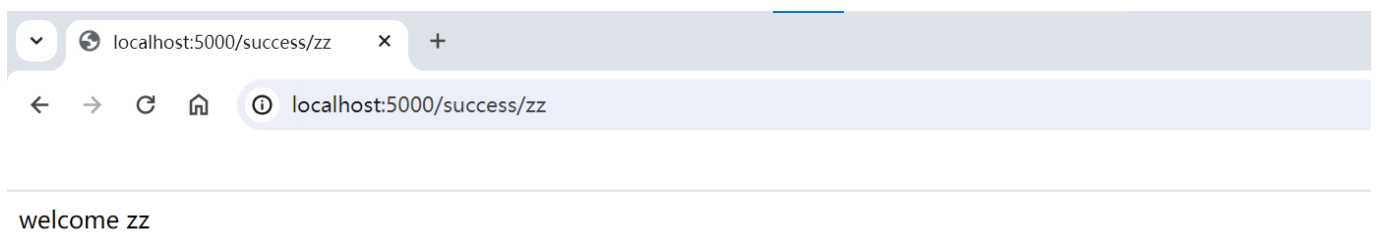
它作为变量部分传递给‘/ success’ URL。浏览器在窗口中显示welcome消息。



在login.html中将方法参数更改为'GET'，然后在浏览器中再次打开它。服务器上接收的数据是通过GET方法获得的。通过以下的步骤获得'nm'参数的值：

```
User = request.args.get('nm')
```

这里，args是包含表单参数对及其对应值对的列表的字典对象。与'nm'参数对应的值将像之前一样传递到'/ success' URL。



4.Flask 模板（Jinja2模版）

在大型应用中,把业务逻辑和表现内容放在一起,会增加代码的复杂度和维护成本.

- 模板其实是一个包含响应文本的文件,其中用占位符(变量)表示动态部分,告诉模板引擎其具体的值需要从使用的数据中获取
- 使用真实值替换变量,再返回最终得到的字符串,这个过程称为'渲染'
- **Flask 是使用 Jinja2 这个模板引擎来渲染模板（底层使用，flask函数内部包含）**

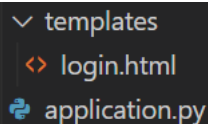
使用模板的好处

- 视图函数只负责业务逻辑和数据处理(业务逻辑方面)
- 而模板则取到视图函数的数据结果进行展示(视图展示方面)
- 代码结构清晰,耦合度低

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。Flask 会在 **templates 文件夹内寻找模板**。因此，如果你的应用是一个模块，那么模板文件夹应该在模块旁边；如果是一个包，那么就应该在包里面：

情形 1：一个模块：

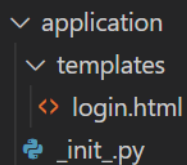
```
/application.py
/templates
  /hello.html
```



```
▼ templates
  <> login.html
  application.py
```

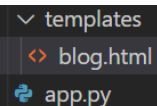
情形 2：一个包：

```
/application
  /__init__.py
  /templates
    /hello.html
```



```
▼ application
  ▼ templates
    <> login.html
    __init__.py
```

举例如下：



```
▼ templates
  <> blog.html
  app.py
```

app.py

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

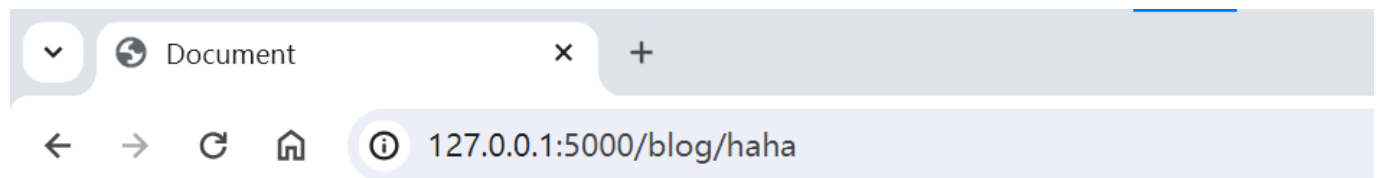
@app.route('/blog/<blog_id>')
def blog_detail(blog_id):
    return render_template("blog.html", blog_id=blog_id)

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

blog.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <p>
    这是博客详情: {{blog_id}}
  </p>
</body>
</html>
```

运行结果:



这是博客详情: haha

4.1 模板访问对象属性

app.py

```

from flask import Flask, render_template
app = Flask(__name__)

class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email

@app.route('/')
def hello_world():
    user = User('李四', 'lisi@qq.com')
    person = {
        "username": "张三",
        "email": "zhangsan@qq.com"
    }
    return render_template("index.html", user=user, person=person)

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

```

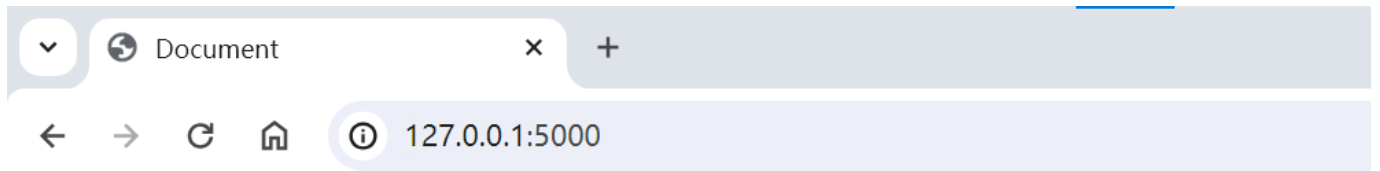
index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
</head>
<body>
    <p>
        类创建对象：
        这是用户姓名： {{user.username}}
        这是用户邮箱： {{user.email}}
    </p>
    <p>
        字典创建对象：
        这是用户姓名： {{person["username"]}}
        这是用户邮箱： {{person.email}}
    </p>
</body>
</html>

```

运行结果：

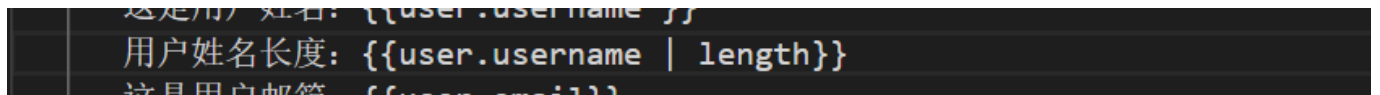


类创建对象：这是用户姓名：李四 这是用户邮箱：lisi@qq.com

字典创建对象：这是用户姓名：张三 这是用户邮箱：zhangsan@qq.com

4.2 模板过滤器

过滤器是通过管道符号 (|) 进行使用的，例如：{{ name|length }}，将返回name的长度。



过滤器相当于是一个函数，把当前的变量传入到过滤器中，然后过滤器根据自己的功能，再返回相应的值，之后再将结果渲染到页面中。Jinja2中内置了许多过滤器

| | | | | |
|----------------------------------|-------------------------------|------------------------------|------------------------------|-----------------------------|
| abs() | float() | lower() | round() | tojson() |
| attr() | forceescape() | map() | safe() | trim() |
| batch() | format() | max() | select() | truncate() |
| capitalize() | groupby() | min() | selectattr() | unique() |
| center() | indent() | pprint() | slice() | upper() |
| default() | int() | random() | sort() | urlencode() |
| dictsort() | join() | reject() | string() | urlize() |
| escape() | last() | rejectattr() | striptags() | wordcount() |
| filesizeformat() | length() | replace() | sum() | wordwrap() |
| first() | list() | reverse() | title() | xmlattr() |

现对一些常用的过滤器进行讲解：

- `abs(value)`：返回一个数值的绝对值。 例如：-1|abs。
- `default(value,default_value,boolean=False)`：如果当前变量没有值，则会使用参数中的值来代替。name|default('xiaotuo')——如果name不存在，则会使用xiaotuo来替代。boolean=False默认是在只有这个变量为undefined的时候才会使用default中的值，如果想使用python的形式判断是否为false，则可以传递boolean=true。也可以使用or来替换。

- `escape(value)`或`e`：转义字符，会将<、>等符号转义成HTML中的符号。例如：`content|escape`或`content|e`。
- `first(value)`：返回一个序列的第一个元素。`names|first`。
- `format(value,*arags,**kwargs)`：格式化字符串。例如以下代码：

```
{{ "%s" - "%s"|format('Hello?',"Foo!") }}
```

将输出：Hello? - Foo!

- `last(value)`：返回一个序列的最后一个元素。示例：`names|last`。
- `length(value)`：返回一个序列或者字典的长度。示例：`names|length`。
- `join(value,d=u'')`：将一个序列用d这个参数的值拼接成字符串。
- `safe(value)`：如果开启了全局转义，那么safe过滤器会将变量关掉转义。示例：`content_html|safe`。
- `int(value)`：将值转换为int类型。
- `float(value)`：将值转换为float类型。
- `lower(value)`：将字符串转换为小写。
- `upper(value)`：将字符串转换为小写。
- `replace(value,old,new)`：替换将old替换为new的字符串。
- `truncate(value,length=255,killwords=False)`：截取length长度的字符串。
- `striptags(value)`：删除字符串中所有的HTML标签，如果出现多个空格，将替换成一个空格。
- `trim`：截取字符串前面和后面的空白字符。
- `string(value)`：将变量转换成字符串。
- `wordcount(s)`：计算一个长字符串中单词的个数。

4.3 自定义过滤器

app.py

```

from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)

class User:
    def __init__(self, username, email):
        self.username = username
        self.email = email

def datetime_format(value, format="%Y-%m-%d %H:%M"):
    return value.strftime(format)

app.add_template_filter(datetime_format, 'dformat')

@app.route('/')
def hello_world():
    user = User('李四', 'lisi@qq.com')
    mytime=datetime.now()
    return render_template("index.html", user=user, mytime=mytime)

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

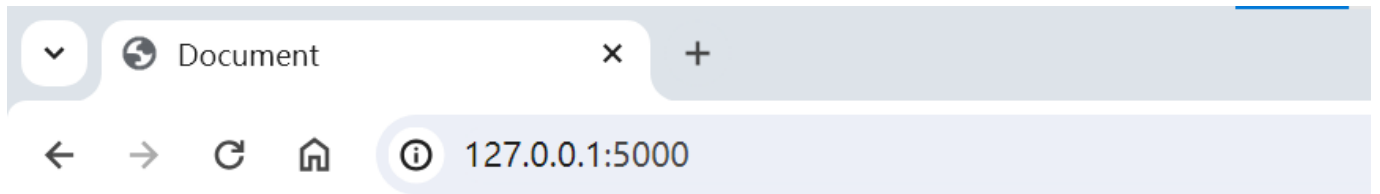
```

index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Document</title>
</head>
<body>
    <p>
        类创建对象: <br>
        这是用户姓名: {{user.username }}<br>
        用户姓名长度: {{user.username | length}}<br>
        这是用户邮箱: {{user.email}}<br>
        现在的时间为: {{mytime | dformat}}<br>
    </p>
</body>
</html>

```



类创建对象：

这是用户姓名：李四

用户姓名长度：2

这是用户邮箱：lisi@qq.com

现在的时间为：2024-02-29 16:44

4.4 控制语句

所有的控制语句都是放在{% ... %}中，并且有一个语句{% endxxx %}来进行结束，Jinja中常用的控制语句有if/for..in..，现对他们进行讲解：

1. if: if语句和python中的类似，可以使用>，<，<=，>=，==，!=来进行判断，也可以通过and，or，not，()来进行逻辑合并操作，以下看例子：

app.py

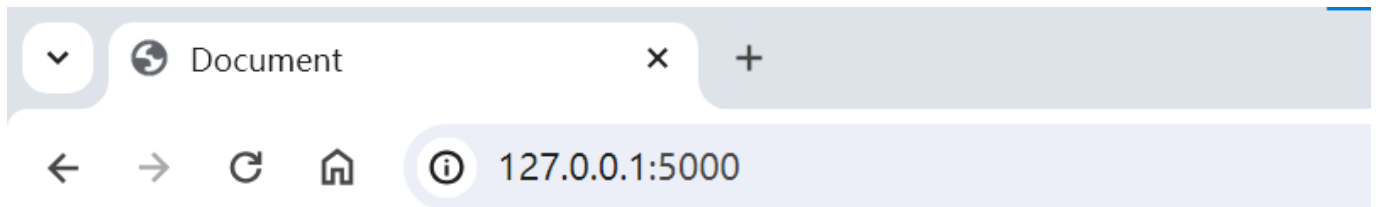
```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)

@app.route("/")
def control_statement():
    age=17
    return render_template("index.html",age=age)
if __name__ == '__main__':
    app.run(debug = True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  {% if age>18 %}
  <div>已满18周岁</div>
  {% elif age==18%}
  <div>刚满18周岁</div>
  {% else %}
  <div>未满18周岁</div>
  {% endif %}
</body>
</html>
```

运行结果：



未满18周岁

2. for...in...: for循环可以遍历任何一个序列包括列表、字典、元组。并且可以进行反向遍历，以下将用几个例子进行解释：

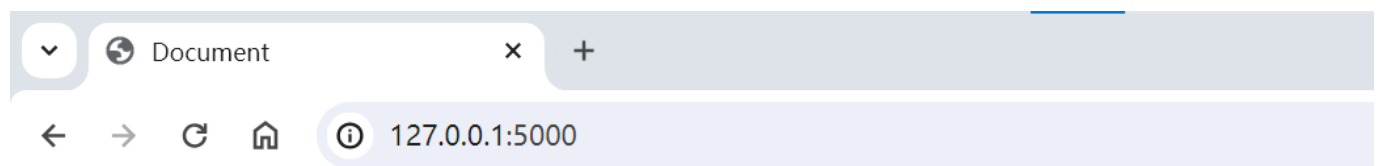
app.py

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)

@app.route("/")
def control_statement():
    books=[{"name":"红楼梦","author":"曹雪芹"},
           {"name":"水浒传","author":"施耐庵"}]
    return render_template("index.html",books=books)
if __name__ == '__main__':
    app.run(debug = True)
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  {% for book in books %}
    <div>书名: {{book.name}}; 作者名: {{book.author}}</div>
  {% endfor %}
</body>
</html>
```



书名: 红楼梦; 作者名: 曹雪芹
书名: 水浒传; 作者名: 施耐庵

不可以使用continue和break表达式来控制循环的执行。

4.5 测试器

测试器主要用来判断一个值是否满足某种类型，并且这种类型一般通过普通的if判断是有很大的挑战的。语法是：if...is...

```

{# 检查变量是否被定义，也可以用undefined检查是否未被定义 #}
{% if name is defined %}
    <p>Name is: {{ name }}</p>
{% endif %}

{# 检查是否所有字符都是大写 #}
{% if name is upper %}
    <h2>"{{ name }}" are all upper case.</h2>
{% endif %}

{# 检查变量是否为空 #}
{% if name is none %}
    <h2>Variable is none</h2>
{% endif %}

{# 检查变量是否为字符串，也可以用number检查是否为数值 #}
{% if name is string %}
    <h2>{{ name }} is a string.</h2>
{% endif %}

{# 检查变量是否可被迭代循环，也可以用sequence检查是否是序列 #}
{% if [1,2,3] is iterable %}
    <h2>Variable is iterable.</h2>
{% endif %}

{# 检查变量是否是字典 #}
{% if {'name':'test'} is mapping %}
    <h2>Variable is dict.</h2>
{% endif %}

```

| 测试器 | 说明 |
|-------------------------------|-----------|
| <code>callable(object)</code> | 是否可调用 |
| <code>defined(object)</code> | 是否已经被定义了。 |
| <code>escaped(object)</code> | 是否已经被转义了。 |
| <code>upper(object)</code> | 是否全是大写。 |
| <code>lower(object)</code> | 是否全是小写。 |
| <code>string(object)</code> | 是否是一个字符串。 |
| <code>sequence(object)</code> | 是否是一个序列。 |
| <code>number(object)</code> | 是否是一个数字。 |
| <code>odd(object)</code> | 是否是奇数。 |
| <code>even(object)</code> | 是否是偶数。 |

4.6 宏和import语句

4.6.1 宏

模板中的宏跟python中的函数类似，可以传递参数，但是不能有返回值，可以将一些经常用到的代码片段放到宏中，然后把一些不固定的值抽取出来当成一个变量，以下将用一个例子来进行解释：

```
{% macro input(name, value='', type='text') %}  
    <input type="{{ type }}" name="{{ name }}" value="{{ value|e }}">  
{% endmacro %}
```

以上例子可以抽取出了一个input标签，指定了一些默认参数。那么我们以后创建input标签的时候，可以通过他快速的创建：

```
<p>{{ input('username') }}</p>  
<p>{{ input('password', type='password') }}</p>
```

4.6.2 import语句

在真实的开发中，会将一些常用的宏单独放在一个文件中，在需要使用的时候，再从这个文件中进行导入。import语句的用法跟python中的import类似，可以直接import...as...，也可以from...import...或者from...import...as...，假设现在有一个文件，叫做forms.html，里面有两个宏分别为input和textarea，如下：

```
{% macro input(name, value='', type='text') %}  
    <input type="{{ type }}" value="{{ value|e }}" name="{{ name }}">  
{% endmacro %}  
  
{% macro textarea(name, value='', rows=10, cols=40) %}  
    <textarea name="{{ name }}" rows="{{ rows }}" cols="{{ cols  
    }}">{{ value|e }}</textarea>  
{% endmacro %}
```

4.6.3 导入宏的例子

1.import...as...形式：


```
{% import 'forms.html' as forms %}
<dl>
  <dt>Username</dt>
  <dd>{{ forms.input('username') }}</dd>
  <dt>Password</dt>
  <dd>{{ forms.input('password', type='password') }}</dd>
</dl>
<p>{{ forms.textarea('comment') }}</p>
```

2.from...import...as.../from...import...形式:

```
{% from 'forms.html' import input as input_field, textarea %}
<dl>
  <dt>Username</dt>
  <dd>{{ input_field('username') }}</dd>
  <dt>Password</dt>
  <dd>{{ input_field('password', type='password') }}</dd>
</dl>
<p>{{ textarea('comment') }}</p>
```

另外需要注意的是，导入模板并不会把当前上下文中的变量添加到被导入的模板中，如果你想要导入一个需要访问当前上下文变量的宏，有两种可能的方法：

显式地传入请求或请求对象的属性作为宏的参数。

与上下文一起（with context）导入宏。

```
{% from '_helpers.html' import my_macro with context %}
```

4.7 include和set语句

4.7.1 include语句

include语句可以把一个模板引入到另外一个模板中，类似于把一个模板的代码copy到另外一个模板的指定位置

```
{% include 'header.html' %}
  主体内容
{% include 'footer.html' %}
```

4.7.2 赋值（set）语句

有时候我们想在模板中添加变量，这时候赋值语句（set）就派上用场了

```
{% set name='zhiliao' %}
```

那么以后就可以使用name来代替zhiliao这个值了，同时，也可以给他赋值为列表和元组：

```
{% set navigation = [('index.html', 'Index'), ('about.html', 'About')] %}
```

赋值语句创建的变量在其之后都是有效的，如果不想让一个变量污染全局环境，可以使用with语句来创建一个内部的作用域，将set语句放在其中，这样创建的变量只在with代码块中才有效

```
{% with %}
    {% set foo = 42 %}
    {{ foo }}          foo is 42 here
{% endwith %}
```

也可以在with的后面直接添加变量，比如以上的写法可以修改成这样：

```
{% with foo = 42 %}
    {{ foo }}
{% endwith %}
```

这两种方式都是等价的，一旦超出with代码块，就不能再使用foo这个变量了。

4.8 模版继承

Flask中的模板可以继承，通过继承可以把模板中许多重复出现的元素抽取出来，放在父模板中，并且父模板通过定义block给子模板开一个口，子模板根据需要，再实现这个block，假设现在有一个index.html这个父模板，代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>    {% block title %}{% endblock%}</title>
</head>
<body>
    {% block body %}
    {% endblock%}
</body>
</html>
```

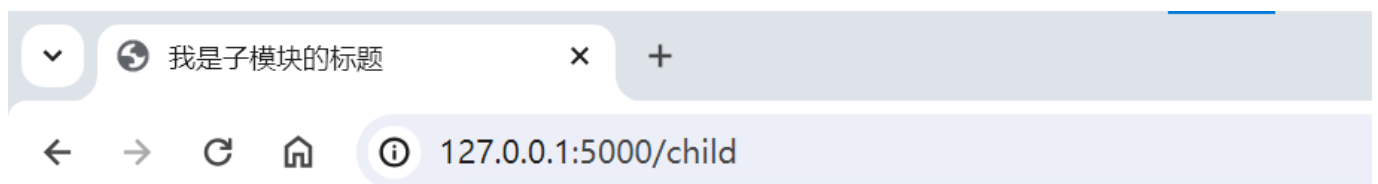
同时对于一些子模板需要重写的地方，比如title、head、body都定义成了block，然后子模板可以根据自己的需要，再具体的实现。以下再来看子模板的代码：

```
{% extends "index.html"%}

{% block title %}
我是子模块的标题
{% endblock %}

{% block body %}
子模块的body
{% endblock %}
```

首先第一行就定义了子模板继承的父模板，并且可以看到子模板实现了title和body这些block
运行结果：



子模块的body

4.9 转义

转义的概念是，在模板渲染字符串的时候，字符串有可能包括一些非常危险的字符比如<、>等，这些字符会破坏掉原来HTML标签的结构，更严重的可能会发生XSS跨域脚本攻击，因此如果碰到<、>这些字符的时候，应该转义成HTML能正确表示这些字符的写法，比如>在HTML中应该用<来表示等。

但是Flask中默认没有开启全局自动转义，针对那些以.html、.htm、.xml和.xhtml结尾的文件，如果采用render_template函数进行渲染的，则会开启自动转义。并且当用render_template_string函数的时候，会将所有的字符串进行转义后再渲染。而对于Jinja2默认没有开启全局自动转义，作者有自己的原因：

1.渲染到模板中的字符串并不是所有都是危险的，大部分还是没有问题的，如果开启自动转义，那么将会带来大量的不必要的开销。 2.Jinja2很难获取当前的字符串是否已经被转义过了，因此如果开启自动转义，将对一些已经被转义过的字符串发生二次转义，在渲染后会破坏原来的字符串。在没有开启自动转义的模式下（比如以.conf结尾的文件），对于一些不信任的字符串，可以通过{{ content_html|e }}或者是{{ content_html|escape }}的方式进行转义。在开启了自动转义的模式下，如果关闭自动转义，可以通过{{ content_html|safe }}的方式关闭自动转义。而{%autoescape true/false%}...{%endautoescape%}可以将一段代码块放在中间，来关闭或开启自动转义，例如以下代码关闭了自动转义：

```
{% autoescape false %}
<p>autoescaping is disabled here
<p>{{ will_not_be_escaped }}
{% endautoescape %}
```

4.10 数据类型和运算符

4.10.1 数据类型

Jinja支持许多数据类型，包括：字符串、整型、浮点型、列表、元组、字典、True/False。

4.10.2 运算符

- `+` 号运算符：可以完成数字相加，字符串相加，列表相加。但是并不推荐使用 `+` 运算符来操作字符串，字符串相加应该使用 `~` 运算符。
- `-` 号运算符：只能针对两个数字相减。
- `/` 号运算符：对两个数进行相除。
- `%` 号运算符：取余运算。
- `*` 号运算符：乘号运算符，并且可以对字符串进行相乘。
- `**` 号运算符：次幂运算符，比如 `2**3=8`。
- `in` 操作符：跟python中的 `in` 一样使用，比如 `{{1 in [1,2,3]}}` 返回 `true`。
- `~` 号运算符：拼接多个字符串，比如 `{{"Hello" ~ "World"}}` 将返回 `HelloWorld`。

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)

@app.route("/index")
def index():
    a=2
    b=3
    return render_template("index.html",a=a,b=b)

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

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>文档</title>
</head>
<body>
  {% autoescape false %}
  <p> {{a+b}}</p>
  {% endautoescape %}

</body>
</html>
```

4.11 静态文件的配置

Web应用中会出现大量的静态文件来使得网页更加生动美观。类似于CSS样式文件、JavaScript脚本文件、图片文件、字体文件等静态资源。在Jinja中加载静态文件非常简单，只需要通过url_for全局函数就可以实现，看以下代码：

```
<link href="{{ url_for('static',filename='about.css') }}">
```

url_for函数默认会在项目根目录下的static文件夹中寻找about.css文件，如果找到了，会生成一个相对于项目根目录下的/static/about.css路径。当然我们也可以把静态文件不放在static文件夹中，此时就需要具体指定了，看以下代码：

```
app = Flask(__name__,static_folder='C:\static')
```

那么访问静态文件的时候，将会到/static这个文件夹下寻找。

举例如下：

app.py

```
from flask import Flask, render_template
from datetime import datetime
app = Flask(__name__)

@app.route("/")
def index():
    return render_template("index.html")

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

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>文档</title>
  <link rel="stylesheet"
        href="{{url_for('static',filename='css/style.css')}}">
  <script src="{{url_for('static',filename='js/my.js')}}"></script>
</head>

<body>

</body>
</html>
```

style.css

```
body{
  background-color: antiquewhite;
}
img{
  width: 300px;
  height: 120px;
}
```

my.js

```
alert("这是流浪地球的宣传海报");
```

运行结果：



