Flask框架（基础）

- 讲师：pansir

框架对比

Flask和Django是现在最流行的python-web框架

通常用途：django常用做开发常规网站，flask常用作api开发

	Flask	Django	Tornado
github星星(requests-41.7k)	49k	47.3k	18.8k
性能	中	中	高（40%）
框架特点	1. 轻量级，开发灵活 2. 扩展性极强 3. 不指定ORM，可用NoSQL	1. 刚性目录（可读性高） 2. 组件大而全（ORM组件/用户认证/权限管理/分页/缓存），但笨重 3. 可插拔式设计思想	1. 天然支持异步非阻塞处理方式 2. 抗负载能力强
文档与社区	文档多，社区活跃，增量多	文档超多，社区活跃	文档少，相对不活跃
应用场景	1. 轻量级网站 2. API服务/微服务 3. 尝试新技术	1. 传统企业级项目：电商/社交平台/办公OA 2. 方便团队协作开发与项目管理	
针对测试人员	推荐	不推荐	可选

基本概念

web框架：包含库和模块的集合，专注于功能，而不用关注底层协议，线程管理

flask：基于werkzeug WSGI工具包和Jinja2模块引擎

WSGI：web服务器和web应用程序之间的通用接口规范

werkzeug：WSGI工具包，实现请求，响应对象和使用函数

jinja2：模版引擎，通过模版+数据，组合成动态网页

Django太笨重，对应flask是一个轻量级框架，但支持各种扩展

开发环境

建议使用：pipenv来作为开发环境

第一个应用

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

@app.route('/')
def say_hello():
    return 'hello tester'

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

解释：

1. 导入Flask模块，`__name__`是当前模块的名称，通过`Flask(__name__)`即可得到一个`WSGI`应用程序
2. Flask的`route()`是一个装饰器函数，第一个参数为路由地址，即：url访问地址，也可以认为是接口名称。例如：`@app.route('/')`表示`'/'`url与`say_hello()`函数绑定，当有接口请求`/'`时，触发函数`say_hello()`并返回内容，函数的返回即http接口的返回
3. `app.run()`表示运行服务程序，即：启动一个进程，监听一个端口，默认为5000

启动服务

1. `app.run()`即运行服务程序，但它接收多个可选参数：`app.run(host, port, debug, options...)`
    - host：要监听的主机名称，默认127.0.0.1(localhost)，设置为：0.0.0.0可以使服务器在外部可以访问
    - port：http监听端口，默认是5000
    - debug：调试模式，默认是False，开发时设为True，但注意一定不要在生产使用
2. 要运行服务，直接`python xxx.py`，或者pycharm运行都可以

调试模式

开发过程中，每次改动代码，都需要重启服务才能生效，很不方便。可以使用调试模式启动，如果代码更改，服务器自动重新加载。一般是`ctrl + s`保存后会立即生效。设置调试模式的方法有两种：

```
# 1. 通过属性变量设置
app.debug = True
app.run()
# 2. 通过参数设置
app.run(debug = True)
```

添加路由

http请求通过url访问接口，服务端就是通过url来路由到对应到函数，url -> 路由 -> 函数。设定路由的方式有两种，推荐使用第一种

- 使用装饰器的参数直接指定

```
@app.route('/')
def say_hello():
    return 'hello tester'
```

- 使用`add_url_rule()`方法指定

```
def say_hello():
    return 'hello tester'
app.add('/', 'hello', say_hello)
```

接口变量

例如请求：http://0.0.0.0:5001/user/root/sign/44 有很多参数包含在接口名称里，注意，这里不是get参数。接口服务应该怎么接收这些参数呢?

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

@app.route('/hello/<name>')
def say_hello(name):
    print(name, type(name))
    return 'hello %s!' % name

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

解释：装饰器通过`<name>`来定义一个变量，用来匹配url，例如浏览器打开：http://127.0.0.1:5000/hello/pan name则为：pan，注意：接口函数必须传入对应的变量

变量转换器

当我们需要获取指定类型的数据时，可以使用`<type: name>`语法进行强制转换，如果类型转换失败，则会返回404

```python
from flask import Flask

app = Flask(__name__)


@app.route('/getint/<int:sign_id>')
def getint(sign_id):
    print(sign_id, type(sign_id))
    return 'sign_id is %d' % sign_id


@app.route('/getfloat/<float:ft>')
def getfloat(ft):
    print(ft, type(ft))
    return 'ft is %f' % ft


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

路由规范

定义路由的时候一定要注意规范，尽量要以`/`结尾，比如使用`/rule/`，而不是`/rule`

```python
from flask import Flask

app = Flask(__name__)


@app.route('/rule')
def rule_1():
    return 'Hello rule_1'


@app.route('/rule/')
def rule_2():
    return 'Hello rule_2'


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

解释说明：

1. 当两者同时使用时，各自生效。即：使用`/rule`访问，会路由到`roule_1`；使用`/rule/`访问会路由到`roule_2`

2. 当仅使用：`/rule`时。使用`/rule`访问，会路由到`roule_1`；使用`/rule/`访问会报错404
3. 当仅使用：`/rule/`时。使用`/rule/`访问，会路由到`roule_2`；使用`/rule`访问，会重定向到`/rule/`，从而也访问`roule_2`

所以从规范上讲，使用`/`结尾的路由，兼容性更好

url构建

实际使用中，跳转url时不会直接使用url地址，而是使用url_for方法找到对应的url。url_for接收一个函数名称(字符串)，通过这个函数名称，找到与之绑定的url，发起请求

```
from flask import Flask, redirect, url_for
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s:')

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):
    logging.info("[name is]: {}".format(name))
    # 根据url参数控制逻辑分支
    if name == 'admin':
        # return redirect("http://127.0.0.1:5000/admin")
        return redirect(url_for('hello_admin'))
    else:
        return redirect(url_for('hello_guest', guest=name))


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

浏览器分别请求：

`http://127.0.0.1:5000/user/admin`

`http://127.0.0.1:5000/user/pansir`

解释说明：

1. `hello_user`是个入口函数，例如我们请求`/user/admin`时，会跳转到`hello_admin()`方法，跳转可以直接使用url地址：`http://127.0.0.1:5000/admin`，但是一般不这样使用，而是使用`url_for("hello_admin")`找到对应的接口
2. `url_for()`在构造url的时候，也可以传递参数，例如：`url_for('hello_guest', guest=name)`，需要注意：`hello_guest()`必须接收该参数

HTTP方法

5种http请求方法：get/post/put/delete/head都支持，使用方法类似，重点关注：get和post请求处理

将下面内容保存到`login.html`文件中

```
<html>
<body>

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

</body>
</html>
```

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s %(message)s')
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=["GET", "POST"])
def login():
    if request.method == 'POST':
        logging.info("[请求表单是]: {}".format(request.form))
        # 处理post请求
        user = request.form['name']
        logging.error("[从表单取出name值]: {}".format(user))
```

```
        # 重定向到success()
        # 1. 重定向的地址  2. 参数
        return redirect(url_for('success', name=user))
    else:
        # 处理其他(get)请求
        logging.info("[get请求参数]: {}".format(request.args))
        # 必须传入name参数
        user = request.args.get('name')
        return redirect(url_for('success', name=user))


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

- 模拟get请求：`http://127.0.0.1:5000/login/gugu`，发现重定向到：`http://127.0.0.1:5000/login/gugu`
- 模拟post请求：
  1. 用浏览器打开login.html文件
  2. 输入框输入名称：gugu 后，点击`submit`
  3. 重定向到：`http://127.0.0.1:5000/login/gugu`

解释说明：

1. 在装饰器内指定参数：`methods=['POST', 'GET']`来指定接收哪些http请求方法，若使用限定外的请求方法，则默认报错`405 method not allowed`；若不传methods参数，则默认只接收get请求
2. 通过`request.method`属性来获取客户端的请求方法，值为字符串类型
3. 通过`request.form`获取post表单数据，类型为`ImmutableMultiDict`，支持字典取值方式：`request.form['name']`
4. 通过`request.args.get('name')`获取get请求参数，与3类似

请求数据与接收数据（重点）

四种类型

我们使用http请求，带参数内容

1. args：一个包含解析过的查询字符串（ URL 中问号后的部分）内容的 MultiDict 。
2. data：如果进入的请求数据是 服务端 不能处理的 mimetype ， 数据将作为字符串存于此。
3. form：一个包含解析过的从 POST 或 PUT 请求发送的表单对象的 MultiDict 。**请注意上传的文件不会在这里，而是在 files 属性中。**
4. files：一个包含 POST 和 PUT 请求中上传的文件的 MultiDict 。每个文件存储为 FileStorage 对象。其基本的行为类似你在 Python 中见到的标准文件对象，差异在于这个对象有一个 save() 方法可以把文件存储到文件系统上。

args参数

- requests（客户端）发送请求带args参数

```
import requests
# get请求只有args
url = "http://httpbin.org/get?a=1"
r = requests.get(url=url)
print(r.text)

# post请求也有可带args
url = "http://httpbin.org/post?a=1"
r = requests.post(url=url)
print(r.text)

# 使用params带args参数
url = "http://httpbin.org/get"
params = {"a": 1}
r = requests.get(url=url, params=params)
print(r.text)
```

- flask接收args参数

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

from flask import Flask, request

app = Flask(__name__)


@app.route("/test/")
def get_args():
    logging.info("[请求方式]: {}".format(request.method))
    logging.info("[请求args]: {}".format(request.args))
    logging.info("[尝试获取args中的name参数]:
{}".format(request.args.get("name")))
    # 没有return则会报错
    return request.args


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

```
import requests

url = "http://127.0.0.1:5000"
```

```
import requests

# get请求只有args
url = "http://127.0.0.1:5000/test/?a=1"
r = requests.get(url=url)
print(r.text)

# post请求也有可带args
url = "http://127.0.0.1:5000/test/?a=1"
r = requests.post(url=url)
print(r.text)

# 使用params带args参数
url = "http://127.0.0.1:5000/test/"
params = {"a": 1}
r = requests.get(url=url, params=params)
print(r.text)
```

- 解释：request.args会获取客户端提交的args参数，具体对象是：`ImmutableMultiDict`，可以通过`request.args.get("name")`取值
- 小结：客户端请求的args参数，flask可以通过`request.args`获取

data参数

data参数即普通参数。data参数可以传字符串，也可以传字典

```
import requests
# post请求也有可带args
url = "http://httpbin.org/post?a=1"
# 1. 直接传字符串，data=data，服务端会识别成为 data 参数，内容解析为字符串
data = "hello"
r = requests.post(url=url, data=data)
print(r.text)

# 2. 直接传字典，data=data，因为requests库默认使用application/x-www-form-urlencoded，客户端会处理成form
# 服务端会识别到 form，内容解析为字典，所以识别data为空，注意json是为空的!
data = {"name": "hello"}
r = requests.post(url=url, data=data)
print(r.text)

# 3. 通过json=data关键字传参数，服务端会识别 data 参数，但是是json串，同时json内容识别正常
data = {"name": "hello"}
r = requests.post(url=url, json=data)
print(r.text)
```

- flask接收data参数

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

from flask import Flask, request

app = Flask(__name__)


@app.route("/test/", methods=["GET", "POST"])
def get_args():
    logging.info("[请求方式]: {}".format(request.method))
    logging.info("[请求args]: {}".format(request.args))
    # 1. requests客户端直接提交的字符串数据，会识别成为bytes数据，如果需要字符串，需要自行
解码
    # 2. requests客户端直接提交的字典数据，request.data会识别为 b""
    logging.info("[请求data]: {} {}".format(request.data, type(request.data)))
    logging.info("[data数据解码后结果]: {}".format(request.data.decode()))
    logging.info("[尝试获取data中的name参数]:
{}".format(request.args.get("name")))
    # 若没有取到则使用默认值：World
    logging.info(request.args.get("name", "World"))
    # 获取form数据
    logging.info("[请求form数据]: {} {}".format(request.form,
type(request.form)))
    logging.info("[尝试获取form中的name参数]:
{}".format(request.form.get("name")))
    # 获取json数据
    logging.info("[请求json数据]: {} {}".format(request.json,
type(request.json)))
    # 没有return则会报错
    return request.data


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

```
import requests

# post请求也有可带args
url = "http://127.0.0.1:5000/test/?a=1"
# 1. 直接传字符串，data=data，服务端会识别成为 data 参数，内容解析为字符串
data = "hello"
data = "你好".encode("utf-8")  # 直接发"你好"会报错，必须进行编码
```

```
r = requests.post(url=url, data=data)
# 获取到的数据会自动解码
print(r.text)

# 2. 直接传字典，data=data，flask识别不到客户端发的data数据，会将其识别为form参数
data = {"name": "hello"}
r = requests.post(url=url, data=data)
print(r.text)

# 3. 通过json关键字传参数，服务端会识别 data 参数，但是是json串，同时json内容识别正常
data = {"name": "hello"}
r = requests.post(url=url, json=data)
print(r.text)
```

- 解释：
    1. 服务端request.data可以获取到客户端发送的data字符串数据，但需注意，requests客户端发送的数据是经过编码的，服务端需要自行进行解码
    2. 服务端request.data接收不到requests发送的`data={"name": "hello"}`数据，原因是因为requests客户端，会把类型默认处理成为`application/x-www-form-urlencoded`。所以，服务端需要通过`request.form`来接收数据，该数据类型为`ImmutableMultiDict`，也可通过：`request.form.get("name")`来获取数据
    3. 如果客户端发送的是json数据，服务端通过`request.data`接收到的是bytes数据，需要经过解码 + 反序列化 才可获取到字典对象。若直接通过`request.json`来获取，可以直接解析成为python字典，而且不需要解码

form表单提交

- 提交表单的方式：
    1. data={} 方式，不带content-type，或使用application/x-www-form-urlencoded

```
import requests
url = "http://httpbin.org/post?a=1"
# 若传字典，则会被识别为 form，因为requests库默认使用application/x-www-form-urlencoded
data = {"name": "dcs"}
r = requests.post(url=url, data=data)
print(r.text)
```

2. 通过multipart/form-data(一般是用来提交文件)，构造一个MultipartEncoder来提交表单

```
from requests_toolbelt import MultipartEncoder

import requests

data = MultipartEncoder(
    fields={
```

```
        'field0': 'value',
        'field1': 'value',
    }
)
print(data.content_type)
# 必须添加content_type，否则会识别错误
r = requests.post('http://httpbin.org/post', data=data, headers={'Content-
Type': data.content_type})
print(r.text)
```

- flask接收form数据

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

from flask import Flask, request

app = Flask(__name__)


@app.route("/test/", methods=["GET", "POST"])
def get_args():
    logging.info("[请求方式]: {}".format(request.method))
    logging.info("[请求args]: {}".format(request.args))
    # 1. requests客户端直接提交的字符串数据，会识别成为bytes数据，如果需要字符串，需要自行
解码
    # 2. requests客户端直接提交的字典数据，request.data会识别为 b""
    logging.info("[请求data]: {} {}".format(request.data, type(request.data)))
    logging.info("[data数据解码后结果]: {}".format(request.data.decode()))
    logging.info("[尝试获取data中的name参数]:
{}".format(request.args.get("name")))
    # 获取form数据，如果form-data的key 与 args参数名重复，会优先取args参数
    logging.info("[请求form数据]: {} {}".format(request.form,
type(request.form)))
    logging.info("[尝试获取form中的name参数]:
{}".format(request.form.get("name")))
    # 获取json数据
    logging.info("[请求json数据]: {} {}".format(request.json,
type(request.json)))
    # 没有return则会报错
    return request.data


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

- 测试flask服务端

```
import requests
from requests_toolbelt import MultipartEncoder

url = "http://127.0.0.1:5000/test/?a=1"
# 若传字典，则会被识别为 form，因为requests库默认使用application/x-www-form-
urlencoded
data = {"name": "bird1"}
r = requests.post(url=url, data=data)
print(r.text)

# 构造MultipartEncoder对象传参
data = MultipartEncoder(
    fields={
        'name': 'bird2',
        'age': "13",
    }
)
print(data.content_type)
# 必须添加content_type，否则会识别错误
r = requests.post('http://127.0.0.1:5000/test/?a=1', data=data, headers=
{'Content-Type': data.content_type})
print(r.text)
```

文件上传

1. 通过files参数上传文件

```
import requests
# 参数说明：
# ('name', (None, 'dcs'))  1. files字段名  2. 文件名  3. 文件内容
# 1. 如果文件名为None，文件内容是一个字符串，则requests会将其视为 form内容发送
# 2. 如果文件名不为None，且文件内容为一个文件对象，则requests将其视为files 发送
files = [
    ('name', (None, 'dcs')),
    ("file_field_name", ("file_name", open("file.txt", "r")))  # "r" 方式 与
"rb"方式都可
]

url = "http://httpbin.org/post?a=1"
r = requests.post(url=url, files=files)
print(r.text)
```

2. 使用requests_toolbelt库，同时提交form和files

```
from requests_toolbelt import MultipartEncoder
```

```
import requests

data = MultipartEncoder(
    fields={
        'field0': 'value', # form内容
        'field1': 'value', # form内容
        'field2': ('filename', open('file.txt', 'rb')) # 文件内容，只可rb方式打开
    }
)


r = requests.post('http://httpbin.org/post', data=data, headers={'Content-
Type': data.content_type})
print(r.text)
```

- flask接收文件，并保存到服务端

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

from flask import Flask, request
import os

basedir = os.path.abspath(os.path.dirname(__name__))
# 当前目录下需要创建这个文件夹
path = basedir + "/upload/"
app = Flask(__name__)


@app.route("/test/", methods=["GET", "POST"])
def get_args():
    logging.info("[请求方式]: {}".format(request.method))
    logging.info("[请求args]: {}".format(request.args))
    # 接收文件内容
    logging.info("[接收文件]: {}".format(request.files))
    file_obj = request.files.get("file_field_name")
    logging.info("[尝试获取文件字段对应内容]: {}".format(file_obj))
    logging.info("[尝试获取文件名]: {}".format(file_obj.filename))
    # 读取文件内容，如果需要重复读取，需要调整指针file_obj.seek(0, 0)
    logging.info(file_obj.read())
    # 调整指针，才可重复读取
    file_obj.seek(0, 0)
    logging.info(file_obj.read())
    # 保存到服务端，仍然需要重新调整指针
    file_obj.seek(0, 0)
    file_path = path + file_obj.filename
```

```
    file_obj.save(file_path)
    return "ok"


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

```
import requests
from requests_toolbelt import MultipartEncoder

# 参数说明:
# ('name', (None, 'dcs'))  1. files字段名  2. 文件名  3. 文件内容
# 1. 如果文件名为None，文件内容是一个字符串，则requests会将其视为 form内容发送
# 2. 如果文件名不为None，且文件内容为一个文件对象，则requests将其视为files 发送
files = [
    ('name', (None, 'dcs')),
    ("file_field_name", ("file.txt", open("file.txt", "r")))  # "r" 方式 与
"rb"方式都可
]

url = "http://127.0.0.1:5000/test/?a=1"
r = requests.post(url=url, files=files)
print(r.text)

# 测试上传2
data = MultipartEncoder(
    fields={
        'field0': 'value',  # form内容
        'field1': 'value',  # form内容
        'file_field_name': ('file.txt', open('file.txt', 'rb'))  # 文件内容，只可
rb方式打开
    }
)
# 注意，是data参数
r = requests.post('http://127.0.0.1:5000/test/?a=123', data=data, headers=
{'Content-Type': data.content_type})
print(r.text)
```

解释:

1. 服务端通过`request.files.get("file_field_name")`获取file_obj对象，可通过`file_obj.filename`获取文件名，可通过`file_obj.read()`获取文件内容
2. 注意，注意，注意：文件内容被使用，包括打印输出，指针就会偏移，如果需要重复读取，或重复写入，则需要手动重制指针
3. `file_obj.save(file_path)`可将文件保存到服务端本地

request参数

request对象的重要属性包含：

- request.method：当前客户端请求方法
- request.form：字典对象，包含表单参数及其值的键和值对
- request.data：处理不了的就存在data里面
- request.args：`?`后面的参数内容
- request.values：包含所有客户端提交的参数
- request.cookies：字典对象，包含cookie名称和值
- request.files：客户端提交的文件相关数据
- request.headers：请求头信息

其他属性：

- request.stream：如果表单提交的数据没有以已知的 mimetype 编码，为性能考虑，数据会不经修改存储在这个流中
- request.environ：底层的 WSGI 信息
- request.url_root
- request.url_rule
- request.path
- request.url_charset

flask模版

直接return

我们直接返回字符串，浏览器打开是没有任何样式效果的。如果直接返回整个html内容，浏览器就可以看到渲染后的内容

```
from flask import Flask

app = Flask(__name__)


@app.route('/')
def index():
    return '''<h1>Hello gugu</h1>'''


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

问题：直接返回html的方式满足不了日常使用场景，例如：html内容需要参数化，重复html内容引用，条件判断，循环展示等等，所以需要使用Jinja2来管理模版

Jinja2

集合Jinja2模版引擎，flask可以通过`return_template()`来返回指定的模版，同时也可以进行参数化

在脚本目录下，创建一个文件夹`templates`(注意：文件名要完全一致)，然后新增一个`hello.html`文件，内容如下：

```
<!DOCTYPE html>
<html lang="en">
<h1>Hello {{ name }}!</h1>
</html>
```

```
from flask import Flask, render_template

app = Flask(__name__)


@app.route('/user/<name>')
def index(name):
    return render_template('hello.html', name=name)


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

浏览器打开：`http://127.0.0.1:5000/user/gugu`

解释说明：

1. 当使用`render_template()`方法时，flask会从当前目录下寻找`templates`文件夹，然后寻找该文件夹下的模版文件
2. `render_template()`指定我们要使用的模版，并且可以传入参数，例如：`render_template('hello.html', name=name)`。表示：将变量`name`的值代入到模版中，然后将结果返回给客户端
3. 模版中使用了`{{ name }}`，表示一个占位符，用来接收name参数，所以调用render_template方法传入到参数名，要和模版中的占位参数名一致

Jinja2语法

条件判断

html内容：

```
<!DOCTYPE html>
<html lang="en">
{% if score>59 %}
<h1> 恭喜，成绩合格!</h1>
{% else %}
<h1> 成绩不合格! </h1>
{% endif %}
</html>
```

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s %(message)s')
from flask import Flask, render_template

app = Flask(__name__)


@app.route('/score/<int:score>')
def hello_name(score):
    return render_template('score.html', score=score)


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

浏览器请求：

- 不合格：`http://127.0.0.1:5000/score/59`
- 合格：`http://127.0.0.1:5000/score/60`

解释说明：

1. jinja2有自己的一套语法，语法内容和python有些微差别，所有的语法相关内容都必须用转义符号包起来：`{% %}`
2. 接口接收参数的时候，要注意变量类型，如果类型有误，模版也会渲染失败

循环渲染

html内容：

```
<!DOCTYPE html>
<html lang="en">
<table border="1">
    {% for key, value in result.items() %}
    <tr>
        <th>
            {{ key }}
```

```
        </th>
        <td>
            {{ value }}
        </td>
    </tr>
    {% endfor %}
</table>
</html>
```

```
import logging

logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')
from flask import Flask, render_template

app = Flask(__name__)


@app.route('/students/')
def result():
    dict = {
        'ld': 90,
        'scl': 90,
        'ljh': 90,
        'qxl': 90,
        'xp': 90,
        'hhc': 90,
        'wjq': 90,
    }
    return render_template('students.html', result=dict)


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

浏览器打开：http://127.0.0.1:5000/students/ ；即可看到效果

解释说明：for循环语句需要用`{% %}`包裹起来，for循环有终止语句：`endfor`

练习：将各自账户下的sign_id查询出来，并展示其审核状态：

sign_id	review_status
424	pass