4. Web框架：Flask

- Web框架
- Flask
- 使用div + css规范网页样式
- 网页 + 图片搜索引擎

Part A: Flask 框架简介

Web框架

Web框架的用途、定义与优势

- Web开发的用途：（1）前端网页 （2）后端数据库 （3）Query & Request

- 框架的定义：框架，即 Framework。其实就是某种应用的半成品，把不同应用程序中有共性的一些东西抽取出来，做成一个半成品程序，这样的半成品就是所谓的程序框架。

- 好处：减少重复开发工作量、缩短开发时间、降低开发成本。同时还有其它的好处，如：使程序设计更合理、程序运行更稳定等。

- 常用的Web框架：Django, CubicWeb, Web2py, Weppy, Zope2, Bottle, CherryPy, Falcon, Flask, Pyramid, Tornado

Flask框架

Flask的简要介绍

- Flask是一个轻量级的Web应用框架，使用Python编写。基于 WerkzeugWSGI 工具箱和 Jinja2 模板引擎，使用 BSD 授权。

- Flask也被称为“Microframework”，因为它使用简单的核心，用 extension 增加其他功能。Flask没有默认使用的数据库、窗体验证工具。然而，Flask保留了扩增的弹性，可以用 Flask-extension 加入这些功能：ORM、窗体验证工具、文件上传、各种开放式身份验证技术。

- Flask 很轻，花很少的成本就能够开发一个简单的网站。非常适合初学者学习。Flask 框架学会以后，可以考虑学习插件的使用。例如使用 WTForm + Flask-WTForm 来验证表单数据，用 SQLAlchemy + Flask-SQLAlchemy 来对你的数据库进行控制。

Flask例程

Flask编写 Hello World 的一个简单示例

- 创建一个文件app.py

（1）引入Flask类：from flask import Flask

（2）创建Flask对象，我们将使用该对象进行应用的配置和运行：app = Flask(__name__)

注：name是Python中的特殊变量，如果文件作为主程序执行，那么__name__变量的值就是__main__，如果是被其他模块引入，那么__name__的值就是模块的名称。

（3）在主程序中，执行run()来启动应用

```
if __name__ =="__main__":
    app.run(debug=True, port=8080)
```

改名启动一个本地服务器，默认情况下其地址是localhost:5000，在上面的代码中，我们使用关键字参数Port将监听端口修改为8080。

（4）使用app变量的route()装饰器来告诉Flask框架URL如何触发我们的视图函数：

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

Flask例程

- 完整的app.py代码：

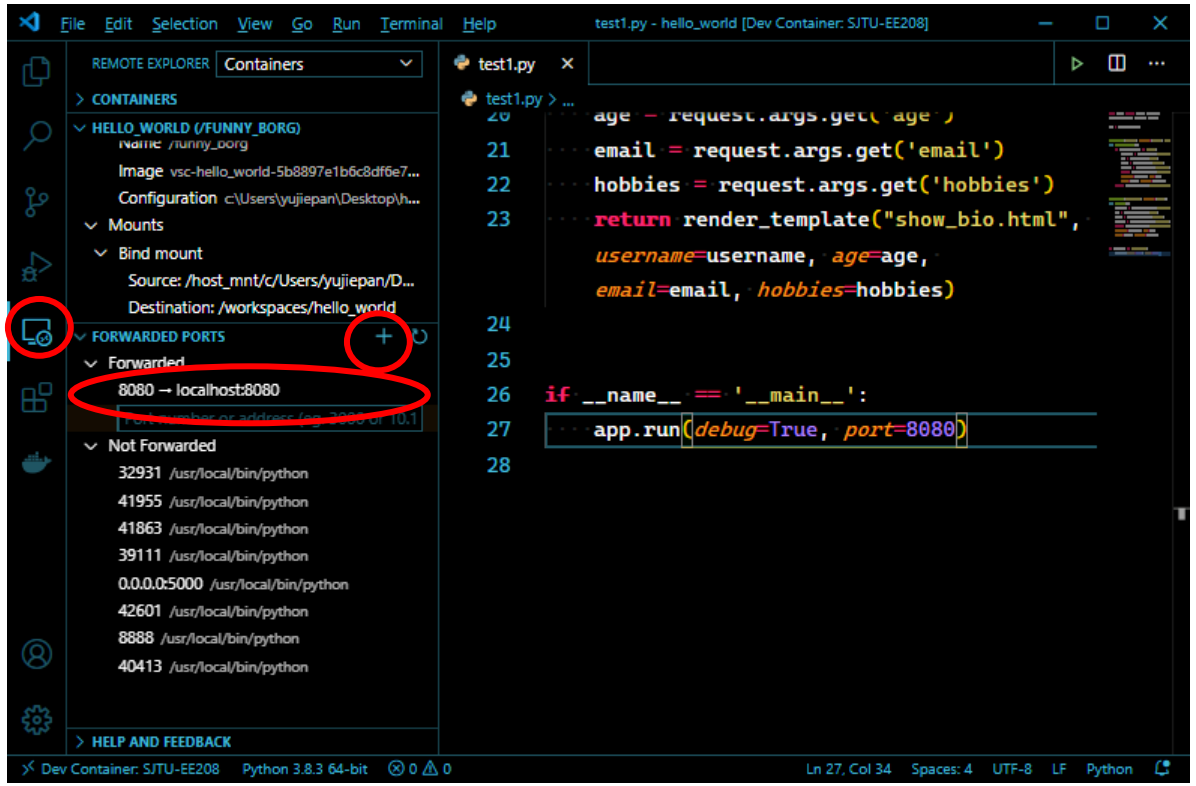
```
from flask import Flask
app = Flask(__name__)

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

if __name__ =="__main__":
    app.run(debug=True,port=8080)
```

- 运行后看到如下输入：

```
* Serving Flask app "app" (lazy loading)
* Environment: production
* Debug mode: on
* Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 770-937-705
```

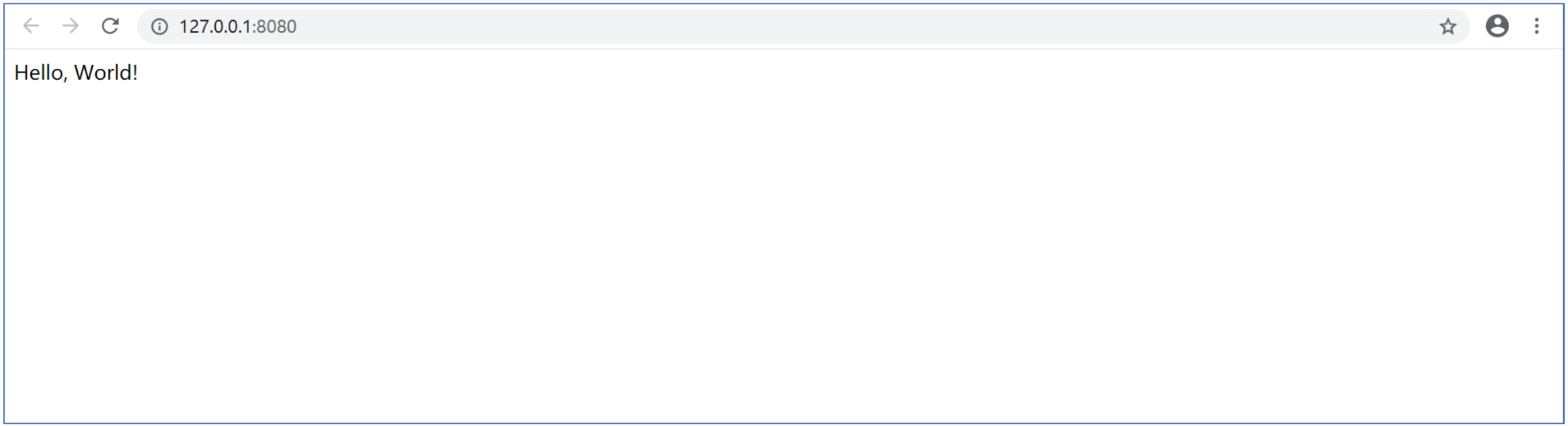
- 此时，docker容器系统将监听8080端口。为了能让本机（你的笔记本）也访问到8080端口，需要进行端口转发。（具体原理将在计算机网络课程中学习）

- 做法：点击加号添加“8080”



Flask例程

- 打开浏览器访问http://127.0.0.1:8080/:



Flask模板

Flask模板的简要介绍

- 模板的定义：视图函数的主要作用是生成请求的响应，这是最简单的请求。实际上，视图函数有两个作用：处理业务逻辑和返回响应内容。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。本节学到的模板，它的作用即是承担视图函数的另一个作用，即返回响应内容。

（1）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。

（2）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”。

（3）Flask是使用 Jinja2 这个模板引擎来渲染模板

- 使用模板的好处：

（1）视图函数只负责业务逻辑和数据处理(业务逻辑方面)

（2）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)

（3）代码结构清晰，耦合度低

- Flask提供的render_template函数封装了Jinjia2这一模板引擎。

Flask模板

Flask模板的使用方法

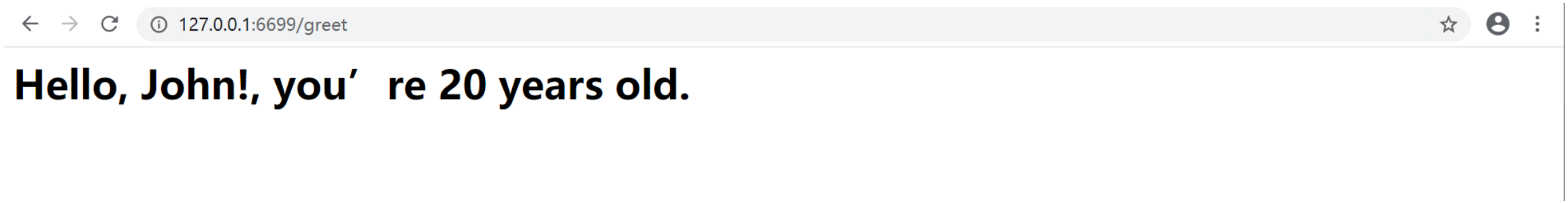
- 首先我们看看如何将原始的HTML代码插入Flask应用：

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

@app.route('/greet')
def greet():
    user = {'username': 'John', 'age': "20"}
    return '''
<html>
    <head>
        <title>Templating</title>
    </head>
    <body>
        <h1>Hello, ''' + user['username'] + '''!, you're ''' + user['age'] + ''' years old.</h1>
    </body>
</html>'''


if __name__ == '__main__':
    app.run(debug = True,port=6699)
```

- 在上面的代码中，我们使用拼接的HTML字符串来展示user字典的数据。

Flask模板

- 现在访问http://127.0.0.1:6699/greet（同样，需要按照PPT第5页设置forward port）



- 拼接HTML字符非常容易出错，因此Flask使用Jinja2模板引擎来分离数据逻辑层和展示层。

- 我们将模板文件按如下路径放置：

```
Apps folder
/app.py
templates
    |-/index.html
```

即在app.py所在文件夹的根目录中新建一个templates的文件夹，将模板文件index.html放入该文件夹中

Flask模板

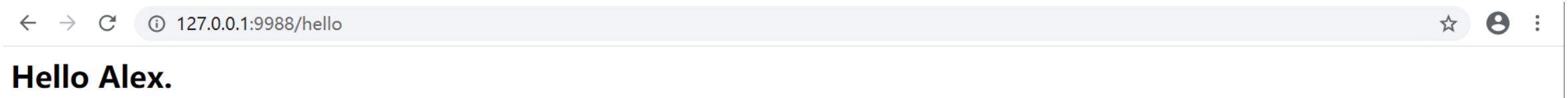
- 使用模板时，视图函数应当返回render_template（）的调用结果。例如下面的代码片段渲染模板index.html，并将渲染结果作为视图函数的返回值：

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

@app.route('/hello')
def hello():
    return render_template('index.html', name="Alex")

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

```
<html>
<body>
  {% if name %}
    <h2>Hello {{ name }}.</h2>
  {% else %}
    <h2>Hello.</h2>
  {% endif %}
 </body>
</html>
```

- 在上面的代码中，模板文件index.html依赖于变量name，模板文件的内容如右上所示：

- 模板文件的语法扩充了HTML，因此可以使用变量和逻辑。
- 在浏览器中访问http://127.0.0.1:9988/hello（端口号此处改为了9988）：



Flask模板（续）：过滤器

过滤器的含义及使用

- **定义**：过滤器的本质就是函数，有时候我们不仅仅只是需要输出变量的值，我们还需要修改变量的显示，甚至格式化、运算等等，而在模板中是不能直接调用 Python 中的某些方法，那么这就用到了过滤器。
- **链式调用**：

```
{{ "hello world" | reverse | upper }}
```

- **常见的内建过滤器（字符串操作）**：

```
safe: 禁用转义  <p>{{ '<em>hello</em>' | safe }}</p>

capitalize: 把变量值的首字母转成大写，其余字母转小写    <p>{{ 'hello' | capitalize }}</p>

lower: 把值转成小写 <p>{{ 'HELLO' | lower }}</p>

upper: 把值转成大写 <p>{{ 'hello' | upper }}</p>

title: 把值中的每个单词的首字母都转成大写   <p>{{ 'hello' | title }}</p>

reverse: 字符串反转 <p>{{ 'olleh' | reverse }}</p>

format: 格式化输出  <p>{{ '%s is %d' | format('name',17) }}</p>

striptags: 渲染之前把值中所有的HTML标签都删掉   <p>{{ '<em>hello</em>' | striptags }}</p>

truncate: 字符串截断    <p>{{ 'hello every one' | truncate(9)}}</p>

```

Flask模板（续）：过滤器

- 常见的内建过滤器（列表操作）：

```
first: 取第一个元素 <p>{{ [1,2,3,4,5,6] | first }}</p>

last: 取最后一个元素    <p>{{ [1,2,3,4,5,6] | last }}</p>

length: 获取列表长度    <p>{{ [1,2,3,4,5,6] | length }}</p>

sum: 列表求和    <p>{{ [1,2,3,4,5,6] | sum }}</p>

sort: 列表排序   <p>{{ [6,2,3,1,5,4] | sort }}</p>
```

- 常见的内建过滤器（语句块操作）：

```
{% filter upper %}
    #一大堆文字#
{% endfilter %}
```

- 自定义过滤器：

```
# step1 定义过滤器
def do_listreverse(li):
    temp_li = list(li)
    temp_li.reverse()
    return temp_li

# step2 添加自定义过滤器
app.add_template_filter(do_listreverse, 'listreverse')
```

Flask模板（续）：控制代码块

如何使用控制代码块

- 条件语句：

```
{% if comments | length > 0 %}
    There are {{ comments | length }} comments
{% else %}
    There are no comments
{% endif %}
```

- 循环语句：

```
{% for post in posts %}
    <div>
        <h1>{{ post.title }}</h1>
        <p>{{ post.text | safe }}</p>
    </div>
{% endfor %}
```

- 结合使用：

```
{% for post in posts if post.text %}
    <div>
        <h1>{{ post.title }}</h1>
        <p>{{ post.text | safe }}</p>
    </div>
{% endfor %}
```

Flask模板（续）：代码复用

- **继承：**继承的本质是代码替换，一般用来实现多个页面中重复不变的区域。
- **关键字：**`block extends`

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>top123 {{ title }}</h1><br>

<!-- 预留空间 给继承的html -->
{% block body %}{% endblock %}

<br><h1>bottom</h1>
</body>
</html>
```

```
{% extends 'd1_base.html' %}

{% block body %}
<h2>detail</h2>
{% endblock %}
```

Flask表单

如何创建Flask表单

- 每个web应用都需要使用表单来采集用户数据。现在让我们使用Flask框架创建一个简单的表单来收集用户的基本信息，例如名称、年龄、邮件、兴趣爱好等，我们将这个模板文件命名为bio_form.html。

```
<!DOCTYPE html>
<html>
<head>
    <title></title>
</head>
<body>
    <h1>Bio Data Form</h1>
    <form action="showbio">
        <label>Username</label>
        <input type="name" name="username"><br>
        <label>Email</label>
        <input type="email" name="email"><br>
        <label>Hobbies</label>
        <input type="name" name="hobbies"><br>
        <input type="submit" name="">
    </form>
</body>
</html>
```

Flask表单

- 视图函数bio_data_form同时支持POST和GET请求。GET请求将渲染bio_form.html模板，而POST请求将重定向到showbio：

```
@app.route('/form', methods=['POST', 'GET'])
def bio_data_form():
    if request.method == "POST":
        username = request.form['username']
        age = request.form['age']
        email = request.form['email']
        hobbies = request.form['hobbies']
        return redirect(url_for('showbio', username=username, age=age, email=email, hobbies=hobbies))
    return render_template("bio_form.html")
```

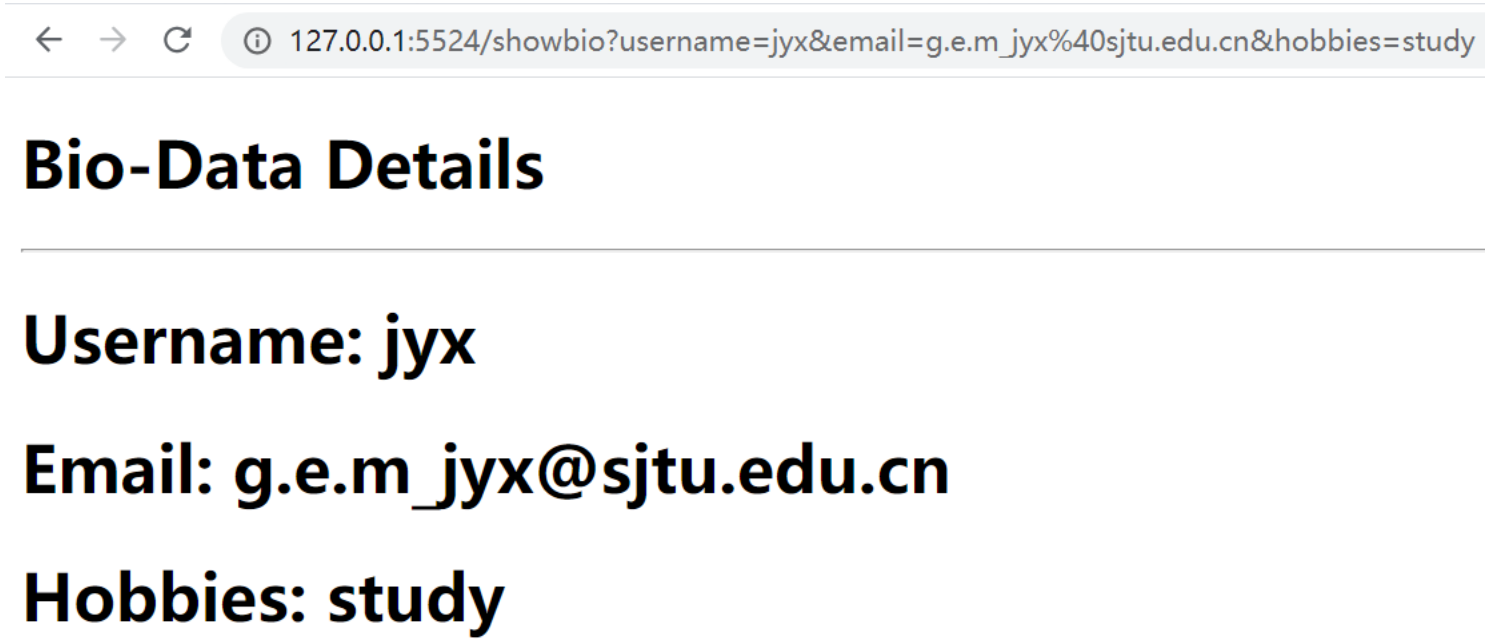
- 下面是showbio的实现：

```
route('/showbio', methods=['GET'])
howbio():
sername = request.args.get('username')
ge = request.args.get('age')
mail = request.args.get('email')
obbies = request.args.get('hobbies')
eturn render_template("show_bio.html", username=username, age=age, email=email, hobbies=hobbies)
```

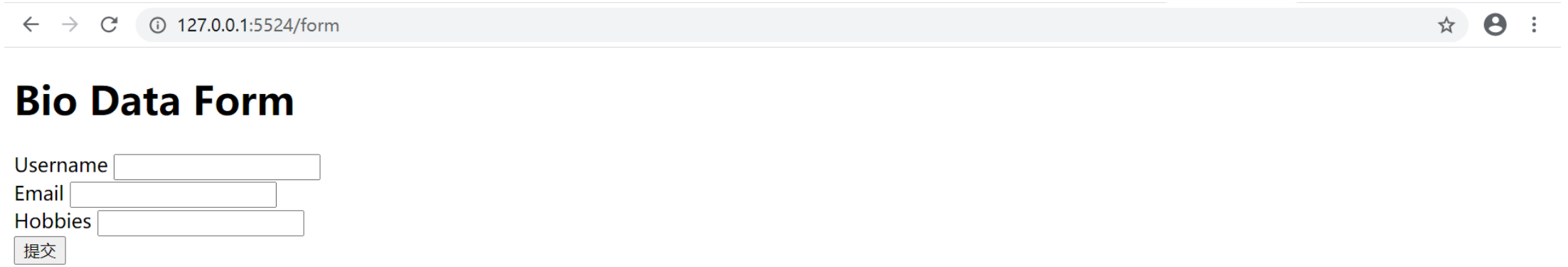
Flask表单

- 其中show_bio.html的内容如下：

```
<!DOCTYPE html>
<html>
<head>
    <title>Bio-Data Details</title>
</head>
<body>
    <h1>Bio-Data Details</h1>
    <hr>
    <h1>Username: {{ username }}</h1>
    <h1>Email: {{ email }}</h1>
    <h1>Hobbies: {{ hobbies }}</h1>
</body>
</html>
```



- 访问http://127.0.0.1:5524/form出现以下表单，提交后出现上方http://127.0.0.1:5524/showbio的页面显示结果：



练习

1.使用Flask，结合前面学习的HTML，Lucene，中文分词等知识点，根据上次实验爬取的网页，建立一个简单的搜索引擎。

搜索界面：

结果界面：

练习

- 结果要求：

  （1）标题

  （2）超链接

  （3）关键词上下文

  （4）网址

- 提示：搜索界面和结果界面两部分，可设置两个路由：

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

  其中，根目录（http://127.0.0.1:8080/）生成搜索框，得到用户输入后重定向到http://127.0.0.1:8080/result，由相应的函数进行结果处理。

- 中文搜索注意分词，注意中文编码格式：gbk或utf8

参考资料

- Flask官方中文文档：https://dormousehole.readthedocs.io/en/latest/

- Flask框架入门：https://blog.csdn.net/sinat_38682860/article/details/82354342
  https://blog.csdn.net/u014793102/article/details/80372815
  https://www.cnblogs.com/sdlypyzq/p/5002084.html

- Flask教程视频：https://www.bilibili.com/video/BV1SJ411P7qb

Part B: 规范网页样式

网页页面布局与样式

- 样式决定了网页中的元素以什么样的形式被显示出来
- 通过html标签属性更改样式的例子：

```
<body bgcolor="yellow"> #背景颜色
<p align="center"> #<p>是分段符号，里面的属性表示内容居中
This is a test paragraph
</p>
<p>
<font face="verdana" color="green">This is some test!</font> #字体
</p>
</body>
```

This is a test paragraph

This is some test!

- 写在html正文段落中，不利于维护和修改
- 若有多个段落使用同样样式，需要一一修改属性

CSS简要介绍

- CSS 指层叠样式表 (Cascading Style Sheets)

- HTML 标签原本被设计为用于定义文档内容。通过使用 <h1>、<p>、<table> 这样的标签，HTML 的初衷是表达“这是标题”、“这是段落”、“这是表格”之类的信息。同时文档布局由浏览器来完成，而不使用任何的格式化标签。

- 由于两种主要的浏览器（Netscape 和 Internet Explorer）不断地将新的 HTML 标签和属性（比如字体标签和颜色属性）添加到 HTML 规范中，创建文档内容清晰地独立于文档表现层的站点变得越来越困难。

- 为了解决这个问题，万维网联盟（W3C），这个非营利的标准化联盟，肩负起了 HTML 标准化的使命，并在 HTML 4.0 之外创造出样式（Style）。把样式添加到 HTML 4.0 中，是为了*解决内容与表现分离的问题。*

DIV + CSS

- DIV+CSS(DIV CSS)为“WEB标准”中常用术语之一。
- DIV是用于搭建html网页结构（框架）标签，像<b>、<h1>、等html标签一样
- CSS用于创建网页表现（样式/美化）
- 通过css来设置div标签样式，这一切常常我们称之为div+css。

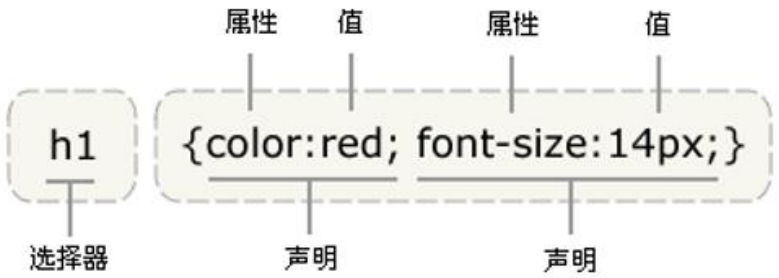
CSS语法

- CSS 规则由两个主要的部分构成：选择器，以及一条或多条声明。

  格式：selector {declaration1; declaration2; ... declarationN }

- 每条声明由一个属性和一个值组成。

  格式：selector {property: value}



- 这行代码的作用是将 h1 元素内的文字颜色定义为红色，同时将字体大小设置为 14 像素。

- 可以对选择器进行分组，这样，被分组的选择器就可以分享相同的声明。

  样式：h1,h2,h3,h4,h5,h6 {color: green; }

CSS选择器

- ID选择器：ID选择器可以为标有特定iD的HTML元素指定特定的样式，ID选择器以“#”来定义。

```
#littleP{
    background-color: blue;
}
<p id="littleP">pppp</p>
```

- 类选择器：类选择可以匹配class属性中值为指定内容的元素，类选择器以一个点号来定义。

```
.info { background:#ff0; }
p.info { background:blue; }
```

- 属性选择器：对带有指定属性的HTML元素设置样式，下面的例子为带有title属性的所有元素设置样式：

```
[title] {color:red;}
a[title="SJTU"]{color:red;}
<h2 title="Hello world">Hello world</h2>
<a title="SJTU" href="http://www.sjtu.edu.cn">SJTU</a>
```

CSS选择器

- 属性选择器的补充：

```
E[att]          匹配所有具有att属性的E元素，不考虑它的值。（注意：E在此处可以省略，比如“[cheacked]”。以下同。）    p[title] {
color:#f00; }
E[att=val]      匹配所有att属性等于“val”的E元素                    div[class=”error”] { color:#f00; }
E[att~=val]     匹配所有att属性具有多个空格分隔的值、其中一个值等于“val”的E元素   td[class~=”name”] { color:#f00; }
E[attr^=val]    匹配属性值以指定值开头的每个元素                    div[class^="test"]{background:#ffff00;}
E[attr$=val]    匹配属性值以指定值结尾的每个元素                    div[class$="test"]{background:#ffff00;}
E[attr*=val]    匹配属性值中包含指定值的每个元素                    div[class*="test"]{background:#ffff00;}
```

- 组合选择器：

```
E,F         多元素选择器，同时匹配所有E元素或F元素，E和F之间用逗号分隔          div,p { color:#f00; }
E F         后代元素选择器，匹配所有属于E元素后代的F元素，E和F之间用空格分隔     li a { font-weight:bold;
E > F       子元素选择器，匹配所有E元素的子元素F                          div > p { color:#f00; }
E + F       毗邻元素选择器，匹配所有紧随E元素之后的同级元素F               div + p { color:#f00; }
```

- 伪类：CSS伪类是用来给选择器添加一些特殊效果：

```
a:link      #（没有接触过的链接）,用于定义了链接的常规状态。
a:hover     #（鼠标放在链接上的状态）,用于产生视觉效果。
a:visited   #（访问过的链接）,用于阅读文章，能清楚的判断已经访问过的链接。
a:active    #（在链接上按下鼠标时的状态）,用于表现鼠标按下时的链接状态。
```

创建CSS

- 当读到一个样式表时，浏览器会根据它来格式化 HTML 文档。插入样式表的方法有三种

（1）**内联样式表**：在标签内使用style属性指定CSS代码，该style只控制内嵌的样式。

```
<a style = "font-size: 40px; color: #c7edcc"> Shonan </a>
```

缺点：当页面（多份html代码）重复使用一种类型的style时，代码大赘余，不易于维护。

（2）**外部样式表**：可以控制多个文件的样式属性，只要文件有连接上等级低于文档内嵌样式表。

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

（3）**文档内嵌样式表**：当单个文档需要特殊的样式时，就应该使用内部样式表。（推荐使用）

```
<style type = "text/css">
    a{
        font-size: 40px;
        color: #c7edcc;
    }
</style>
```

优点：实现了内容与表现相分离，解决了内部样式出现的冗余的问题

CSS属性

可使用CSS规定多种属性

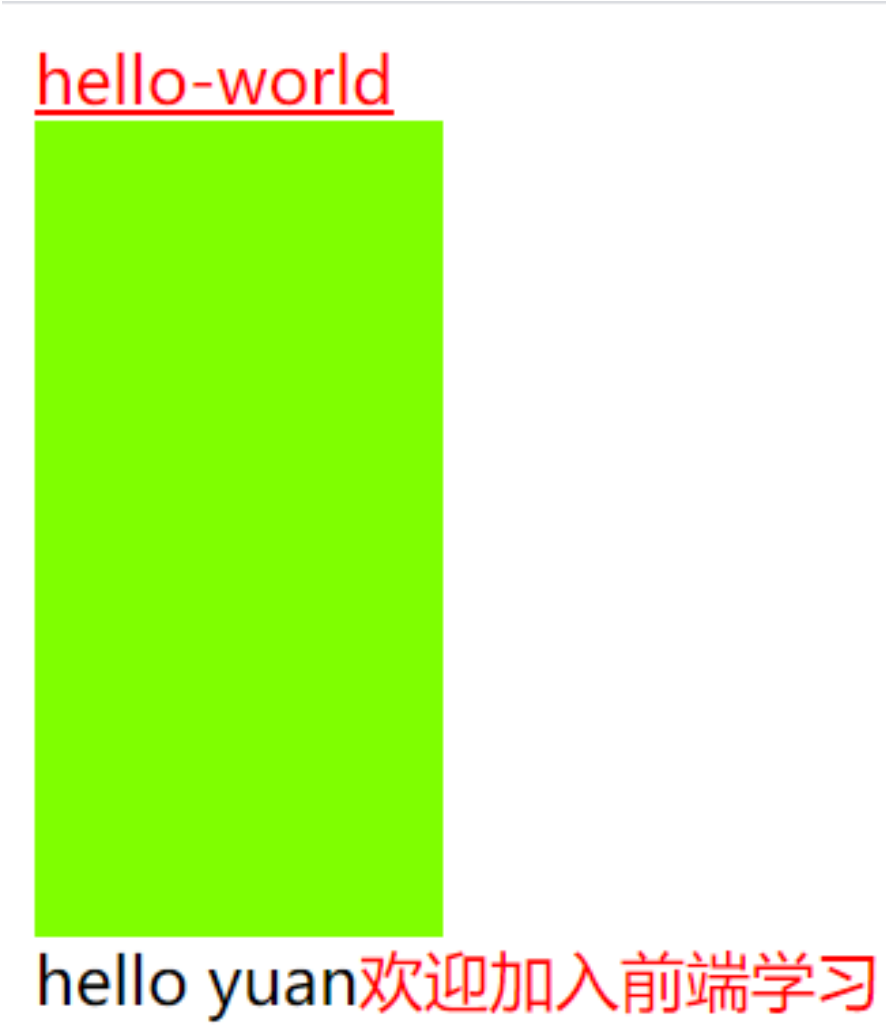
- 高度
- 宽度
- 边框
- 背景
- 排列方式
- 文本
- 字体
- 超链接
- 边距
- 颜色
- 图片
- ……
- 具体实例可参考**CSS属性手册**（http://www.divcss5.com/shouce/）

CSS实例

代码：

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="gbk">
    <title>Title</title>
    <style>
    a:link{color: red;}
    a:visited {color: blue;}
    a:hover {color: green;}
    a:active {color: yellow;}
    .box{width: 100px;}
        .top,.bottom{width: 100px; height: 100px; background-color: chartreuse;}
        .top:hover{background-color: red;}
        .box:hover .bottom{background-color: red;}
        .add:after{content: "欢迎加入前端学习"; color: red;}
    </style>
</head>
<body>
    <a href="css_引入方式.html">hello-world</a>
    <div class="box">
        <div class="top"></div>
        <div class="bottom"></div>
    </div>
    <div class="add">hello yuan</div>
</body>
</html>
```

结果：



练习

1．制作一个图片加文字的搜索引擎。加入图片搜索，使用css制定样式

练习

网页 图片

搜索

"英国"的搜索结果

美国旅游攻略_百度旅游

英国奖牌榜_2012伦敦奥运会_新浪网

英国旅游景点|8月英国旅游攻略 - 艺龙旅游指南

英国_百度百科

英国 - 搜搜百科

英国旅游攻略_百度旅游

英国旅游_英国旅游攻略_英国旅游景点介绍_英国旅游价格_新浪旅游

练习

- 提示：网页和图片搜索可分别建立搜索页和结果页

  比如：@app.route('/')对应网页搜索，@app.route('/im')对应图片搜索；

  @app.route('/result')对应网页结果，@app.route('/imresult')对应图片结果。
- 搜索页中，更换网页/图片搜索，可使用超链接，如在网页搜索页：图片
- 若要在网页中加入中文，Python中加入：

```
reload(sys)
sys.setdefaultencoding('utf8')
```

  HTML模板：另存为UTF8 NO DOM编码
- 使用div+css在单行中输出多个图片：可规定主div和每个子div的width大小。当子div排列数量对应width到达主div的width时，会自动换行
- 把vm_env = initVM()一句写在主函数中，每次搜索时使用vm_env.attachCurrentThread()新建线程，可阻止多次搜索程序崩溃
- 图片自动缩放：注意图片的max-width、max-height属性

练习

- 自主调节css中的各项属性参数与div布局。不存在标准答案，根据个人喜好决定页面表示形式和显示哪些细节。也可以使用已有的前端框架（如 Bootstrap）。

- 有些网站的图片存在防盗链机制，本次试验可先不考虑盗链问题。提示解决方向：添加对应网站的referer。

- 在基础搜索功能上加入新功能，可酌情加分。

参考

- CSS教程：http://www.w3school.com.cn/css/index.asp
- CSS属性手册：http://www.divcss5.com/shouce/
- Codecadamy编程练习：http://www.codecademy.com/