

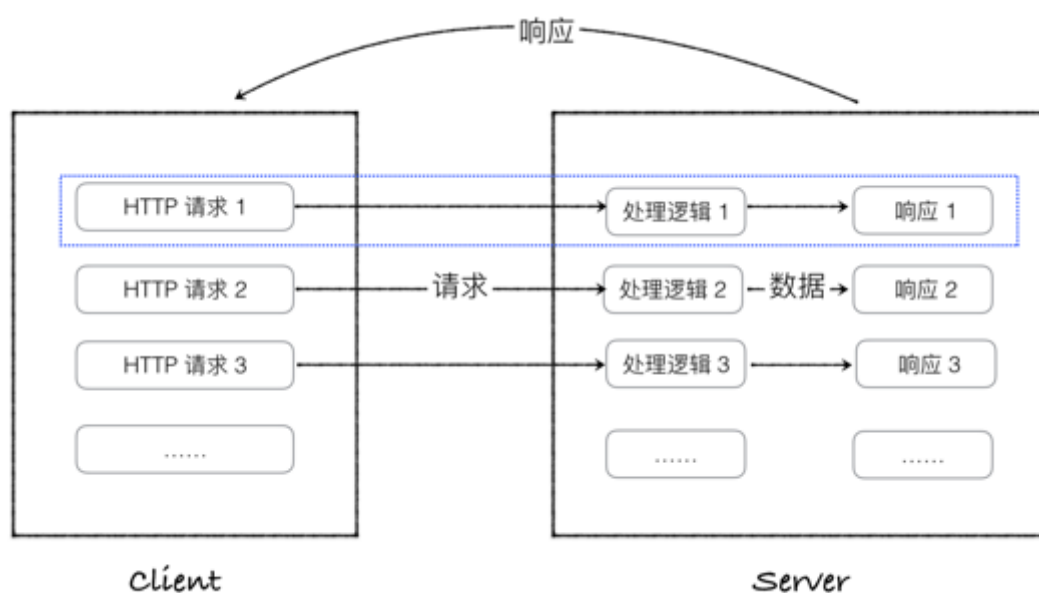
数据可视化

1.Flask入门

1.1 关于Flask

1.1.1 了解框架

Flask作为Web框架，它的作用主要是为了开发Web应用程序。那么我们首先来了解下Web应用程序。Web应用程序 (World Wide Web)诞生最初的目的，是为了利用互联网交流工作文档。



一切从客户端发起请求开始。

- 所有Flask程序都必须创建一个程序实例。
- 当客户端想要获取资源时，一般会通过浏览器发起HTTP请求。
- 此时，Web服务器使用一种名为WEB服务器网关接口的WSGI (Web Server Gateway Interface) 协议，把来自客户端的请求都交给Flask程序实例。
- Flask使用Werkzeug来做路由分发 (URL请求和视图函数之间的对应关系)。根据每个URL请求，找到具体的视图函数。
- 在Flask程序中，路由一般是通程序实例的装饰器实现。通过调用视图函数，获取到数据后，把数据传入HTML模板文件中，模板引擎负责渲染HTTP响应数据，然后由Flask返回响应数据给浏览器，最后浏览器显示返回的结果。

为什么要用Web框架？

web网站发展至今，特别是服务器端，涉及到的知识、内容，非常广泛。这对程序员的要求会越来越高。如果采用成熟，稳健的框架，那么一些基础的工作，比如，网络操作、数据库访问、会话管理等都可以让框架来处理，那么程序开发人员可以把精力放在具体的业务逻辑上面。使用Web框架开发Web应用程序可以降低开发难度，提高开发效率。

总结一句话：避免重复造轮子。

Flask框架的诞生：

Flask诞生于2010年，是Armin ronacher（人名）用Python语言基于Werkzeug工具箱编写的轻量级Web开发框架。它主要面向需求简单的小应用。

Flask本身相当于一个内核，其他几乎所有的功能都要用到扩展（邮件扩展Flask-Mail，用户认证Flask-Login），都需要用第三方的扩展来实现。比如可以用Flask-extension加入ORM、窗体验证工具，文件上传、身份验证等。Flask没有默认使用的数据库，你可以选择MySQL，也可以用NoSQL。其WSGI工具箱采用Werkzeug（路由模块），模板引擎则使用Jinja2。

可以说**Flask框架的核心就是Werkzeug和Jinja2**。

Python最出名的框架要数Django，此外还有Flask、Tornado等框架。虽然Flask不是最出名的框架，但是Flask应该算是最灵活的框架之一，这也是Flask受到广大开发者喜爱的原因。

Flask扩展包：

- Flask-SQLAlchemy：操作数据库；
- Flask-migrate：管理迁移数据库；
- Flask-Mail:邮件；
- Flask-WTF：表单；
- Flask-script：插入脚本；
- Flask-Login：认证用户状态；
- Flask-RESTful：开发REST API的工具；
- Flask-Bootstrap：集成前端Twitter Bootstrap框架；
- Flask-Moment：本地化日期和时间；

1.1.2 安装环境

1.2 视图

Flask程序运行过程：

所有Flask程序必须有一个程序实例。

Flask调用视图函数后，会将视图函数的返回值作为响应的内容，返回给客户端。一般情况下，响应内容主要是字符串和状态码。

当客户端想要获取资源时，一般会通过浏览器发起HTTP请求。此时，Web服务器使用WSGI（Web Server Gateway Interface）协议，把来自客户端的所有请求都交给Flask程序实例。WSGI是为Python语言定义的Web服务器和Web应用程序之间的一种简单而通用的接口，它封装了接受HTTP请求、解析HTTP请求、发送HTTP，响应等等的这些底层的代码和操作，使开发者可以高效的编写Web应用。

程序实例使用Werkzeug来做路由分发（URL请求和视图函数之间的对应关系）。根据每个URL请求，找到具体的视图函数。在Flask程序中，路由的实现一般是通过程序实例的route装饰器实现。route装饰器内部会调用add_url_route()方法实现路由注册。

调用视图函数，获取响应数据后，把数据传入HTML模板文件中，模板引擎负责渲染响应数据，然后由Flask返回响应数据给浏览器，最后浏览器处理返回的结果显示给客户端。

1.2.1 从Helloworld开始

新建文件hello.py:

```
# 导入Flask类
from flask import Flask

# Flask类接收一个参数__name__
app = Flask(__name__)

# 装饰器的作用是将路由映射到视图函数index
@app.route('/')
def index():
    return 'Hello World'

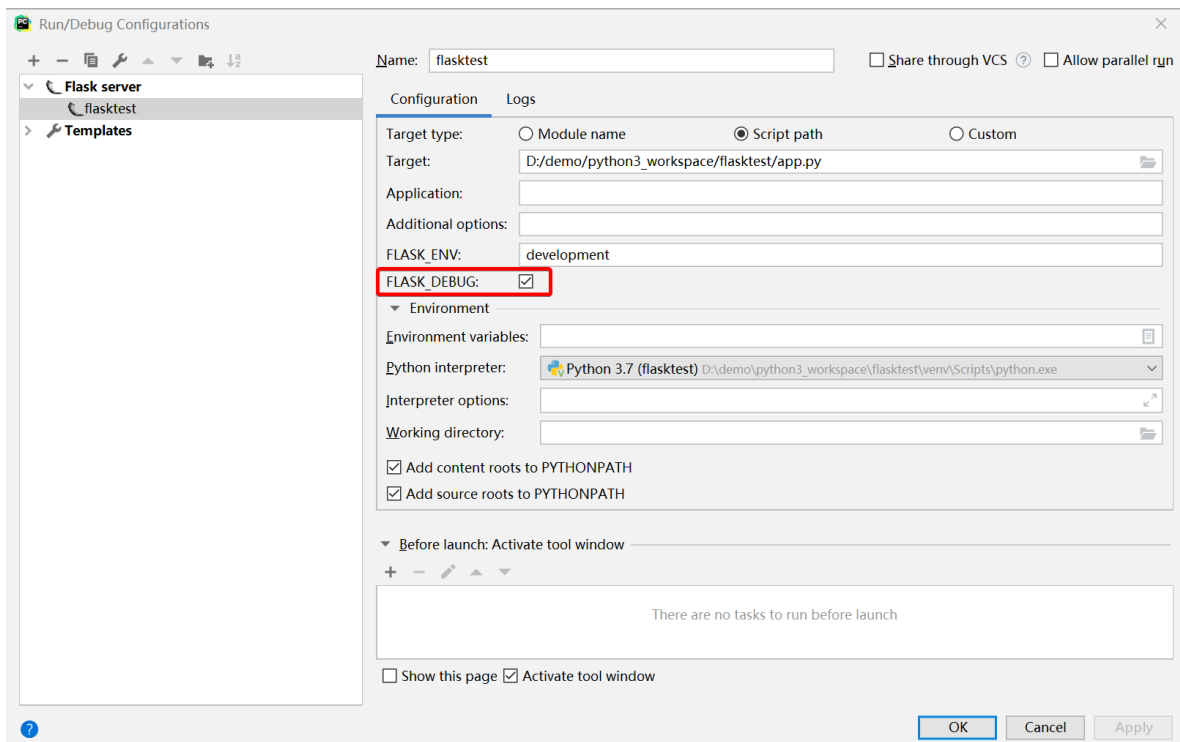
# Flask应用程序实例的run方法启动WEB服务器
if __name__ == '__main__':
    app.run()
```

1.2.2 开启调试模式

Debug模式的开启，可以更好的定位错误，还可以既是刷新修改的页面，不用重启服务器。

```
app.run(debug=True)    #Pycharm中不起作用
```

在运行的环境配置中，勾选Flask Debug



1.2.3 给路由传参示例：

有时我们需要将同一类URL映射到同一个视图函数处理，比如：使用同一个视图函数 来显示不同用户的个人信息。

通过向规则参数添加变量部分，可以动态构建URL。此变量部分标记为。它作为关键字参数传递给与规则相关联的函数。

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

@app.route('/hello/<name>')
def hello_name(name):
    return 'Hello %s!' % name

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

```
# 路由传递的参数默认当做string处理，这里指定int，尖括号中冒号后面的内容是动态的
@app.route('/user/<int:id>')
def hello_itcast(id):
    return 'hello , welcome user %d' % id
```

除了默认字符串变量部分之外，还可以使用以下转换器构建规则：

序号	转换器和描述
1	int 接受整数
2	float 对于浮点值
3	path 接受用作目录分隔符的斜杠

路由的路径不能重复

```
@app.route('/test/')          #/test 和上面/user不同，相同的/user后面可以接不同的类型
def test(name):
    return 'welcome'
```

1.3 模板

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。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使用Jinja2这个模板引擎来渲染模板。Jinja2能识别所有类型的变量，包括{}。Jinja2模板引擎，Flask提供的render_template函数封装了该模板引擎，render_template函数的第一个参数是模板的文件名，后面的参数都是键值对，表示模板中变量对应的真实值。

Jinja2官方文档 (<http://docs.jinkan.org/docs/jinja2/>)

模板的简单使用：

```
@app.route('/')
def hello():
    return render_template('index.html')

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

1.3.1 路由

反向路由：

Flask提供了url_for()辅助函数，可以使用程序URL映射中保存的信息生成URL；url_for()接收视图函数名作为参数，返回对应的URL；

如调用url_for('index',_external=True)返回的是绝对地址，在下面这个示例中是<http://127.0.0.1:5000/index>。

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

@app.route('/user/')
def redirect():
    return url_for('index',_external=True)
```

url_for()**函数对于动态构建特定函数的URL非常有用。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。

以下脚本演示了如何使用url_for()函数：

```
from flask import Flask, redirect, url_for
app = Flask(name)
@app.route('/admin')
def hello_admin():
    return 'Hello Admin'
@app.route('/guest/')
def hello_guest(guest):
    return 'Hello %s as Guest' % guest
@app.route('/user/')
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)
```

上述脚本有一个函数user(name)，它接受来自URL的参数的值。

User()函数检查接收的参数是否与'admin'匹配。如果匹配，则使用url_for()将应用程序重定向到hello_admin()函数，否则重定向到将接收的参数作为guest参数传递给它的hello_guest()函数。

保存上面的代码并从Python shell运行。

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

浏览器中的应用程序响应是：

```
Hello Admin
```

在浏览器中输入以下URL - <http://localhost:5000/user/mvl>

应用程序响应现在更改为：

```
Hello mvl as Guest
```

1.3.2 变量

在模板中`{{ variable }}`结构表示变量，是一种特殊的占位符，告诉模板引擎这个位置的值，从渲染模板时使用的数据中获取；Jinja2除了能识别基本类型的变量，还能识别`{}`；

```
<p>{{mydict['key']}}</p>

<p>{{mylist[1]}}</p>

<p>{{mylist[myvariable]}}</p>
from flask import Flask,render_template
app = Flask(__name__)

@app.route('/')
def index():
    mydict = {'key':'silence is gold'}
    mylist = ['Speech', 'is','silver']
    myintvar = 0

    return render_template('vars.html',
                           mydict=mydict,
                           mylist=mylist,
                           myintvar=myintvar
                           )

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

1.3.3 web表单

web表单是web应用程序的基本功能。

它是HTML页面中负责数据采集的部件。表单有三个部分组成：表单标签、表单域、表单按钮。表单允许用户输入数据，负责HTML页面数据采集，通过表单将用户输入的数据提交给服务器。

Flask 将表单数据发送到模板

我们已经看到，可以在URL规则中指定http方法。触发函数接收的**Form**数据可以以字典对象的形式收集它并将其转发到模板以在相应的网页上呈现它。

在以下示例中，**'/' URL**会呈现具有表单的网页（student.html）。填入的数据会发布到触发 **result()**函数的 **'/result' URL**。

results()函数收集字典对象中的**request.form**中存在的表单数据，并将其发送给**result.html**。

该模板动态呈现**表单**数据的HTML表格。

下面给出的是应用程序的Python代码：

```

from flask import Flask, render_template, request
app = Flask(name)
@app.route('/')
def student():
    return render_template('student.html')
@app.route('/result', methods = ['POST', 'GET'])
def result():
    if request.method == 'POST':
        result = request.form
        return render_template("result.html", result = result)
if name == 'main':
    app.run(debug = True)

```

下面给出的是student.html的HTML脚本。

```

<form action = "http://localhost:5000/result" method = "POST">
  <p>Name <input type = "text" name = "Name" /></p>
  <p>Physics <input type = "text" name = "Physics" /></p>
  <p>Chemistry <input type = "text" name = "chemistry" /></p>
  <p>Maths <input type = "text" name = "Mathematics" /></p>
  <p><input type = "submit" value = "submit" /></p>
</form>

```

下面给出了模板 (result.html) 的代码：

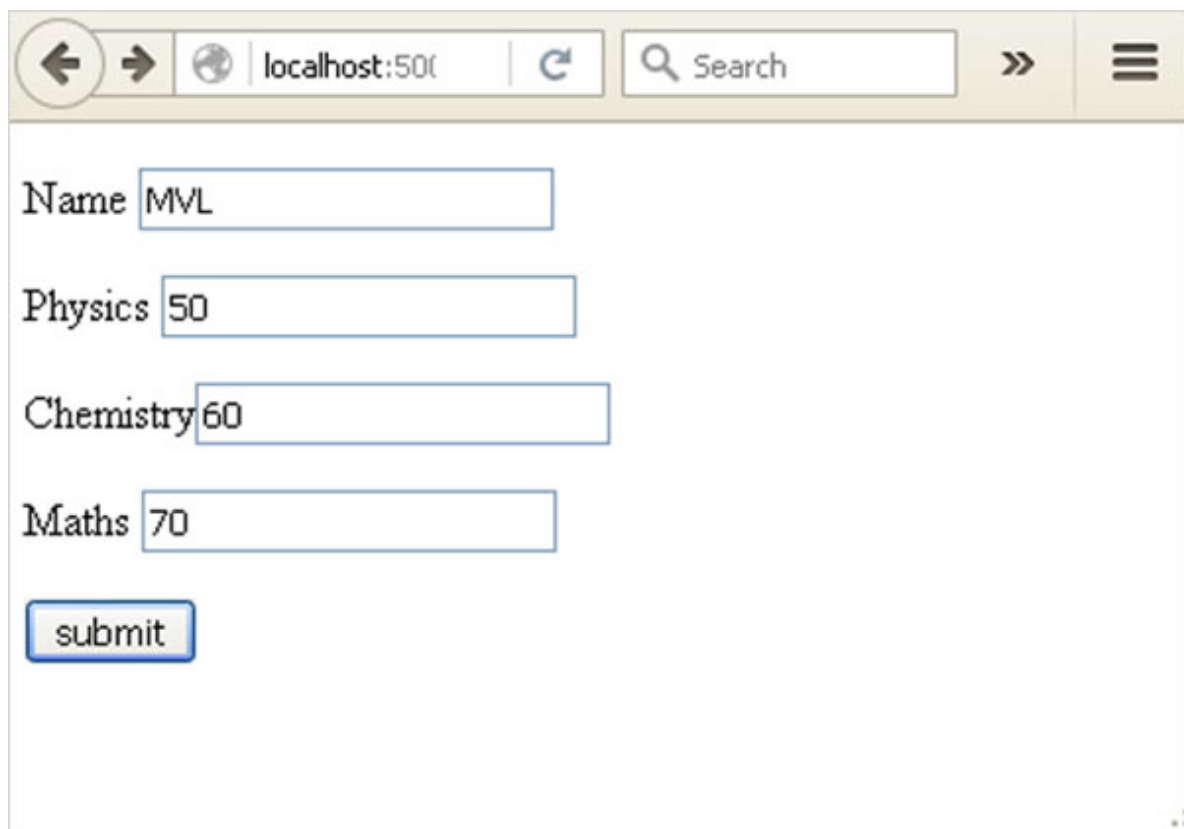
```

<!doctype html>

<table border = 1>
  {% for key, value in result.items() %}
    <tr>
      <th> {{ key }} </th>
      <td> {{ value }} </td>
    </tr>
  {% endfor %}
</table>

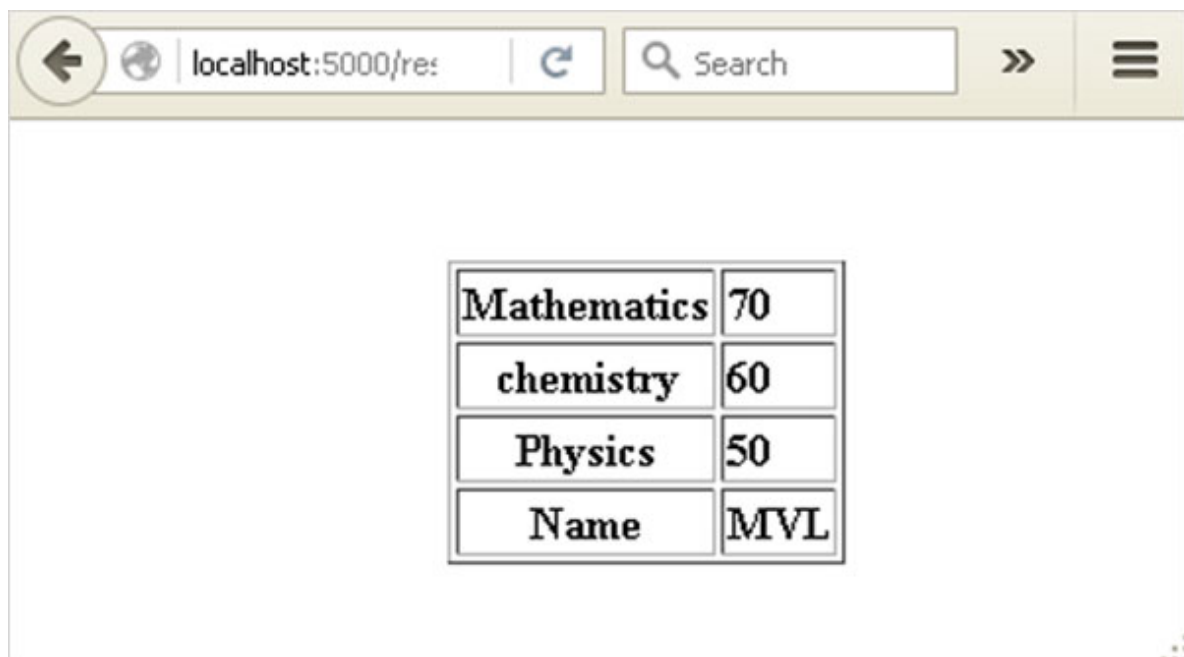
```

运行Python脚本，并在浏览器中输入URL <http://localhost:5000/>。



A screenshot of a web browser window. The address bar shows 'localhost:5000'. The page contains a form with four input fields: 'Name' with the value 'MVL', 'Physics' with the value '50', 'Chemistry' with the value '60', and 'Maths' with the value '70'. Below these fields is a 'submit' button.

当点击提交按钮时，表单数据以HTML表格的形式呈现在result.html上。



A screenshot of a web browser window showing the result of the form submission. The address bar shows 'localhost:5000/result.html'. The page displays a table with the following data:

Mathematics	70
chemistry	60
Physics	50
Name	MVL

1.4 数据库

(略)

1.5 部署

(略)

2.Echarts应用

官方中文网址: <https://www.echartsjs.com/zh/index.html>

2.1 Echarts示例

1.引入Echarts

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <!-- 引入 Echarts 文件 -->
  <script src="echarts.min.js"></script>
</head>
</html>
```

2.设置显示区域

```
<body>
  <!-- 为 ECharts 准备一个具备大小（宽高）的 DOM -->
  <div id="main" style="width: 600px;height:400px;"></div>
</body>
```

3.初始化Echarts

4.指定配置项和数据

5.设置配置项显示图表

下面是完整的示例:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Echarts</title>
  <!--1. 引入 echarts.js -->
  <script src="echarts.min.js"></script>
</head>
<body>
  <!--2. 为ECharts准备一个具备大小（宽高）的Dom -->
  <div id="main" style="width: 600px;height:400px;"></div>
  <script type="text/javascript">
    //3. 基于准备好的dom，初始化echarts实例
    var myChart = echarts.init(document.getElementById('main'));
```

```
//4. 指定图表的配置项和数据
var option = {
    title: {
        text: 'Echarts 入门示例'
    },
    tooltip: {},
    legend: {
        data: ['销量']
    },
    xAxis: {
        data: ["衬衫", "羊毛衫", "雪纺衫", "裤子", "高跟鞋", "袜子"]
    },
    yAxis: {},
    series: [{
        name: '销量',
        type: 'bar',
        data: [5, 20, 36, 10, 10, 20]
    }]
};

//5. 使用刚指定的配置项和数据显示图表。
myChart.setOption(option);
</script>
</body>
</html>
```

2.2 Echarts配置

官方实例: <https://www.echartsjs.com/examples/zh/index.html>

官方教程: <https://www.echartsjs.com/zh/tutorial.html#5%20%E5%88%86%E9%92%9F%E4%B8%8A%E6%89%8B%20Echarts>](<https://www.echartsjs.com/zh/tutorial.html>)

3.WordCloud应用

3.1 WordCloud示例

3.1.1 WordCloud库的安装

第一步: <http://www.lfd.uci.edu/~gohlke/pythonlibs/#wordcloud> 页面下载所需的wordcloud模块的whl文件, 这里选择wordcloud-1.6.0-cp37-cp37m-win32.whl 版本。

注意: 下载的版本!!

cp后面是你的python版本, 例如: 3.7.1就是cp37

查看python的版本号, 可以在命令提示符界面下输入: python

```
C:\Users\ThinkPad>python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

第二步：下载后进入存储该文件的路径，cmd打开dos界面，cd进你们刚刚下载到的文件路径

第三步：执行“pip install wordcloud-1.6.0-cp37-cp37m-win32.whl”，安装成功。

3.1.2 WordCount的例子

第一步：引入库文件

```
import jieba                #分词
from matplotlib import pyplot as plt    #绘图
from wordcloud import WordCloud        #词云
from PIL import Image            #图片处理
import numpy as np              #矩阵运算
import sqlite3                  #sqlite3数据库
```

第二步：读取文件，或者读取数据库

```
path = r'.'                  #设定根路径
#font = r'C:\Windows\Fonts\msyh.ttc'    #设定字体

#text = (open(path + r'\岗位需求.txt', 'r', encoding='utf-8')).read()    #读取文件

con = sqlite3.connect('D:\\demo\\python3_workspace\\baidu_demo\\movie.db')    #连接数据库
cur = con.cursor()
sql = 'select introduction from movie250'
data = cur.execute(sql)
text = ""
for item in data:            #将查询出的结果连接成一个字符串
    text = text + item[0]

print(text)
cur.close()
con.close()
```

第三步：应用特定图片合成词云

```
cut = jieba.cut(text)        # 分词
string = ' '.join(cut)
print(len(string))           #打印分词数量
img = Image.open(path + r'\\2.jpg')    # 打开图片
img_array = np.array(img)        #将图片转换为数组
wc = WordCloud(               #词云参数设置
    background_color='white',    # 设置背景颜色
    mask=img_array,              # 设置背景图片
    font_path="msyh.ttc"         #'C:\Windows\Fonts\STZHONGS.TTF',    # 若是有中文的
                                #话，这句代码必须添加，不然会出现方框，不出现汉字
).generate_from_text(string)
#wc.generate(string)           # 绘制图片
#wc.to_file(path + r'\\new.png')    # 保存默认图片
```

```

fig = plt.figure(1)          #新建一个名叫 Figure1的画图窗口
plt.imshow(wc)               #显示图片，同时也显示其格式
plt.axis('off')              # 是否显示x轴、y轴下标

#plt.show()                 #显示生成合成图片

plt.savefig(path+'\\new.png',dpi=500)  #保存合成图片，dpi是设定分辨率，默认为400

```

3.2 WordCloud配置

WordCloud个参数的含义：

font_path : string #字体路径，需要展现什么字体就把该字体路径+后缀名写上，如：font_path = '黑体.ttf'

width : int (default=400) #输出的画布宽度，默认为400像素

height : int (default=200) #输出的画布高度，默认为200像素

prefer_horizontal : float (default=0.90) #词语水平方向排版出现的频率，默认 0.9 （所以词语垂直方向排版出现频率为 0.1 ）

mask : nd-array or None (default=None) #如果参数为空，则使用二维遮罩绘制词云。如果 mask 非空，设置的宽高值将被忽略，遮罩形状被 mask 取代。除全白（#FFFFFF）的部分将不会绘制，其余部分会用于绘制词云。如：bg_pic = imread('读取一张图片.png')，背景图片的画布一定要设置为白色（#FFFFFF），然后显示的形状为不是白色的其他颜色。可以用ps工具将自己要显示的形状复制到一个纯白色的画布上再保存，就ok了。

scale : float (default=1) #按照比例进行放大画布，如设置为1.5，则长和宽都是原来画布的1.5倍

min_font_size : int (default=4) #显示的最小的字体大小

font_step : int (default=1) #字体步长，如果步长大于1，会加快运算但是可能导致结果出现较大的误差

max_words : number (default=200) #要显示的词的最大个数

stopwords : set of strings or None #设置需要屏蔽的词，如果为空，则使用内置的STOPWORDS

background_color : color value (default="black") #背景颜色，如background_color='white',背景颜色为白色

max_font_size : int or None (default=None) #显示的最大的字体大小

mode : string (default="RGB") #当参数为"RGBA"并且background_color不为空时，背景为透明

relative_scaling : float (default=.5) #词频和字体大小的关联性

color_func : callable, default=None #生成新颜色的函数，如果为空，则使用 self.color_func

regex : string or None (optional) #使用正则表达式分隔输入的文本

collocations : bool, default=True #是否包括两个词的搭配

colormap : string or matplotlib colormap, default="viridis" #给每个单词随机分配颜色，若指定 color_func，则忽略该方法

random_state : int or None #为每个单词返回一个PIL颜色

```
fit_words(frequencies) #根据词频生成词云
generate(text) #根据文本生成词云
generate_from_frequencies(frequencies[, ...]) #根据词频生成词云
generate_from_text(text) #根据文本生成词云
process_text(text) #将长文本分词并去除屏蔽词（此处指英语，中文分词还是需要自己用别的库
先行实现，使用上面的 fit_words(frequencies) )
recolor([random_state, color_func, colormap]) #对现有输出重新着色。重新上色会比重新生成
整个词云快很多
to_array() #转化为 numpy array
to_file(filename) #输出到文件
```

4.项目说明

4.1 需求说明

项目名称：51job招聘网数据爬取分析平台

项目背景：

随着互联网在中国的普及，网络招聘景气不断攀升。它允许以更加灵活的交互方式，提供更加丰富的信息资源。为求职者提供更加多样、精确、新鲜的招聘信息。

项目意义：

面对眼花缭乱的招聘信息，帮助求职者能够通过招聘信息的数据可视化，分析招聘薪资等信息的分布，明确自己的方向，找到适合的工作。

项目目标：

- 1、实现通过搜索对相关招聘的主要信息进行展示、下载（选做）。
- 2、对搜索出的信息进行地域、薪资、工作经验、学历、职责与要求等个方面的数据可视化分析。

4.2 技术要求

技术环境：

Python3、urllib库、flask框架、Echarts.js、wordcloud库、sqlite3数据库

分工要求：

每组4-5人，每人有一个菜单项，完成相应的数据可视化的功能展示，功能不能相同。

考核内容：

+ 基本功能：

- 1.使用爬虫爬取51job数据至少1000条，并保存到数据库中
- 2.应用flask框架完成网站搭建并能够本地访问

3.能够引导用户搜索关键词，完成相应内容可视化的展现

+ 鼓励创新：

1.数据呈现的多样化：多种图表形式

2.数据维度的设计：能够从不同维度的数据分析，为用户提供更多价值

3.界面表现的美化

提交材料：

- 项目源码
- 实验报告