一、requests

通过前面 urllib 库的学习，你会发现，虽然 Python 的标准库中 urllib 模块已经包含了平常我们使用的大多数功能，但是 urllib 用起来非常的不方便，而 Requests 自称 “HTTP for Humans”，说明它会比 urllib 更加方便更加简洁，可以节约我们大量的工作。（用了 requests 之后，你基本都不愿意用 urllib 了）一句话，requests 是 python 实现的最简单易用的 HTTP 库，建议爬虫使用 requests 库。

Requests 继承了 urllib 的所有特性。Requests 支持 HTTP 连接保持和连接池，支持使用 cookie 保持会话，支持文件上传，支持自动确定响应内容的编码，支持国际化的 URL 和 POST 数据自动编码。

requests 的底层实现其实就是 urllib3 Requests 的文档非常完备，中文文档也相当不错。Requests 能完全满足当前网络的需求，支持 Python2.6—3.5，而且能在 PyPy 下完美运行。

开源地址：https://github.com/kennethreitz/requests

中文文档 API： http://docs.python-requests.org/zh_CN/latest/index.html

1.equests 的基础使用

1.整体演示

```
import requests

response  = requests.get("https://www.baidu.com")
print(type(response))
print(response.status_code)
print(type(response.text))
print(response.text)
print(response.cookies)
print(response.content)

# 很多情况下的网站如果直接 response.text 会出现乱码的问题，所以这个使用 response.content，这样返回的数据格式其实是二进制格式，然后通过 decode()转换为 utf-8，这样就解决了通过 response.text 直接返回显示乱码的问题.
print(response.content.decode("utf-8"))
```

请求发出后，Requests 会基于 HTTP 头部对响应的编码作出有根据的推测。当你访问 response.text 之时，Requests 会使用其推测的文本编码。你可以找出 Requests 使用了什么编码，并且能够使用 response.encoding 属性来改变它.如：

```
import requests
```

```
response =requests.get("http://www.baidu.com")

response.encoding="utf-8"

print(response.text)
```

不管是通过 response.content.decode("utf-8)的方式还是通过 response.encoding="utf-8"的方式都可以避免乱码的问题发生

2.基本请求

1) get 请求

最基本的 GET 请求可以直接用 get 方法

```
import requests

response = requests.get("http://www.baidu.com/")

# 也可以这么写
# response = requests.request("get", "http://www.baidu.com/")

print(response.text)
```

添加 headers 和 查询参数

如果想添加 headers，可以传入 headers 参数来增加请求头中的 headers 信息。如果要将参数放在 url 中传递，可以利用 params 参数。

```
import requests

headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36"}


data = {
    "name":"python"
}
response = requests.get("http://httpbin.org/get",params=data,headers = headers)

# 查看响应内容，response.text 返回的是Unicode 格式的数据
print(response.text)

# 查看响应内容，response.content 返回的字节流数据
print(respones.content)
```

```
# 如果是 json 文件可以直接显示
print(response.json())

# 查看完整 url 地址
print(response.url)

# 查看响应头部字符编码
print(response.encoding)

# 查看响应码
print(response.status_code)
```

```
{"args":{"name":"python"},"headers":{"Accept":"*/*","Accept-Encoding":"gzip, def
late","Connection":"close","Host":"httpbin.org","User-Agent":"Mozilla/5.0 (Windo
ws NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.9
9 Safari/537.36"},"origin":"183.6.137.130","url":"http://httpbin.org/get?name=py
thon"}

b'{"args":{"name":"python"},"headers":{"Accept":"*/*","Accept-Encoding":"gzip, d
eflate","Connection":"close","Host":"httpbin.org","User-Agent":"Mozilla/5.0 (Win
dows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.
99 Safari/537.36"},"origin":"183.6.137.130","url":"http://httpbin.org/get?name=p
ython"}\n'

{'origin': '113.108.202.180', 'headers': {'Host': 'httpbin.org', 'Connection': 'c
lose', 'Accept': '*/*', 'Accept-Encoding': 'gzip, deflate', 'User-Agent': 'Mozill
a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome
/54.0.2840.99 Safari/537.36'}, 'url': 'http://httpbin.org/get?name=python', 'arg
s': {'name': 'python'}}


http://httpbin.org/get?name=python

None

200
```

通过运行结果可以判断，请求的链接被自动构造成 http://httpbin.org/get?name=python

另外，网页的返回类型实际上是 str 类型，但是它是 json 格式的，所以想直接解析返回结果，得到一个字典格式的话可以直接调用 json()方法

2) post 请求

通过在发送 post 请求时添加一个 data 参数，这个 data 参数可以通过字典构造成，这样对于发送 post 请求就非常方便

```
import requests

data = {
    "name":"python",
}
response = requests.post("http://httpbin.org/post",data=data)

print(response.text)
```

{"args":{},"data":"","files":{},"form":{"name":"python"},"headers":{"Accept":"*/*","Accept-Encoding":"gzip, deflate","Connection":"close","Content-Length":"11","Content-Type":"application/x-www-form-urlencoded","Host":"httpbin.org","User-Agent":"python-requests/2.9.1"},"json":null,"origin":"113.108.202.180","url":"http://httpbin.org/post"}

3.超时设置

通过 timeout 参数可以设置超时的时间

```
import requests

res = requests.get('http://www.hqjy.com/',timeout = 1)
print(res.status_code)
```

将超时时间设置 1 秒，如果 1 秒之内没有响应就抛出异常

4.异常处理

关于 reqeusts 的异常在这里可以看到详细内容：
http://www.python-requests.org/en/master/api/#exceptions 所有的异常都是在 requests.excepitons 中

Exceptions

exception `requests.`**`RequestException`**(**args*, ***kwargs*) [source]
There was an ambiguous exception that occurred while handling your request.

exception `requests.`**`ConnectionError`**(**args*, ***kwargs*) [source]
A Connection error occurred.

exception `requests.`**`HTTPError`**(**args*, ***kwargs*) [source]
An HTTP error occurred.

exception `requests.`**`URLRequired`**(**args*, ***kwargs*) [source]
A valid URL is required to make a request.

exception `requests.`**`TooManyRedirects`**(**args*, ***kwargs*) [source]
Too many redirects.

exception `requests.`**`ConnectTimeout`**(**args*, ***kwargs*) [source]
The request timed out while trying to connect to the remote server.

Requests that produced this error are safe to retry.

exception `requests.`**`ReadTimeout`**(**args*, ***kwargs*) [source]
The server did not send any data in the allotted amount of time.

exception `requests.`**`Timeout`**(**args*, ***kwargs*) [source]
The request timed out.

Catching this error will catch both **`ConnectTimeout`** and **`ReadTimeout`** errors.

从源码我们可以看出 RequestException 继承 IOError, HTTPError，ConnectionError,Timeout 继承 RequestionException ProxyError，SSLError 继承 ConnectionError ReadTimeout 继承 Timeout 异常 这里列举了一些常用的异常继承关系，详细的可以看：http://cn.python-requests.org/zh*CN/latest/*modules/requests/exceptions.html#RequestException

```
import requests
from requests.exceptions import ReadTimeout,ConnectionError,RequestException


try:
    response = requests.get("http://httpbin.org/get",timout=0.1)
    print(response.status_code)
except ReadTimeout:
    print("timeout")
except ConnectionError:
    print("connection Error")
except RequestException:
    print("error")
```

其实最后测试可以发现，首先被捕捉的异常是 timeout,当把网络断掉的 haul 就会捕捉到 ConnectionError，如果前面异常都没有捕捉到，最后也可以通过 RequestExctption 捕捉到

2.requests 的高级使用

1.代理（proxies 参数）

如果需要使用代理，你可以通过为任意请求方法提供 proxies 参数来配置单个请求：

```
import requests

proxies= {
    "http":"http://127.0.0.1:9999"
}
response  = requests.get("https://www.baidu.com",proxies=proxies)

print(response.text)
```

如果代理需要设置账户名和密码,只需要将字典更改为如下：

```
proxies = {
"http":"http://user:password@127.0.0.1:9999"
}
```

2.认证设置

如果碰到需要认证的网站可以通过 requests.auth 模块实现

```
import requests
from requests.auth import HTTPBasicAuth

response = requests.get("http://120.27.34.24:9001/",auth=HTTPBasicAuth("user","123"))
print(response.status_code)
```

另一种方式：

```
import requests

response = requests.get("http://120.27.34.24:9001/",auth=("user","123"))
print(response.status_code)
```

3.Cookies 和 Session

1) 获取 cookie

```
import requests

response = requests.get("http://www.baidu.com")
```

```
print(response.cookies)

for key,value in response.cookies.items():
    print(key+"="+value)
```

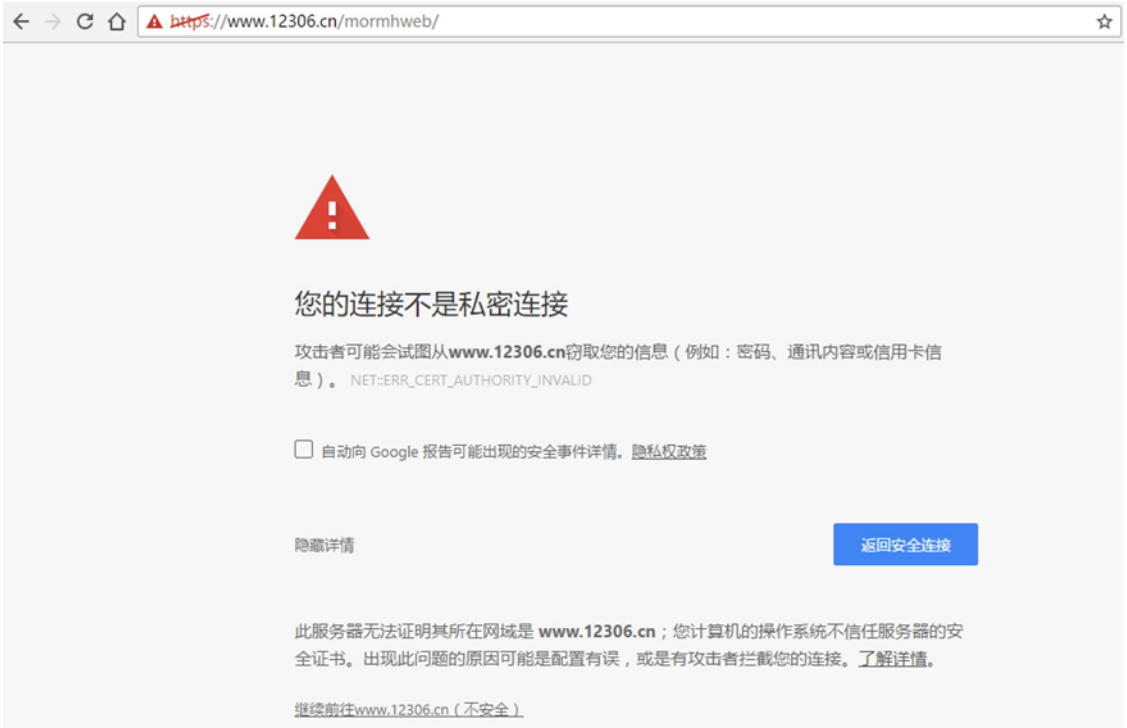
2) 会话维持 session

cookie 的一个作用就是可以用于模拟登陆，做会话维持

```
import requests

# 创建 session 对象，可以保存 Cookie 值
s = requests.Session()

# 发送请求，并获取 Cookie 值保存在 ssion 里
s.get("http://httpbin.org/cookies/set/name/python")

response = s.get("http://httpbin.org/cookies")

print(response.text)
```

4. SSL 证书验证

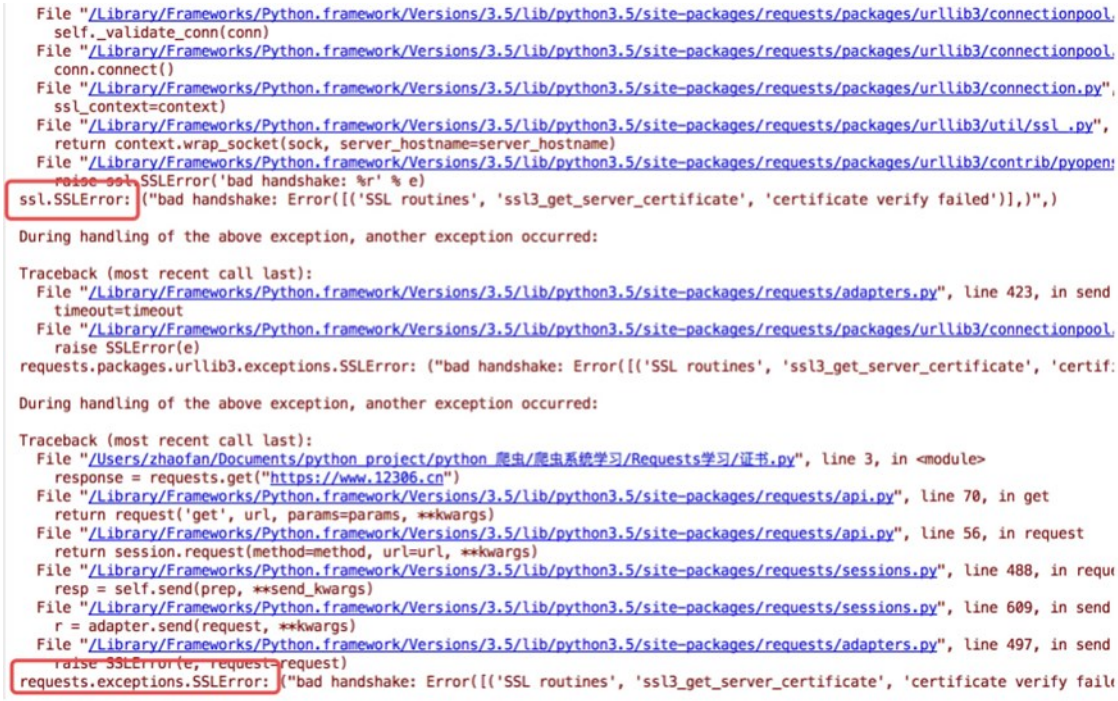
现在的很多网站都是 https 的方式访问，所以这个时候就涉及到证书的问题




```
import requests
```

```
response = requests.get("https://www.12306.cn/mormhweb/")
print(response.status_code)
```

如果 SSL 证书验证不通过，或者不信任服务器的安全证书，则会报出 SSLError，默认的 12306 网站的证书是不合法的，这样就会提示如下错误



为了避免这种情况的发生可以通过 verify=False 即可

```
response = requests.get("https:/www.12306.cn",verify=False)
```

二、数据存储

之前已经知道怎么爬取数据了，但是，现在问题来了，我们爬下来的数据怎么办呢，当然是存起来了，存到哪里呢？

保存的形式有很多种，最简单的是直接保存为文本文件，如 TXT，JSON，CSV。另外高级一点的我们可以存在数据库里面，然后存在数据库里又分为保存在关系型数据库和保存在非关系型数据库

1.TXT 文件存储

将数据保存到 TXT 文本的操作非常简单，而且 TXT 文本几乎兼容任何平台，但是这有个缺点，那就是不利于检索。所以如果对检索和数据结构要求不高，追求方便第一的话，可以采用 TXT 文本存储。

```
import requests
from pyquery import PyQuery as pq

url = 'https://www.zhihu.com/explore'
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3) AppleWebKit/53
7.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36'
}
html = requests.get(url, headers=headers).text
doc = pq(html)
items = doc('.explore-tab .feed-item').items()
for item in items:
    question = item.find('h2').text()
    author = item.find('.author-link-line').text()
    answer = pq(item.find('.content').html()).text()
    file = open('explore.txt', 'a', encoding='utf-8')
    file.write('\n'.join([question, author, answer]))
    file.write('\n' + '=' * 50 + '\n')
    file.close()
```

首先用 Requests 提取了知乎发现页面，然后将热门问题的问题、回答者、答案全文提取出来，然后利用了 Python 提供的 open() 方法打开一个文本文件，获取一个文件操作对象，这里赋值为 file，然后利用 file 对象的 write() 方法将提取的内容写入文件，最后记得调用一下 close() 方法将其关闭，这样抓取的内容即可成功写入到文本中。

1.打开方式

刚才的实例中，open()方法的第二个参数设置成了 a，这样在每次写入文本时不会清空源文件，是在文件末尾写入新的内容，这是一种文件打开方式。关于文件的打开方式，其实还有其他几种

- r：以只读方式打开文件，文件的指针将会放在文件的开头。这是默认模式。
- rb: 以二进制只读方式打开一个文件。文件指针将会放在文件的开头。
- r+：以读写方式打开一个文件，文件指针将会放在文件的开头。
- rbt：以二进制读写方式打开一个文件。文件指针将会放在文件的开头。
- w ：以写入方式打开一个文件，如果该文件已经存在，则将其覆盖，如果该文件不存在，则创建新的文件
- wb：以二进制写入方式打开一个文件，如果该文件已经存在，则将其覆盖，如果该文件不存在，则创建新的文件
- w+ ：以读写方式打开一个文件，如果该文件已经存在，则将其覆盖，如果该文件不存在，则创建新的文件

- wb+：以二进制读写方式打开一个文件，如果该文件已经存在，则将其覆盖，如果该文件不存在，则创建新的文件
- wb+：以二进制读写方式打开一个文件，如果该文件已经存在，则将其覆盖，如果该文件不存在，则创建新的文件
- a：以追加方式打开一个文件，如果该文件已经存在，文件指针将会放在文件末尾，也就是新的内容将会被写入到已有的文件之后，如果文件不存在，则创建新的文件来写入
- ab：以二进制追加方式打开一个文件，如果该文件已经存在，文件指针将会放在文件末尾，也就是新的内容将会被写入到已有的文件之后，如果文件不存在，则创建新的文件来写入
- a+ ：以读写方式打开一个文件，如果该文件已经存在，文件指针将会放在文件末尾，文件打开时会是追加模式，如果文件不存在，则创建新的文件来读写
- ab+ ：以二进制追加方式打开一个文件，如果该文件已经存在，文件指针将会放在文件末尾,如果文件不存在,则创建新的文件来读写

2.简化写法

另外，文件写人还有一种简写方法，那就是使用 with as 语法，在 with 控制模块结束时，文件会自动关闭，所以不需要再调用 close()方法，这种保存方式可以简写如下:

```
with open('explore.text', 'a', encoding='utf-8') as f:
    f.write('\n'.join([question, author, anster]))
    f.write('\n' + '=' * 50 + '\n')
```

上面便是利用 Python 将结果保存为 TXT 文件的方法，这种方法简单易用，操作高效、是一种最基本的保存数据的方法。

2.Json 文件存储

Json，全称为 JavaScript Object Notation, 也就是 JavaScript 对象标记，通过对象和数组的组合来表示数据，构造简洁但是结构化程度非常高，它是一种轻量级的数据交换格式，本节我们来了解一下利用 Python 保存数据到 Json 文件的方法。

1.对象和数组

在 JavaScript 语言中，一切都是对象。因此，任何支持的类型都可以通过 Json 来表示，例如字符串、数字、对象、数组等。但是对象和数组是比较特殊且常用的两种类型。

- 对象，对象在 JavaScript 中是使用花括号 {} 包裹起来的内容，数据结构为 {key1：value1, key2：value2, ...} 的键值对结构。在面向对象的语言中，key 为对象的属性，value 为对应的值。键名可以使用整数和字符串来表示。值的类型可以是任意类型。

- 数组，数组在 JavaScript 中是方括号 [] 包裹起来的内容，数据结构为 ["java", "javascript", "vb", ...] 的索引结构。在 JavaScript 中，数组是一种比较特殊的数据类型，它也可以像对象那样使用键值对，但还是索引使用得多。同样，值的类型可以是任意类型。 所以一个 Json 对象可以写为如下形式：

```
[{
    "name": "Bob",
    "gender": "male",
    "birthday": "1992-10-18"
}, {
     "name": "Selina",
    "gender": "female",
    "birthday": "1995-10-18"
}]
```

由中括号包围的就相当于列表类型，列表的每个元素可以是任意类型，在示例中它是字典类型，由大括号包围。

Json 可以由以上两种形式自由组合而成，可以无限次嵌套，结构清晰，是数据交换的极佳方式。

1)读取 Json

Python 为我们提供了简单易用的 json 库来供我们实现 Json 文件的读写操作，我们可以调用 json 库的 loads() 方法将 Json 文本字符串转为 Python 对象，可以通过 dumps()方法将 Python 对象转为文本字符串。

有一段 Json 形式的字符串，它是 str 类型，我们用 Python 将可其转换为可操作的数据结构，如列表或字典。

```
import json

str = '''
[{
    "name": "Bob",
    "gender": "male",
    "birthday": "1992-10-18"
}, {
    "name": "Selina",
    "gender": "female",
    "birthday": "1995-10-18"
}]
'''
print(type(str))
data = json.loads(str)
print(data)
print(type(data))
```

运行结果：

```
<class 'str'>
[{'name': 'Bob', 'gender': 'male', 'birthday': '1992-10-18'}, {'name': 'Selina',
'gender': 'female', 'birthday': '1995-10-18'}]
<class 'list'>
```

在这里我们使用了 loads() 方法将字符串转为 Python 对象，由于最外层是中括号，所以最终的类型是列表类型。

这样一来我们就可以用索引来取到对应的内容了，例如我们想取第一个元素里的 name 属性，就可以使用如下方式获取：

```
data[0]['name']
data[0].get('name')
```

得到的结果都是：

```
Bob
```

通过中括号加 0 索引我们可以拿到第一个字典元素，然后再调用其键名即可得到相应的键值。在获取键值的时候有两种方式，一种是中括号加键名，另一种是 get() 方法传入键名。推荐使用 get() 方法来获取内容，这样如果键名不存在的话不会报错，会返回 None。另外 get() 方法还可以传入第二个参数即默认值，我们用一个示例感受一下：

```
data[0].get('age')
data[0].get('age', 25)
```

运行结果：

```
None
25
```

在这里我们尝试获取年龄 age，其实在原字典中是不存在该键名的，如果不存在，默认会返回 None，如果传入第二个参数即默认值，那么在不存在的情况下则返回该默认值。

值得注意的是 Json 的数据需要用双引号来包围，不能使用单引号。例如若使用如下形式表示则会出现错误：

```
import json

str = '''
[{
    'name': 'Bob',
    'gender': 'male',
    'birthday': '1992-10-18'
}]
'''
data = json.loads(str)
```

运行结果：

```
json.decoder.JSONDecodeError: Expecting property name enclosed in double quotes:
line 3 column 5 (char 8)
```

在这里会出现 Json 解析错误的提示，是因为在这里数据用了单括号来包围，请千万注意 Json 字符串的表示需要用双引号，否则 loads() 方法会解析失败。

如果我们是从 Json 文本中读取内容，例如在这里有一个 data.json 文本文件，其内容是刚才我们所定义的 Json 字符串。

我们可以先将文本文件内容读出，然后再利用 loads() 方法转化。

```
import json

with open('data.json', 'r') as file:
    str = file.read()
    data = json.loads(str)
    print(data)
```

运行结果：

```
[{'name': 'Bob', 'gender': 'male', 'birthday': '1992-10-18'}, {'name': 'Selina',
'gender': 'female', 'birthday': '1995-10-18'}]
```

以上是读取 Json 文件的方法。

2) 输出 Json

另外我们还可以调用 dumps() 方法来将 Python 对象转化为字符串。例如我们将刚上例中的列表重新写入到文本。

```
import json

data = [{
    'name': 'Bob',
    'gender': 'male',
    'birthday': '1992-10-18'
}]
with open('data.json', 'w') as file:
    file.write(json.dumps(data))
```

利用 dumps() 方法我们可以将 Json 对象转为字符串，然后再调用文件的 write() 方法即可写入到文本

另外如果我们想保存 Json 的格式，可以再加一个参数 indent，代表缩进字符个数。

```
with open('data.json', 'w') as file:
    file.write(json.dumps(data, indent=2))
```

这样得到的内容会自动带有缩进，格式会更加清晰。 另外如果 Json 中包含中文字符，例如我们将之前的 Json 的部分值改为中文，再用之前的方法写入到文本。

```
import json

data = [{
    'name': '王伟',
    'gender': '男',
    'birthday': '1992-10-18'
}]
with open('data.json', 'w') as file:
    file.write(json.dumps(data, indent=2))
```

中文字符都变成了 Unicode 字符，这并不是我们想要的结果。 为了输出中文，我们还需要指定一个参数 ensure_ascii 为 False，另外规定文件输出的编码。

```
with open('data.json', 'w', encoding='utf-8') as file:
    file.write(json.dumps(data, indent=2, ensure_ascii=False))
```

这样我们就可以输出 Json 为中文了，所以如果字典中带有中文的内容我们需要设置 ensure_ascii 参数为 False 才可正常写入中文。

3.CSV 文件存储

CSV，全称叫做 Comma-Separated Values，中文可以叫做逗号分隔值或字符分隔值，其文件以纯文本形式存储表格数据。该文件是一个字符序列，可以由任意数目的记录组成，记录间以某种换行符分隔，每条记录由字段组成，字段间的分隔符是其它字符或字符串，最常见的是逗号或制表符，不过所有记录都有完全相同的字段序列，相当于一个结构化表的纯文本形式，它相比 Excel 文件更加简介，XLS 文本是电子表格，它包含了文本、数值、公式和格式等内容，而 CSV 中不包含这些内容，就是特定字符分隔的纯文本，结构简单清晰，所以有时候我们用 CSV 来保存数据是比较方便的，本节我们来讲解下 Python 读取和写入 CSV 文件的过程。

1.写入

在这里我们先看一个最简单的例子：

```
import csv

with open('data.csv', 'w') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['id', 'name', 'age'])
    writer.writerow(['10001', 'Mike', 20])
    writer.writerow(['10002', 'Bob', 22])
    writer.writerow(['10003', 'Jordan', 21])
```

首先打开了一个 data.csv 文件，然后指定了打开的模式为 w，即写入，获得文件句柄，随后调用 csv 库的 writer() 方法初始化一个写入对象，传入该句柄，然后调用 writerow() 方法传入每行的数据即可完成写入。

运行结束后会生成一个名为 data.csv 的文件，数据就成功写入了，直接文本形式打开的话内容如下：

```
id,name,age
10001,Mike,20
10002,Bob,22
10003,Jordan,21
```

可以看到写入的文本默认是以逗号分隔的，调用一次 writerow() 方法即可写入一行数据，我们用 Excel 打开

如果我们想修改列与列之间的分隔符可以传入 delimiter 参数，代码如下：

```
import csv

with open('data.csv', 'w') as csvfile:
    writer = csv.writer(csvfile, delimiter=' ')
    writer.writerow(['id', 'name', 'age'])
    writer.writerow(['10001', 'Mike', 20])
    writer.writerow(['10002', 'Bob', 22])
    writer.writerow(['10003', 'Jordan', 21])
```

例如这里在初始化写入对象的时候传入 delimiter 为空格，这样输出的结果的每一列就是以空格分隔的了，内容如下：

```
id name age
10001 Mike 20
10002 Bob 22
10003 Jordan 21
```

另外我们也可以调用 writerows() 方法同时写入多行，此时参数就需要为二维列表，例如：

```
import csv

with open('data.csv', 'w') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(['id', 'name', 'age'])
    writer.writerows([['10001', 'Mike', 20], ['10002', 'Bob', 22], ['10003', 'Jor
dan', 21]])
```

输出效果是相同的，内容如下：

```
id,name,age
10001,Mike,20
```

```
10002,Bob,22
10003,Jordan,21
```

但是一般情况下爬虫爬取的都是结构化数据，我们一般会用字典来表示，在 csv 库中也提供了字典的写入方式，实例如下：

```
import csv

with open('data.csv', 'w') as csvfile:
    fieldnames = ['id', 'name', 'age']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerow({'id': '10001', 'name': 'Mike', 'age': 20})
    writer.writerow({'id': '10002', 'name': 'Bob', 'age': 22})
    writer.writerow({'id': '10003', 'name': 'Jordan', 'age': 21})
```

在这里我们先定义了三个字段，用 fieldnames 表示，然后传给 DictWriter 初始化一个字典写入对象，然后可以先调用 writeheader() 方法先写入头信息，然后再调用 writerow() 方法传入相应字典即可，最终写入的结果是完全相同的，内容如下：

```
id,name,age
10001,Mike,20
10002,Bob,22
10003,Jordan,21
```

这样我们就可以完成字典到 CSV 文件的写入了。

另外如果我们想追加写入的话可以修改文件的打开模式，如将 open() 函数的第二个参数改成 a 就可以变成追加写入，代码如下：

```
import csv

with open('data.csv', 'a') as csvfile:
    fieldnames = ['id', 'name', 'age']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writerow({'id': '10004', 'name': 'Durant', 'age': 22})
```

这样在上面的基础上再执行这段代码，文件内容便会变成：

```
id,name,age
10001,Mike,20
10002,Bob,22
10003,Jordan,21
10004,Durant,22
```

可见数据被追加写入到了文件中。

如果我们要写入中文内容的话可能会遇到字符编码的问题，此时我们需要给 open() 参数指定一个编码格式，比如这里再写入一行包含中文的数据，代码需要改写如下：

```
import csv

with open('data.csv', 'a', encoding='utf-8') as csvfile:
    fieldnames = ['id', 'name', 'age']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writerow({'id': '10005', 'name': '王伟', 'age': 22})
```

在这里需要给 open() 函数指定编码，否则可能会发生编码错误。

以上便是 CSV 文件的写入方法。

另外如果我们接触过 Pandas 等库的话，可以调用 DataFrame 对象的 to_csv() 方法也可以非常方便地将数据写入到 CSV 文件中。

2.读取

我们同样可以使用 csv 库来读取 CSV 文件，例如我们现在将刚才写入的文件内容读取出来，代码如下：

```
import csv

with open('data.csv', 'r', encoding='utf-8') as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        print(row)
```

运行结果：

```
['id', 'name', 'age']
['10001', 'Mike', '20']
['10002', 'Bob', '22']
['10003', 'Jordan', '21']
['10004', 'Durant', '22']
['10005', '王伟', '22']
```

在这里我们构造的是 Reader 对象，通过遍历输出了每行的内容，每一行都是一个列表形式，注意在这里如果 CSV 文件中包含中文的话需要指定文件编码。

另外如果我们接触过 Pandas 的话，可以利用 read_csv() 方法将数据从 CSV 中读取出来，例如：

```
import pandas  as pd

df = pd.read_csv('data.csv')
print(df)
```

运行结果：

```
      id    name  age
0  10001    Mike   20
1  10002     Bob   22
2  10003  Jordan   21
3  10004  Durant   22
4  10005      王伟   22
```

在做数据分析的时候此种方法用的比较多，也是一种比较方便的读取 CSV 文件的方法。