(2) read( ): 读取 robots.txt 文件并进行分析。如果不调用该方法，后续的操作判断都是 False，因此一定要调用

(3) parse( ): 用于解析 robots.txt 文件，传入参数是 robots.txt 某些行的内容，该方法会按照 robots.txt 的语法规则来分析.

(4) can_fetch( ): 传入两个参数，一个是 User-agent，另一个是要抓取的 URL，它表示该搜索引擎是否可以抓取该 URL，返回结果为 True 或 False.

(5) mtime( ): 返回上次抓取分析 robots.txt 的时间。可用于长时间分析和抓取的搜索爬虫，用来定期检查来抓取最新的 robots.txt

(6) modified( ): 将当前时间设置为上次抓取和分析 robots.txt 的时间.

3. 实例:

```
from urllib.robotparser import RobotFileParser
rp = RobotFileParser( ) ⇒ 创建 RobotFileParser( ) 对象
rp.set_url ('http://www.jianshu.com/robots.txt') ⇒ 加入 URL
rp.read ( ) ⇒ 读取分析
print (rp.can_fetch ('*', 'http://www.jianshu.com/p/b67554025d7d'))
print (rp.can_fetch ('*', 'http://www.jianshu.com/search?q=python&page=1&type=collections'))
```

判断是否能爬取.

3.2 使用 requests

§ 3.2.1 基本用法

一、准备工作：安装 request 库

二、GET 请求

1. 基本实例:

```
import requests
r = requests.get ('http://httpbin.org/get')
print (r.text)
```

2、参数设置

如果要在请求中附加额外信息，可以配置 params 参数

eg:

```
import requests
data = {
    'name': 'germey',
    'age': 22
}
```

```
r = request.get('http://httpbin.org/get', params = data)
print(r.text)
```

这样我们就会发现在请求信息已经加入了想要添加的信息，同时，请求的链接会自动构造为 http://httpbin.org/get?age=22&name=germey

注意：这时返回的结果是 str 类型，但也是 JSON 格式的，所以如果我们想解析返回结果，得到一个字典格式的话，可以直接调用 json() 方法；但如果返回结果不是 JSON 格式，便会出现解析错误。

3. 抓取网页

上述请求链接返回的是 JSON 形式的字符串，如果请求普通的网页，就能获得相应的内容。

实例见 P125 或 github 仓库。

4. 抓取二进制数据

(1) 实例：抓取 Github 的站点图标

```
import requests
r = requests.get("https://github.com/favicon.ico")
print(r.text)    ⇒ 返回图片的 text 属性，由于图片是二进制数据，在转化为 str 字符串类型时会出现乱码
print(r.content)
```

↓

返回图片的 content 属性，这代表可以返回 bytes 类型的数据

(2) 将提取到的图片保存下来：

```
import requests
r = requests.get("http://github.com/favicon.ico")
with open('favicon.ico', 'wb') as f:
    f.write(r.content)
```

→ 这里用到 open() 方法，它的第一个参数是文件名，第二个参数表示以二进制写的形式打开。

5. 添加 headers

(1) 该部分本质上属于前面参数配置部分，我们可以通过 headers 参数来传递头信息。

(2) 具体做法和 urllib.request.urlopen 类似

```
eg: headers = {
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4)
    }
    r = request.get('https://www.zhihu.com/explore', headers = headers)
```

AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.116 Safari/537.36

除了 User-Agent，也可以在 headers 中添加其他字段的信息。

三、POST请求

1. 工作原理与GET请求类似，通过data参数来传递POST请求所提交的内容。

2. 实例：

```
import requests
data = {'name': 'germey', 'age': '22'}
r = request.post("http://httpbin.org/post", data=data)
print(r.text)
```

⇒ 得到的结果中可以发现post提交的信息以form表单的形式存在。

四、响应

1. 获取响应信息的属性和方法

(1) 响应内容：{.text 属性 / .content 属性}

(2) 状态码：.status_code 属性

(3) 响应头：.headers 属性 ⇒ 得到的结果为CaseInsensitiveDict类型

(4) Cookies：.cookies 属性 ⇒ 得到的结果为RequestsCookieJar类型

(5) url：.url 属性

(6) 请求历史：.history 属性

2. 状态码详解：

(1) 状态码常用来判断请求是否成功，而requests还提供了一个内置的状态码查询对象requests.codes

(2) 实例：

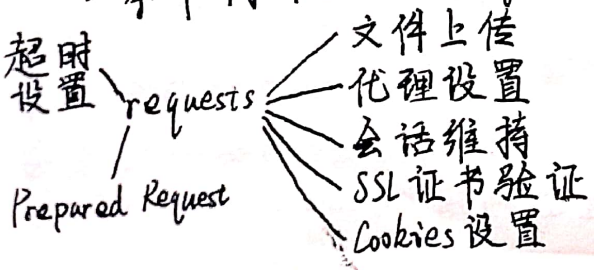
```
import requests
r = requests.get('http://www.jianshu.com')
exit() if not r.status_code == requests.codes.ok else print('Request Successfully')
```

(3) 除了上面实例中的ok条件码，还有很多不同的内置成功返回码

*各种返回码和相应的查询条件：见书 P_{128} ~ P_{129}

§3.2.2 高级用法

⇒ 本节将介绍一些requests的高级用法



一、文件上传

1. 实例：
```
import requests
files = {'file': open('favicon.ico', 'rb')}
r=requests.post("http://httpbin.org/post", files=files)
print(r.text)
```
↗ 该部分替换为要传递的文件名即可.

2. 原理：将文件以files参数的形式通过requests.post传递进去

3. 注意：requests提交的文件必须和当前脚本在同一目录下.

二、Cookies

1. 获取cookies

实例：
```
import requests
r= requests.get("https://www.baidu.com")
print(r.cookies)
for key, value in r.cookies.items():
    print(key +"="+ value)
```
直接获取Cookies并打印

⇒ 这里由于cookies属性返回的是RequestCookieJar类型，所以先用items()方法将其转化为元组组成的列表，遍历输出每一个Cookie的名称和值.

2. 利用Cookies来维持登录状态.

(1) 可以直接打开网页找到Headers中的Cookie内容并复制下来，然后直接粘贴到自己的脚本Cookies中，最后发送请求.

(2) 也可以通过cookies参数来设置，但这样就需要构造RequestCookieJar对象，而且需要分割一下cookies.

① 实例：
```
import requests
cookies = '----------'
jar = requests.cookies.RequestsCookieJar()
headers = {
    'Host': 'www.zhihu.com',
    'User-Agent': 'Mozilla/5.0 ----------'
}
for cookie in cookies.split(';'):
    key, value = cookie.split('=', 1)
    jar.set(key, value)
r = requests.get("http://www.zhihu.com", cookies=jar, headers=headers)
print(r.text)
```
↗ 该部分为cookies内容

⇒ 新建一个RequestCookieJar对象

↗ 伪装浏览器的User-Agent部分

② 原理过程：首先新建一个RequestCookieJar对象，然后将复制下来的cookies利用split()方法进行分割，再利用set()方法设置好每个cookie的key和value，然后通过调用requests的get()方法并传递给cookies参数即可.

三、会话维持

1. 关键问题：在请求时维持同一个会话，相当于打开一个新的浏览器选项卡，而不是新开一个浏览器。

2. 解决方法：利用 Session 对象

实例：
```
import requests
s = requests.Session()  ⇒ 新建一个 Session 对象
s.get("http://httpbin.org/cookies/set/number/123456789")
r = s.get("http://httpbin.org/cookies")
print(r.text)
```
（利用同一个 Session 对象进行请求）

3. 作用：可以做到模拟同一个会话而不用担心 cookies 的问题。它通常用于模拟登录成功后进行下一步的操作，在一个浏览器中打开同一站点的不同页面。

四、SSL 证书验证。

1. verify 参数：当发送 HTTP 请求时，它会检查 SSL 证书，我们可以使用 verify 参数来控制是否检查证书。如果不加 verify 参数，则默认为 True，会自动验证。如果将 verify 参数设置为 False 就可以不验证证书。

2. 实例：
```
import requests
response = requests.get('https://www.12306.cn', verify=False)
print(response.status_code)
```
⇒ 会打印出成功状态码 200，但是会出现一个建议我们给它指定证书的警告。

3. 忽略警告：

(1) 直接设置忽略警告：
```
import requests
from requests.packages import urllib3
urllib3.disable_warnings()  ⇒ 设置忽略警告
response = requests.get('https://www.12306.cn', verify=False)
print(response.status_code)
```

(2) 捕获警告到日志
```
import requests
import logging
logging.captureWarnings(True)  ⇒ 捕获警告到日志。
response = requests.get('https://www.12306.cn', verify=False)
print(response.status_code)
```

4. 指定一个本地证书用作客户端证书，这可以是单个文件（包含密钥和证书）或一个包含两个文件路径的元组

实例：
```
import requests
response = requests.get('https://www.12306.cn', cert=('/path/server.crt', '/path/key'))
print(response.status_code)
```

注意：上述代码实例仅供演示，实际运行我们需要有 crt 和 key 文件，并且指定它们的路径。同时，本地私有证书的 key 必须是解密状态，加密状态的 key 是不支持的。

五、代理设置

1. 直接设置 proxies 参数

实例：
```
import requests
proxies = {
  "http": "http://10.10.1.10:3128",
  "https": "http://10.10.1.10:1080",
}
requests.get("https://www.taobao.com", proxies=proxies)
```

注意 proxies 参数的设置格式。

2. 如果代理需要使用 HTTP Basic Auth，可以使用类似 http://user:password@host:port 这样的语法来设置代理

实例：
```
import requests
proxies = {
    "http": "http://user:password@10.10.1.10:3128",
}
request.get("https://www.taobao.com", proxies=proxies)
```

3. 除了基本的 HTTP 代理外，requests 还支持 SOCKS 协议的代理

实例：
```
import requests
proxies = {
    "http": "socks5://user:password@host:port",
    "https": "socks5://user:password@host:port",
}
requests.get("https://www.taobao.com", proxies=proxies)
```

六、超时设置

1. timeout 参数：这个时间的计算是发出请求到服务器返回响应的时间。

它的作用是设置一个超时时间，如果超过这个时间还没有得到响应就会报错，防止服务器不能及时响应。

2. 设置方式：直接对 timeout 参数进行设置即可，单位为秒。

(1) 直接设置：timeout=1 表示超时时间为 1 秒

(2) 分别设置：实际上 timeout 分为连接(connect)和读取(read)两部分，所以如果要分别指定，则要传入一个元组，如 timeout=(5,11)

(3) 永久等待：不设置 timeout 或 timeout 设置为 None。

七、身份验证

1. 使用 HTTPBasicAuth 类

实例：
```
import requests
from requests.auth import HTTPBasicAuth
r = requests.get('http://localhost:5000', auth=HTTPBasicAuth('username', 'password'))
print(r.status_code)
```

auth 参数用来进行身份验证，但它的参数必须是一个 HTTPBasicAuth 对象

2. 上述代码中，我们可以不用给每一个参数都传一个 HTTPBasicAuth 类，而是直接传递一个元组。

如：`r = requests.get('http://localhost:5000', auth=('username', 'password'))`

3. 还可以通过 OAuth 进行验证，这里不深入研究

八、Prepared Request

1. 什么是 Prepared Request？

与之前 urllib 库相似，那时我们尝试构建一个 Request 对象，来表示各个参数；在 requests 库中，Prepared Request 的作用就和 urllib 中的 Request 相同。

2. 实例：
```
from requests import Request, Session
url = 'http://httpbin.org/post'
data = {
    'name': 'germey',
}
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_4) ......'
}
s = Session()
req = Request('POST', url, data=data, headers=headers)
prepped = s.prepare_request(req)
r = s.send(prepped)
print(r.text)
```

设置好各个参数

⇒ 先建立一个 Session 对象

再建立一个 Request 对象 ⇐

⇒ 利用 Session 的 prepare_request() 方法将其转换为一个 Prepared Request 对象

⇒ 最后调用 send() 方法发送即可