4.1 scrapy 框架爬虫简介

4.1.1 安装 scrapy 框架

在 Python 的安装目录中找到 scripts 目录，在 scripts 中执行：

pip install scrapy

程序执行完毕就完成安装。

4.1.2 建立 scrapy 项目

1. 进入命令行窗体，在 c 盘中建立一个文件夹例如 example，进入 c:\example 然后执行命令：

scrapy startproject demo

该命令时建立一个名称为 demo 的 scrapy 项目，如图 4-2-1。

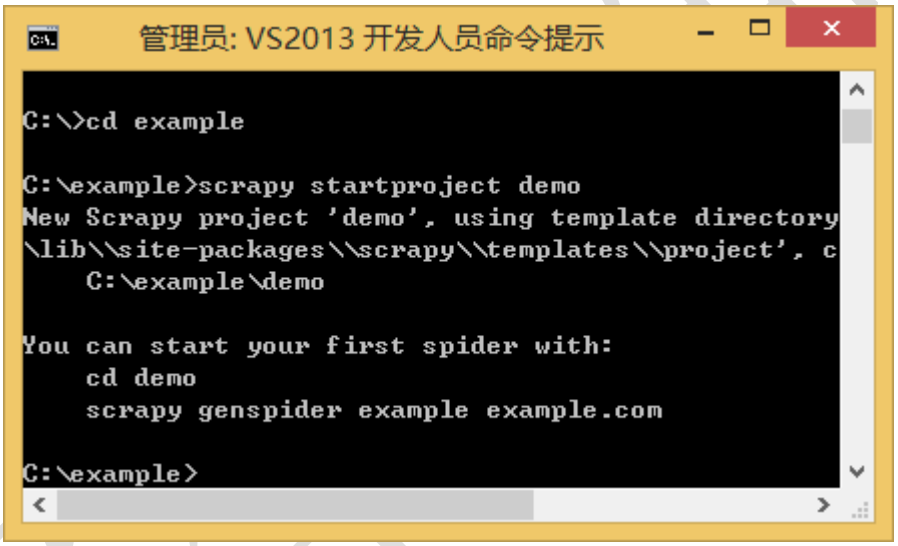

图 4-2-1 建立 scrapy 项目

2. scrapy 项目建立后会在 c:\example 中建立 demo 文件夹，同时下面还有另外一个 demo 子文件夹，结构果如图 4-2-2 所示。

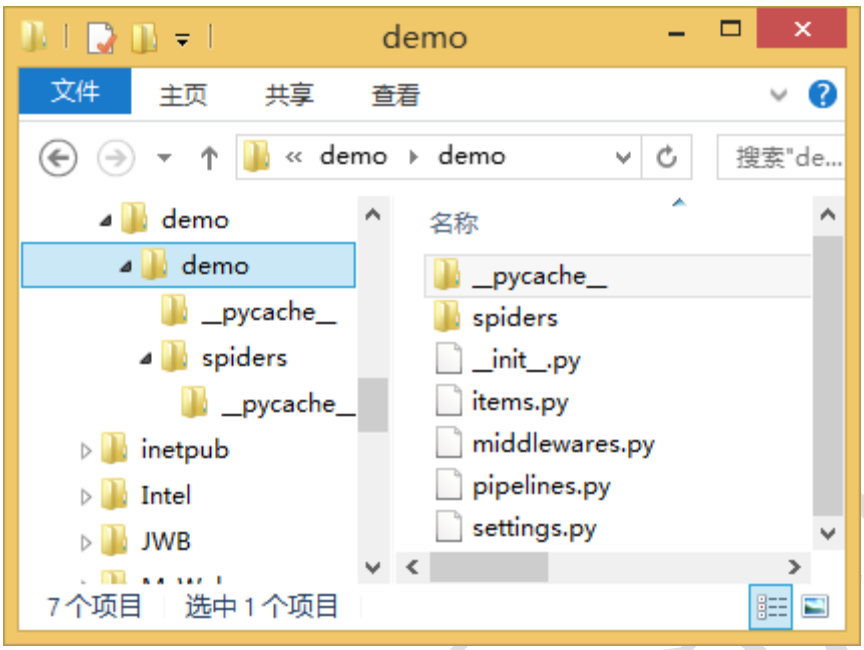

图 4-2-2 scrapy 的项目结构

3. 用 PyCharm 打开 example 项目，如图 4-2-3 所示。

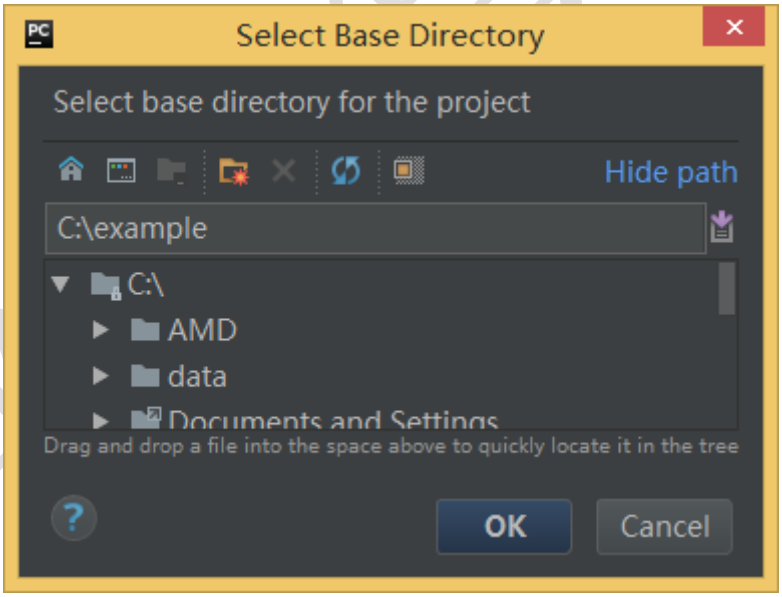

图 4-2-3 PyCharm 打开 example 项目

4. 为了测试我们的这个 scrapy 项目，我们先建立一个 Web 网站，可以在 c:\example 中建立一个 server.py 程序如下：

```
import flask

app=flask.Flask(__name__)

@app.route("/")
def index():
```

```
    return "测试 scrapy"

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

这个程序建立好后就可以执行了，建立 http://127.0.0.1:5000 的网站，访问这个网站返回"测试 scrapy"。

5. 在 c:\example\demo\demo\spider 文件夹中建立一个自己的 python 文件，例如 MySpider.py，这个就是我们的爬虫程序了，这个程序如下：

```
import scrapy

class MySpider(scrapy.Spider):
    name = "mySpider"

    def start_requests(self):
        url ='http://127.0.0.1:5000'
        yield scrapy.Request(url=url,callback=self.parse)

    def parse(self, response):
        print(response.url)
        data=response.body.decode()
        print(data)
```

6. 在 c:\example\demo\demo 文件夹中建立一个执行程序例如 run.py，程序如下：

```
from scrapy import cmdline
cmdline.execute("scrapy crawl mySpider -s LOG_ENABLED=False".split())
```

7. 保存这些程序并运行 run.py，可以在 PyCharm 中看到结果：

```
http://127.0.0.1:5000
测试 scrapy
```

由此可见程序 MySpider.py 访问了我们自己的 Web 网站并爬取了网站的网页。这个项目初步看起来有点复杂，但是仔细分析也不难理解，下面我们来分析 MySpider.py 程序。

(1) import scrapy

引入 scrapy 程序包，这个包中有一个请求对象 Request 与一个响应对象 Response 类。

(2)

```
class MySpider(scrapy.Spider):
    name = "mySpider"
```

任何一个爬虫程序类都继承于 scrapy.Spider 类，任何一个爬虫程序都有一个名字，这个名字在整个爬虫项目中是唯一的，我们这个爬虫名字为"mySpider"。

(3)

```
    def start_requests(self):
        url ='http://127.0.0.1:5000'
```

yield scrapy.Request(url=url,callback=self.parse)

这个地址 url 是爬虫程序的入口地址，这个 start_requests 函数是程序的入口函数。程序开始时确定要爬取的网站地址，然后建立一个 scrapy.Request 请求类，向这个类提供 url 参数，指明要爬取的网页地址。爬取网页完成后就执行默认的回调函数 parse。

值得指出的是 scrapy 的执行过程是异步进行的，即指定一个 url 网址开始爬取数据时，程序不用一直等待这个网站的相应，如果网站迟迟不响应，那么整个程序既不是卡死了！scrapy 不这么做，它提供一个回调函数机制，爬取网站时同时提供一个回调函数，当网站相应后就触发执行这个回调函数，网站什么时候响应就什么时候调用这个回调函数，这样对于响应时间很长的网站也不怕了。

(4)

```
def parse(self, response):
    print(response.url)
    data=response.body.decode()
    print(data)
```

回调函数 parse 包含一个 scrapy.Response 类的对象 response，它是网站相应的一切信息，其中 response.url 是网站的网址，response.body 是网站相应的二进制数据，即网页的内容。通过 decode()解码后变成字符串，我们就可以 print 出来了。

到目前为止爬虫程序就编写好了，但是这个程序 MySpider.py 只是一个类，不能单独执行，有爱执行这个爬虫程序必须使用 scrapy 中专门的命令 scrapy crawl。我们回到命令行窗体，在 c:\example\demo\demo 中执行命令：

scrapy crawl mySpider -s LOG_ENABLED=False

那么就可以看到执行的结果如图 4-1-X 所示，其中 mySpider 就是我们爬虫程序的名称，后面的参数是不显示调试信息。

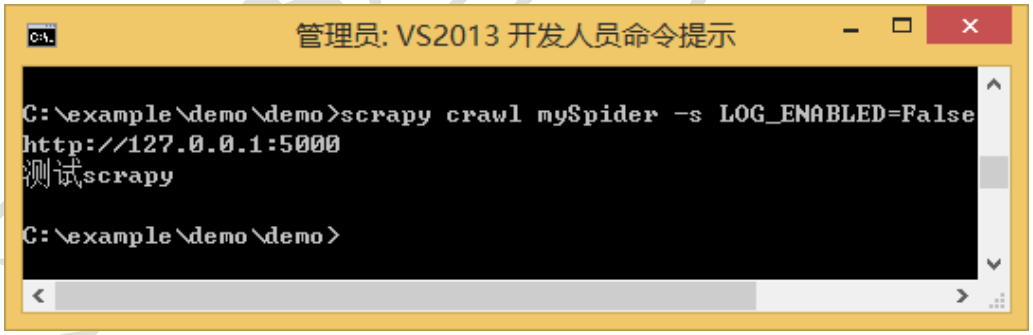

图 4-1-x 命令行执行 scrapy 程序

但是这样需要我们从 PyCharm 与命令行窗体之间来回切换，太麻烦。为了简单起见，我们专门设计一个 Python 程序 run.py，它包含执行命令行的语句：

```
from scrapy import cmdline
cmdline.execute("scrapy crawl mySpider -s LOG_ENABLED=False".split())
```

直接运行 run.py 就可以执行 MySpider.py 的爬虫程序了，效果与在命令行窗体中执行一样，结果还直接显示在 PyCharm 中，对于我们开发爬虫程序十分方便。

4.1.3 入口函数与入口地址

在程序中我们使用了入口函数：

```
def start_requests(self):
```

```
        url ='http://127.0.0.1:5000'
        yield scrapy.Request(url=url,callback=self.parse)
```

实际上这个函数也可以用 start_urls 的入口地址来代替：

```
    start_urls=['http://127.0.0.1:5000']
```

入口地址可以有多个，因此 start_urls 是一个列表。入口函数一入口地址的作用是一样的，都是引导函数的开始。

4.1.4 Python 的 yield 语句

在入口函数中我们看到一条 yield 语句，yield 是 Python 的一种特殊语句，主要作用是返回一个值等待被取走。我们先看一个实例：

```
def fun():
    s=['a','b','c']
    for x in s:
        yield x
    print("fun End")

f=fun()
print(f)
for e in f:
    print(e)
```

程序结果：

```
<generator object fun at 0x0000003EA99BD728>
a
b
c
fun End
```

由此可见，fun 返回一个 generator 的对象，这种对象包含一系列的元素，可以使用 for 循环提取，执行循环的过程如下：

```
for e in f:
    print(e)
```

第一次 for e in f 循环，f 执行到 yield 语句，就返回一个值'a'，for 循环从 f 抽取的元素是"a"，然后 e='a'打印 a。fun 中执行到 yield 时会等待 yield 返回的值被抽走，同时 fun 停留在 yield 语句，一旦被抽走后，再次循环，yield 返回'b'。

第二次 for e in f 循环，抽取 f 函数的 'b'元素，打印出'b'，然后 f 中的循环继续 yield 返回'c'元素。

第三次 for e in f 循环，抽取 f 函数的 'c'元素，打印出'c'，然后 f 中的循环就结束了，显示 fun End，随后 for e in f 循环中 f 也没有元素可以继续抽取了，也结束，随后程序结束。

只要包含 yield 语句的函数都返回一个 generator 的可循环对象，执行到 yield 语句是只是返回一个值，等待调用循环抽取，一旦调用循环抽取后，函数又继续进行。这个过程非常类似两个线程的协作过程，当提供数据的一方准备好数据把 yield 提交数据时，就等待另外一方把数据抽取走，如果不抽走 yield 就一直等待，一旦抽走后数据提供方继续它的程序，一直等到出现下一次 yield 或者程序结束。

scrapy 的框架使用的是异步执行的过程，因此大量使用 yield 语句。