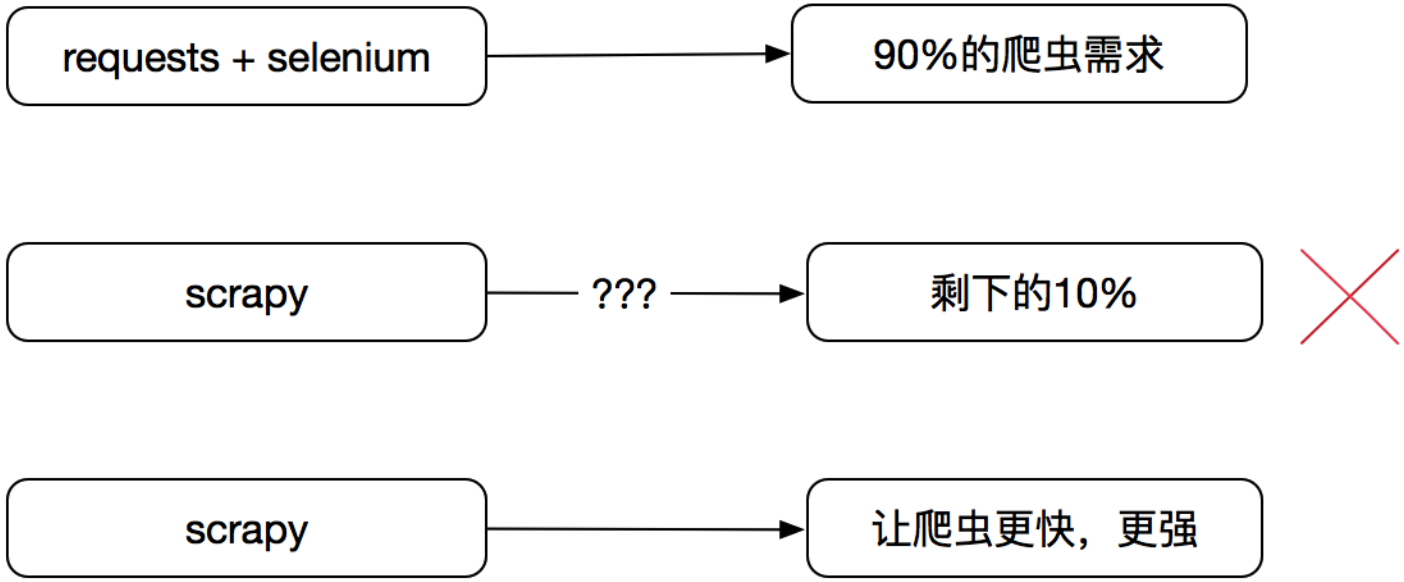
Scrapy框架学习

第四部分课程概要

1. scrapy的基础概念
2. scrapy的工作流程
3. scrapy入门使用
4. scrapy的深入
5. crawlspider的使用

为什么要学习scrapy



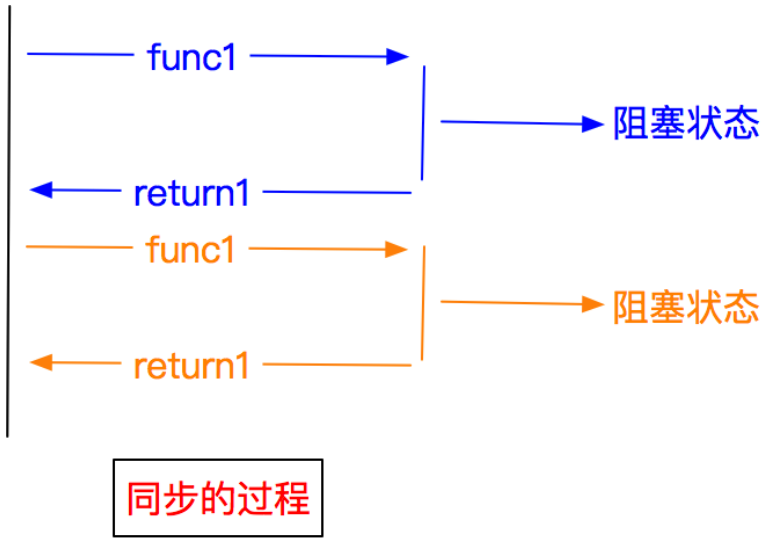
什么是scrapy

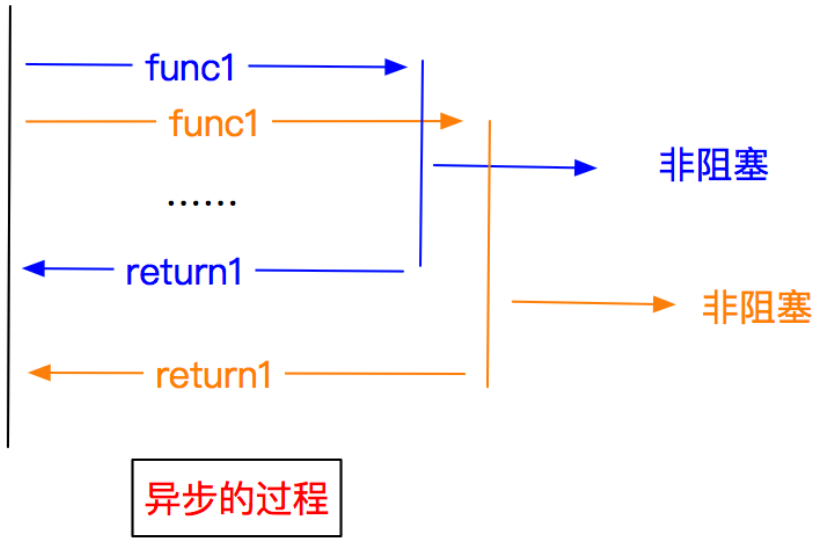
Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架，我们只需要实现少量的代码，就能够快速的抓取

Scrapy 使用了 Twisted['twɪstɪd]异步网络框架，可以加快我们的下载速度。

http://scrapy-chs.readthedocs.io/zh_CN/1.0/intro/overview.html

补充：异步和非阻塞的区别：



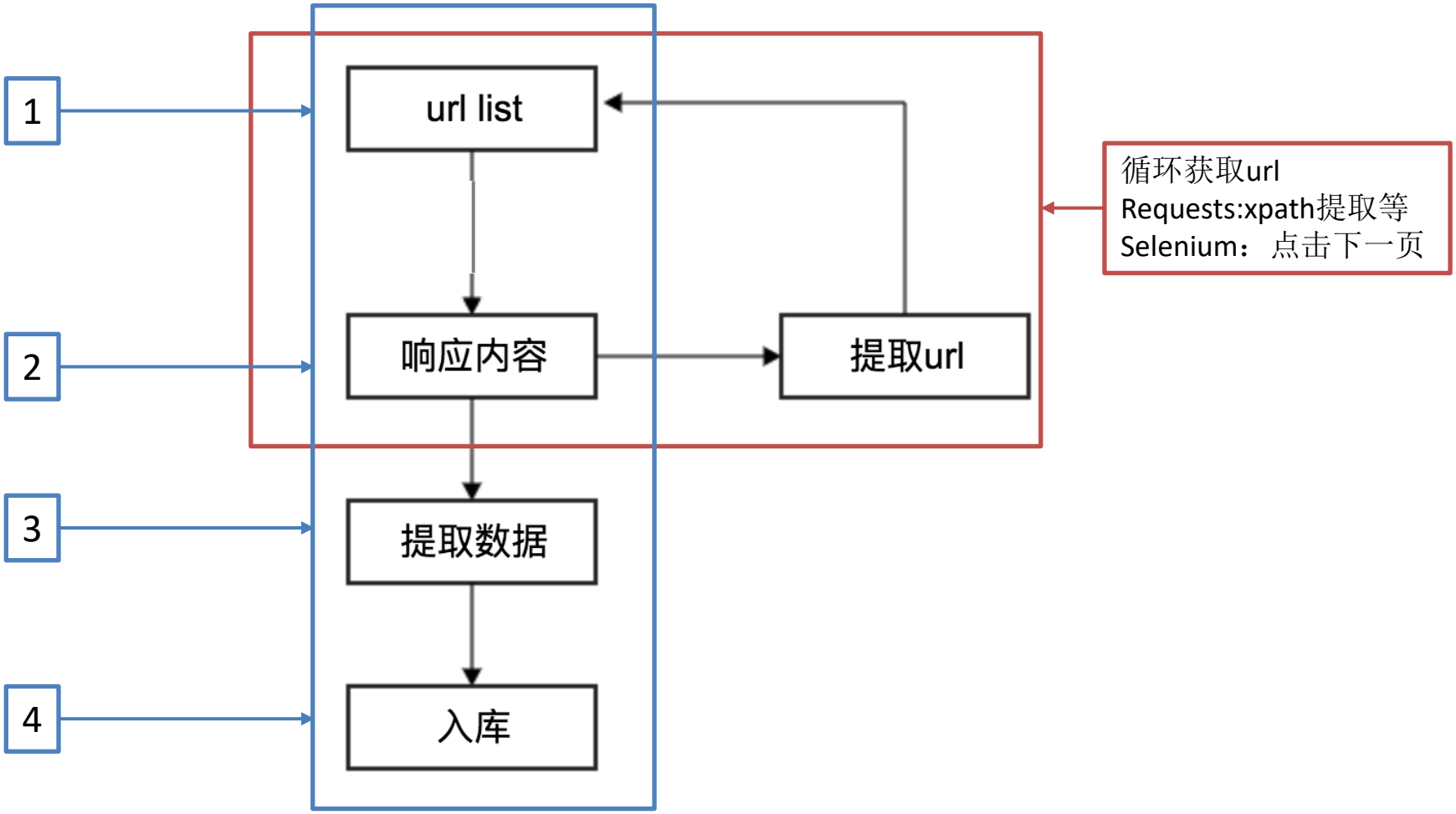


异步：**调用**在发出之后，这个调用就直接返回，不管有无结果
非阻塞：关注的是程序在等待调用结果（消息，返回值）时的状态，指在不能立刻得到结果之前，该调用不会阻塞当前线程。

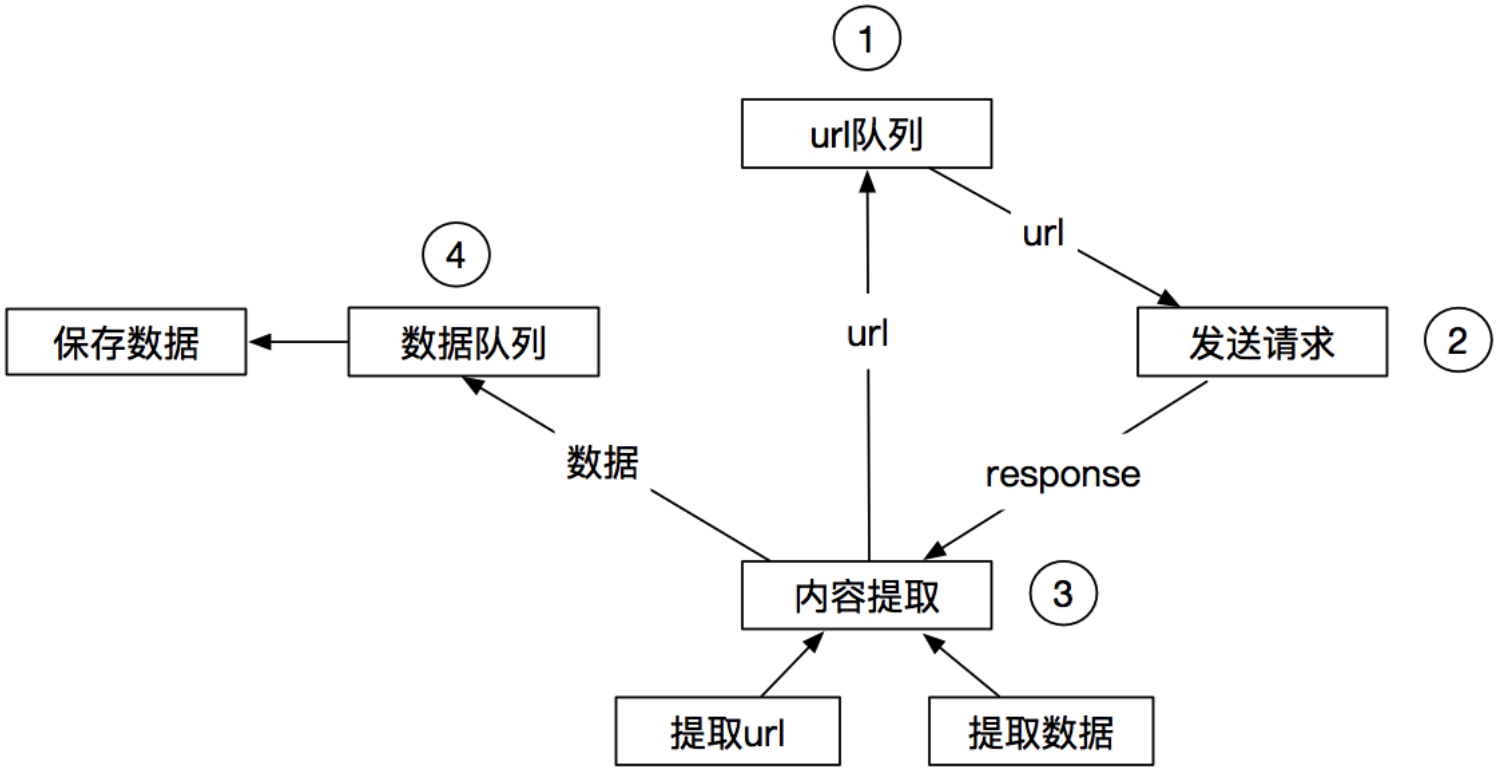
scrapy是工作流程

那么，scrapy是如何帮助我们抓取数据的呢？

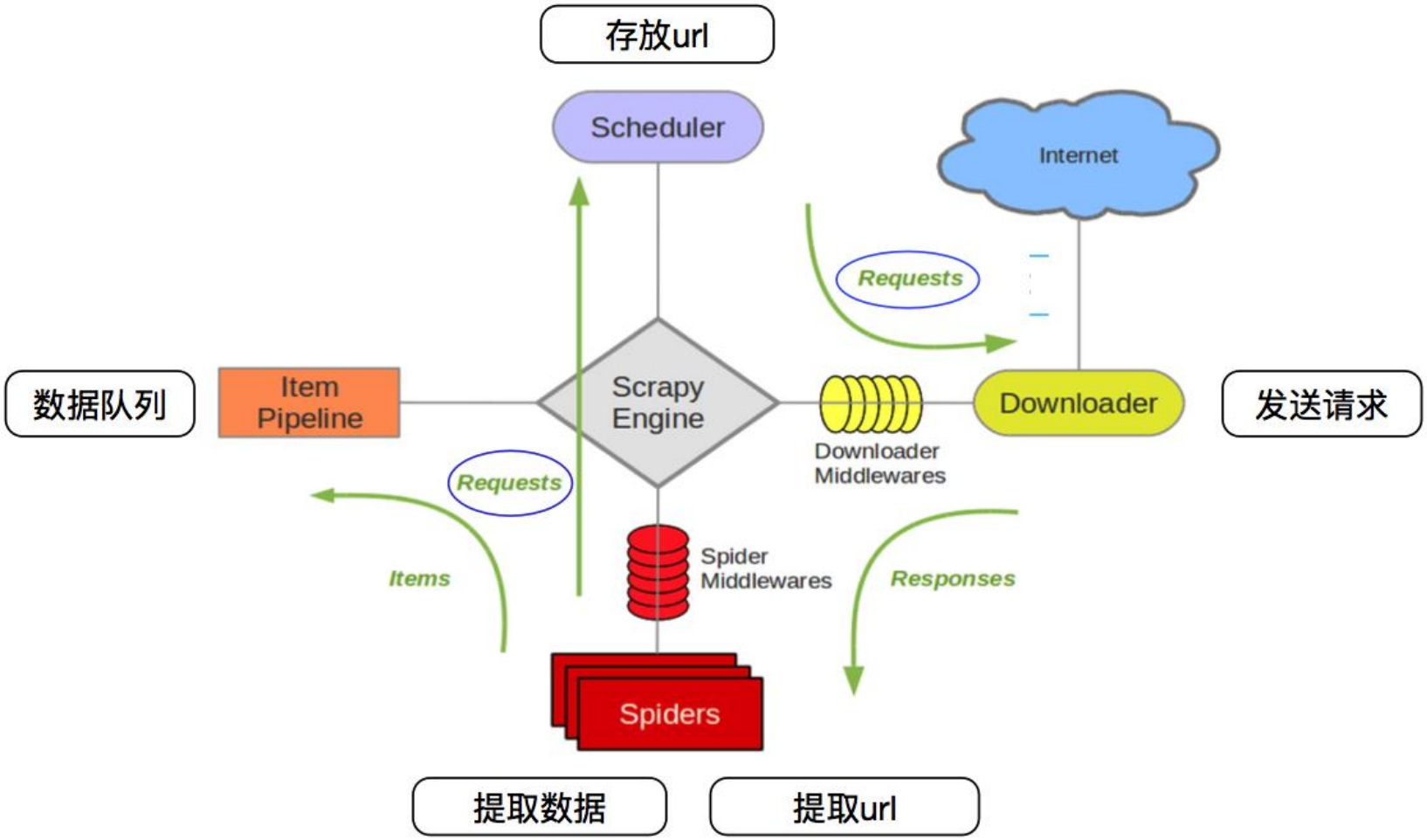
回顾前面的爬虫流程


1
2
3
4
url list
响应内容
提取数据
入库
提取url
循环获取url
Requests:xpath提取等
Selenium： 点击下一页

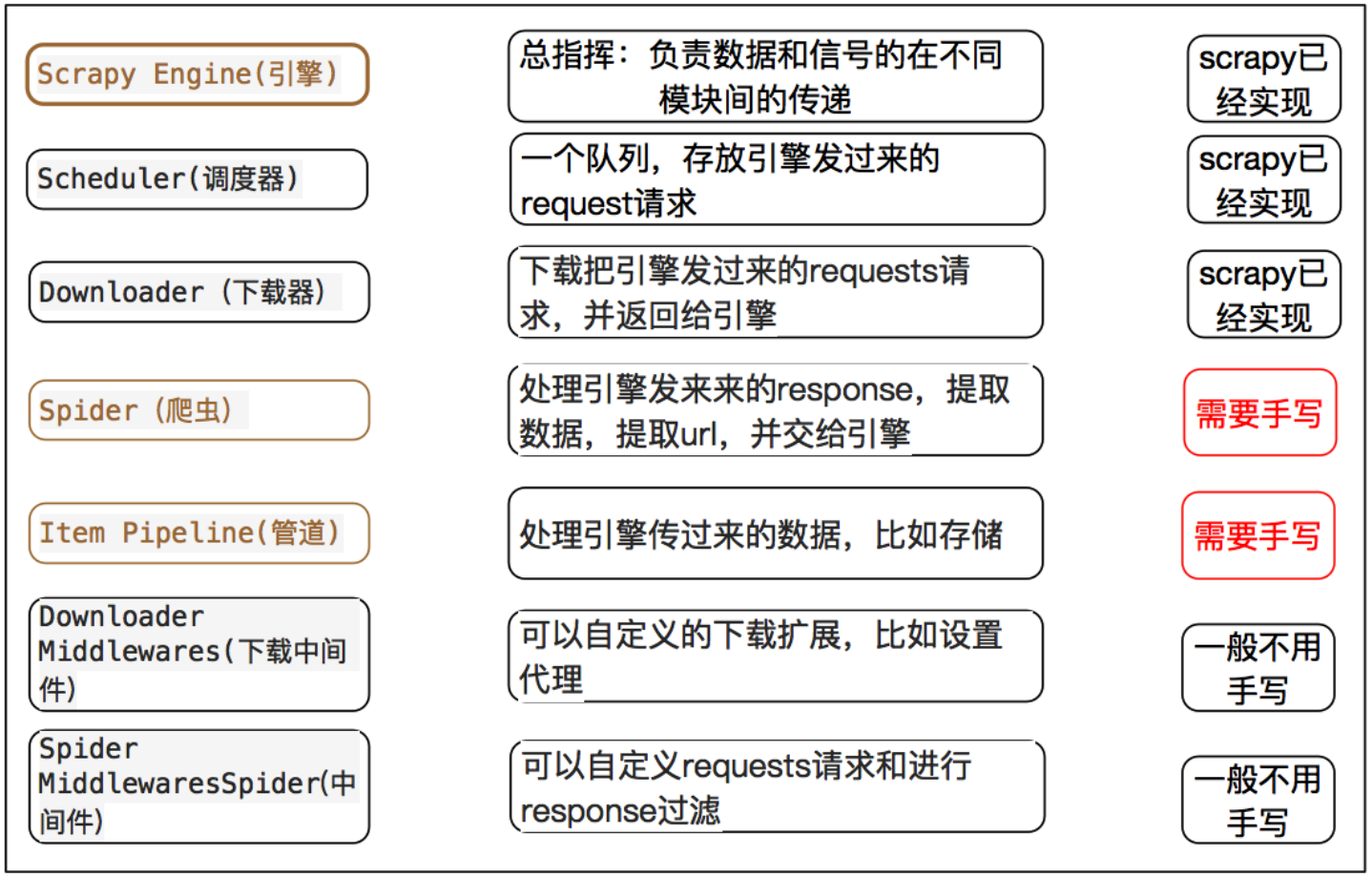
另一种形式爬虫流程


1
url队列
url
4
保存数据
数据队列
url
发送请求
2
数据
response
内容提取
3
提取url
提取数据

scrapy的爬虫流程



We need know more



Scrapy 入门

1. 创建一个scrapy项目
   - scrapy startproject mySpider
2. 生成一个爬虫
   - scrapy genspider itcast "itcast.cn”
3. 提取数据
   - 完善spider，使用xpath等方法
4. 保存数据
   - pipeline中保存数据

创建一个scrapy项目

命令：scrapy startproject +<项目名字>
scrapy startproject myspider

```
(python3) ➜  code tree
.
└── myspider
    ├── myspider
    │   ├── __init__.py
    │   ├── __pycache__
    │   ├── items.py          ——>自己预计需要爬取的内容
    │   ├── middlewares.py    ——>自定义中间件的文件
    │   ├── pipelines.py      ——> 管道，保存数据
    │   ├── settings.py       ——> 设置文件，UA，启动管道
    │   └── spiders           ——> 自己定义的spider的文件夹
    │       ├── __init__.py
    │       └── __pycache__
    └── scrapy.cfg            ——>项目的配置文件
```

创建一个爬虫

命令：scrapy genspider +<爬虫名字> + <允许爬取的域名>
scrapy genspider itcast “itcast.cn”

```
.
├── myspider
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-36.pyc
│   │   └── settings.cpython-36.pyc
│   ├── items.py        ——>自己预计需要爬取的内容
│   ├── middlewares.py      ——>自定义中间件的地方
│   ├── pipelines.py        ——> 管道，保存数据
│   ├── settings.py     ——> 设置文件，UA，启动管道
│   └── spiders     ——> 自己定义的spider的文件夹
│       ├── __init__.py
│       ├── __pycache__
│       │   └── __init__.cpython-36.pyc
│       └── itcast.py   ——> 定义spider的文件
└── scrapy.cfg
```

完善spider

```
class ItcastSpider(scrapy.Spider):  ——>自定义spider类，继承自scrapy.spider
    name = 'itcast'   ——> 爬虫名字 <爬虫启动时候使用：scrapy crawl itcast>
    allowed_domains = ['itcast.cn']  ——> 允许爬取的范围，防止爬虫爬到了别的网站
    start_urls = ['http://www.itcast.cn/channel/teacher.shtml']
                                        ——>开始爬取的地址
    def parse(self, response):  ——> 数据提起方法，接收下载中间件传过来的response
        names = response.xpath("//div[@class='tea_con']//li/div/h3/text()")
        print(names)            ——>返回包含选择器的列表
```

从选择器中提取字符串：

1、extract() 返回一个包含有字符串数据的列表

2、extract_first() 返回列表中的第一个字符串

注意：

1、spider中的parse方法名不能修改

2、需要爬取的url地址必须要属于allow_domain下的连接

3、respone.xpath()返回的是一个含有selector对象的列表

spider的数据传到pipeline

```
def parse(self, response):
    teachers = response.xpath("//div[@class='tea_con']//li")
                ——>xpath 分组提取
    for t in teachers:
        name = t.xpath("./div/h3/text()").extract_first()
        position = t.xpath("./div/h4/text()").extract_first()
        profile = t.xpath("./div/p/text()").extract_first()
        item = dict(
            name= name,
            position = position,
            profiel = profile
        )
        yield item    ——>yield 就可以了
```

为什么要使用yield?

让整个函数变成一个生成器，变成generator(生成器)有什么好处?

每次遍历的时候挨个读到内存中，不会导致内存的占用量瞬间变高

python3中range 和python2中的xrange 同理

使用pipeline

```
import json
class MyspiderPipeline(object):
    def process_item(self, item, spider): ——>实现存储方法
        with open('temp.txt','a') as f:
            json.dump(item,f,ensure_ascii=False,indent=2)
```

完成pipeline代码后，需要在setting中设置开启

```
# Configure item pipelines
# See http://scrapy.readthedocs.org/en/latest/topics/item-pipeline.html
ITEM_PIPELINES = {
   'myspider.pipelines.MyspiderPipeline': 300,
}
```

pipeline的位置　　权重

使用pipeline

从pipeline的字典形式可以看出来，pipeline可以有多个，而且确实pipeline能够定义多个

为什么需要多个pipeline：

1、可能会有多个spider，不同的pipeline处理不同的item的内容

2、一个spider的内容可能要做不同的操作，比如存入不同的数据库中

注意：

1、pipeline的权重越小优先级越高

2、pipeline中process_Item方法名不能修改为其他的名称

简单设置log

为了让我们自己希望输出到终端的内容能容易看一些：

我们可以在setting中设置log级别

在setting中添加一行（全部大写）：LOG_LEVEL = "WARNING"

默认终端显示的是debug级别的log信息

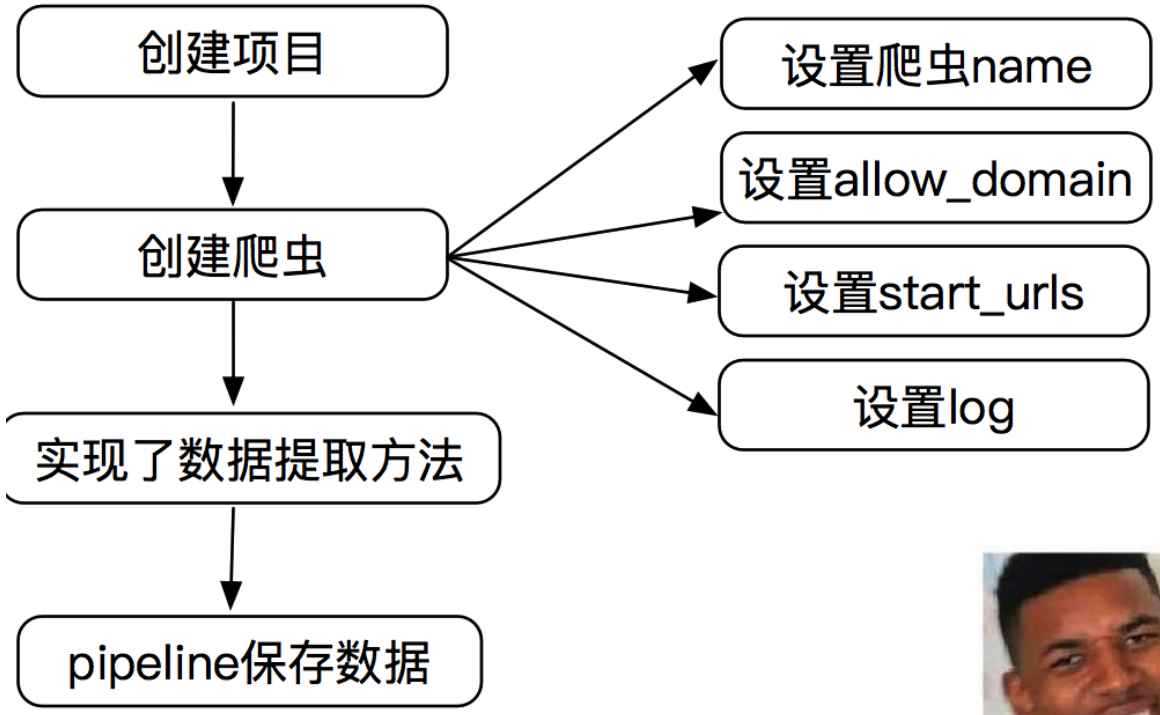
```
import logging
def process_item(self, item, spider):
    logging.warning(item)

>>2017-07-12 10:47:08 [root] WARNING: {'name':
```

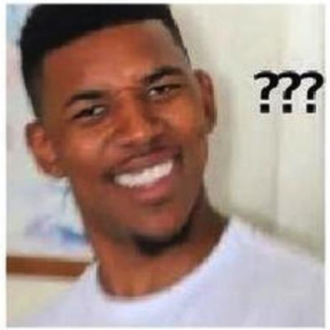
无法显示log位置

```
import logging
logger = logging.getLogger(__name__)

def process_item(self, item, spider):
    logger.warning(item)

>>2017-07-12 11:06:20 [myspider.pipelines] WARNING:
```

能够显示log位置

回顾：刚刚我们都干了什么





问题来了：如何实现翻页请求

回忆：

requests模块是如何发送翻页的请求的?

1、找到下一页地址

2、之后调用requests.get(url)

思路：

1、找到下一页的地址

2、构造一个关于下一页url地址的request请求传递给调度器

实现翻页请求

通过爬取腾讯招聘的页面的招聘信息,学习如何实现翻页请求
http://hr.tencent.com/position.php

```
next_page_url = response.xpath("//a[text()='下一页']/@href").extract()
while len(next_page_url)>0:
    yield scrapy.Request(next_page_url,callback=self.parse)
```

——>scrapy.Request能构建一个requests，同时指定提取数据的callback函数

在setting中设置User-Agent:
USER_AGENT = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/59.0.3071.115 Safari/537.36'

实现翻页请求

```
next_page_url = response.xpath("//a[text()='下一页']/@href").extract()
while len(next_page_url)>0:
    yield scrapy.Request(next_page_url,callback=self.parse)
```

——>scrapy.Request能构建一个requests，同时指定提取数据的callback函数

scrapy.Request知识点：

scrapy.Request(url [, callback, method='GET', headers, body, cookies, meta, dont_filter=False])

注：一般文档中方括号中的参数表示可选参数

scrapy.Request常用参数为：

callback:指定传入的url交给哪个解析函数去处理

meta：实现在不同的解析函数中传递数据，meta默认会携带部分信息，比如下载延迟，请求深度等

dont_filter： 让scrapy的去重不会过滤当前url,scrapy默认有url去重的功能,对需要重复请求的url有重要用途

Scrapy深入之定义Item

```
class MyspiderItem(scrapy.Item):  —>scrapy.Item也是一个字典
    # define the fields for your item here like:
    name = scrapy.Field() —>scrapy.Field()是一个字典
    pass
```

所以总的来说，可以把我们定义的MyspiderItem理解为一个字典

那么scrapy吃饱了撑着为什么要定义一个字典呢?
个人理解:

在获取到数据的时候，使用不同的Item来存放不同的数据

在把数据交给pipeline的时候，可以通过isinstance(item,MyspiderItem)来判断数据是属于哪个item，进行不同的数据(item)处理

Item的使用

通过爬取阳光热线问政平台来学习item的使用
目标：所有的投诉帖子的编号、帖子的url、帖子的标题和内容
url：
http://wz.sun0769.com/index.php/question/questionType?type=4&page=0

items文件中：

```
class YangguangItem(scrapy.Item):
    title = scrapy.Field()
    href = scrapy.Field()
    number = scrapy.Field()
    hanle_state = scrapy.Field()
    update_time = scrapy.Field()
    involved_department = scrapy.Field()
    content = scrapy.Field()
```

Spider中：

```
from yangguang.items import YangguangItem
item = YangguangItem()
```

—>实例化一个自定义的item
—>item的操作和字典一样

问题来了：在不同的解析函数中传递参数

```
def parse(self, response):
    tr_list = response.xpath('//div[@class="greyframe"]/table//table/tr')
    for i in tr_list:
        item = YangguangItem()
        .......
        .......
        #yield item
        yield scrapy.Request(item["href"],callback=self.get_content,meta={"item":item})

def get_content(self,response):
    item = response.meta['item']
    item["content"] = response.xpath("//div[@class='content text14_2']//text()").extract()
    yield item    ——>数据统一的传递pipeline
```

Scrapy深入之认识程序的debug信息

[scrapy.utils.log] INFO: Overridden settings:自己设置的setting的信息

[scrapy.middleware] INFO: Enabled extensions:启动的扩展，默认有一堆

[scrapy.middleware] INFO: Enabled downloader middlewares:启动的下载中间价，默认一堆

[scrapy.middleware] INFO: Enabled spider middlewares:启动的爬虫中间件，默认一堆

[scrapy.middleware] INFO: Enabled item pipelines: 启动的管道

[scrapy.extensions.telnet] DEBUG: 爬虫运行的时候能够使用talent命令对爬虫做一些控制，比如暂停等

[scrapy.statscollectors] INFO: Dumping Scrapy stats:爬虫结束时候的一些统计信息，比如请求响应数量等

[scrapy.core.scraper] DEBUG: Scraped from <200 http://wz.sun0769.com/html/question/201707/340346.shtml>{'content':.....} :每次yield item的时候回提示item的内容以及这个item来自的url地址

Scrapy深入之scrapy shell

Scrapy shell是一个交互终端，我们可以在未启动spider的情况下尝试及调试代码，也可以用来测试XPath表达式

使用方法：

scrapy shell http://www.itcast.cn/channel/teacher.shtml

response.url：当前响应的url地址
response.request.url：当前响应对应的请求的url地址
response.headers：响应头
response.body：响应体，也就是html代码，默认是byte类型
response.requests.headers：当前响应的请求头

Scrapy深入之认识setting文件

为什么需要配置文件：

配置文件存放一些公共的变量（比如数据库的地址，账号密码等）

方便自己和别人修改

一般用全大写字母命名变量名　SQL_HOST = ‘192.168.0.1’

Scrapy深入之认识setting文件

在spider中，setting能够通过 self.settings的方式访问到:

```
class MySpider(scrapy.Spider):
    name = 'myspider'
    start_urls = ['http://example.com']

    def parse(self, response):
        print("Existing settings: %s" % self.settings.attributes.keys())
```

更重要的，我们需要知道如何在pipeline中使用setting所设置的内容:

```
class YangguangPipeline(object):
    def open_spider(self,spider):
        print(spider.settings.get("BOT_NAME",None))
```

那么：问题来了，什么是open_spider是干什么用的?

Scrapy深入之pipeline使用

```
import json
class JsonWriterPipeline(object):

    def open_spider(self, spider):    —>在爬虫开启的时候执行，仅执行一次
        self.file =open(spider.settings.get("SAVE_FILE", "./temp.json"), 'w')

    def close_spider(self, spider): —>在爬虫关闭的时候执行，仅执行一次
        self.file.close()

    def process_item(self, item, spider):
        line = json.dumps(dict(item)) + "\n"
        self.file.write(line)
        return item    —>不return的情况下，另个一个权重较低的pipeline就不会获取到该item
```

Mongodb回顾

```
from pymongo import MongoClient

class SnbookPipeline(object):
    def open_spider(self, spider):
        con = MongoClient(spider.settings.get("HOST"), spider.settings.get("PORT"))
                    ——>实例化一个mongoclient
        db = con[spider.settings.get("DB")]  ——>连接到数据库
        self.collection = db[spider.settings.get("COLLECTION")]  ->连接到集合

    def process_item(self, item, spider):
        self.collection.insert(dict(item))  ——>插入字典形式的item
```

mongodb shell命令回顾：

```
show dbs    ——>查看数据库
use books    ——>使用数据库
db.suning.find() ——>无条件查找
db.suning.find().pretty()  ——>查找并美化打印
db.suning.find().count()   ——>查找并统计
db.suning.remove({'catetory':'文学小说'})  ——>删除所有category为文学小说的数据
db.suning.drop()   ——>删除集合（表）
```

动手

需求：爬取苏宁易购所有下所有图书和图书分类信息，以及子链接页面的价格内容。

url : http://snbook.suning.com/web/trd-fl/999999/0.htm

目标：熟悉前面的知识点

Scrapy中CrawlSpider

回头看：

之前的代码中，我们有很大一部分时间在寻找下一页的url地址或者是内容的url地址上面，这个过程能更简单一些么？

思路：

1、从response中提取所有的a标签对应的url地址

2、自动的构造自己requests请求，发送给引擎

上面的功能可以做的更好：

满足某个条件的url地址，我们才发送给引擎，同时能够指定callback函数

Scrapy中CrawlSpider

需求：爬取csdn上面所有的博客专家及其文章的文章

Url地址：http://blog.csdn.net/experts.html

目标：通过csdn爬虫了解crawlspider的使用

生成crawlspider的命令：

scrapy genspider –t crawl csdn “csdn.cn”

Scrapy中CrawlSpider

```
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule


class CsdnspiderSpider(CrawlSpider):    —>继承自spiders的crawlspider类
    name = 'csdnspider'
    allowed_domains = ['blog.csdn.net']
    start_urls = ['http://blog.csdn.net/peoplelist.html?channelid=0&page=1']
                ——>第一次会请求的url，如果对这个url有特殊的需求，
                        可以定义一个parse_start_url函数专门处理这个url 所对应的响应

    rules = (
        Rule(LinkExtractor(allow=r'http://blog.csdn.net/\w+$'), follow=True),
                ——>能够找到所有作者的博客地址并且请求，
                    $符号加上表示已\w结尾，否则会匹配上\w+'/abc/def'等内容
        Rule(LinkExtractor(allow=r'peoplelist.html?channelid=\d+&page=\d+$'),follow=True),
                ——>找到所有的翻页地址并且请求，$符号同理
        Rule(LinkExtractor(allow=r'/article/details/\d+$'),callback="parse_article",follow=True),
                ——>找到所有的文章的url地址，并且请求，调用parase_article函数处理response
        Rule(LinkExtractor(allow=r'/article/list/\d+$'), follow=True)
                ——>找到所有的博客文章列表的翻页地址并且请求
    )
```

Scrapy中CrawlSpiders

注意点：

1、用命令创建一个crawlspider的模板：scrapy genspider –t crawl <爬虫名字> <all_domain>，也可以手动创建

2、CrawlSpider中不能再有以parse为名字的数据提取方法，这个方法被CrawlSpider用来实现基础url提取等功能

3、一个Rule对象接收很多参数，首先第一个是包含url规则的LinkExtractor对象，

常用的还有callback(制定满足规则的url的解析函数的字符串)和follow(response中提取的链接是否需要跟进)

4、不指定callback函数的请求下，如果follow为True，满足该rule的url还会继续被请求

5、如果多个Rule都满足某一个url，会从rules中选择第一个满足的进行操作

CrawlSpider补充(了解)

LinkExtractor更多常见参数：

allow：满足括号中“正则表达式”的URL会被提取，如果为空，则全部匹配。

deny：满足括号中“正则表达式”的URL一定不提取（优先级高于allow）。

allow_domains：会被提取的链接的domains。

deny_domains：一定不会被提取链接的domains。

restrict_xpaths：使用xpath表达式，和allow共同作用过滤链接，级xpath满足范围内的url地址会被提取

spiders.Rule常见参数：

link_extractor：是一个Link Extractor对象，用于定义需要提取的链接。

callback： 从link_extractor中每获取到链接时，参数所指定的值作为回调函数

follow：是一个布尔(boolean)值，指定了根据该规则从response提取的链接是否需要跟进。

如果callback为None，follow 默认设置为True ，否则默认为False。

process_links：指定该spider中哪个的函数将会被调用，从link_extractor中获取到链接列表时将会调用该函数，该方法主要用来过滤url。

process_request：指定该spider中哪个的函数将会被调用， 该规则提取到每个request时都会调用该函数，用来过滤request

下载中间件

使用方法：

编写一个Downloader Middlewares和我们编写一个pipeline一样，定义一个类，然后在setting中开启

Downloader Middlewares默认的方法：

process_request(self, request, spider)：

当每个request通过下载中间件时，该方法被调用。

process_response(self, request, response, spider)：

当下载器完成http请求，传递响应给引擎的时候调用

下载中间件

```
class RandomUserAgent(object):
    def process_request(self, request, spider):
        useragent = random.choice(USER_AGENTS)
        request.headers["User-Agent"] =  useragent
```

——>添加自定义的UA，给request的headers赋值即可

```
class ProxyMiddleware(object)
    def process_request(selfs,request,spider):
        reqeust.meta["proxy"] = "http://124.115.126.76:808"
```

——>添加代理，需要在request的meta信息中添加proxy字段

——>代理的形式为：协议+ip地址+端口

scrapy模拟登陆

为什么需要模拟登陆?

获取cookie，能够爬取登陆后的页面

回顾:

requests是如何模拟登陆的?

1、直接携带cookies请求页面

2、找接口发送post请求存储cookie

selenium是如何模拟登陆的?

找到对应的input标签，输入文字点击登录

scrapy模拟登陆

那么对于scrapy来说，也是有两个方法模拟登陆：

1、直接携带cookie

2、找到发送post请求的url地址，带上信息，发送请求

scrapy模拟登陆之携带cookie

应用场景：

1、cookie过期时间很长，常见于一些不规范的网站

2、能在cookie过期之前把搜有的数据拿到

3、配合其他程序使用，比如其使用selenium把登陆之后的cookie获取到保存到本地，scrapy发送请求之前先读取本地cookie

携带cookie登录之前

我们在spider下面定义了start_urls,那么这个start_urls是交给谁去处理的?

scrapy.Spider部分源码:

```
from scrapy.http import Request

def start_requests(self):
    cls = self.__class__
    if method_is_overridden(cls, Spider, 'make_requests_from_url'):
        warnings.warn(
            "Spider.make_requests_from_url method is deprecated; it "
            "won't be called in future Scrapy releases. Please "
            "override Spider.start_requests method instead (see %s.%s)." % (
                cls.__module__, cls.__name__
            ),
        )
        for url in self.start_urls:
            yield self.make_requests_from_url(url)
    else:
        for url in self.start_urls:
            yield Request(url, dont_filter=True)

def make_requests_from_url(self, url):
    """ This method is deprecated. """
    return Request(url, dont_filter=True)
```

我们定义在spider下的start_urls=[]都是默认交给start_requests处理的
所以如果必要，我们可以重写start_requests方法

scrapy模拟登陆之携带cookie

```
class RenrenspiderSpider(scrapy.Spider):
    name = 'renrenspider'
    allowed_domains = ['renren.com']
    cookies = dict(    ——>自定义的cookies，字典形式
        anonymid="j3jxk555-nrn0wh",
        ....
        wp_fold=0
    )

    def start_requests(self):  ——>重写start_requests函数，指定start_urls的处理方式
        start_urls = 'http://www.renren.com/'
        yield scrapy.Request(start_urls, callback=self.parse, cookies=self.cookies)
                                ——>指定callback函数，同时携带cookie

    def parse(self, response):
        print(re.findall(r'毛**', response.body.decode()))
        my_profile_url = "http://www.renren.com/327550029/profile"
        yield scrapy.Request(my_profile_url,callback=self.parse_profile)

    def parse_profile(self,response):
        print('*' * 100)
        print(re.findall(r'毛**', response.body.decode()))
```

scrapy模拟登陆之携带cookie

那么问题来了：如何知道我的cookie确实是在不同的解析函数中传递呢？

```
# Disable cookies (enabled by default)
#COOKIES_ENABLED = False  —>cookie在setting中默认是开启的
COOKIES_DEBUG = True    —>在settings中添加该参数即可
    ——>cookie能够在不同的解析函数中传递，前提也是COOKIES_ENABLED为True
```

终端效果如下：

```
[scrapy.downloadermiddlewares.cookies] DEBUG: Sending cookies to:
<GET http://zhibo.renren.com/top>
Cookie: anonymid=j3jxk555-nrn0wh; _r01_=1; wp=0; ......
```

scrapy模拟登陆之发送post请求

```
class GithubSpider(scrapy.Spider):
    name = 'github'
    allowed_domains = ['github.com']
    start_urls = ['https://github.com/login']    —>请求首页，为了获取登录参数
    headers = {
        "Accept": "*/*",
        "Accept-Language": "en-US,en;q=0.8,zh-TW;q=0.6,zh;q=0.4",
        ……
    }

    def parse(self, response):
        print(response.url)
        utf8 = response.xpath("//form[@action='/session']/div[1]/input[1]/@value").extract_first()
        authenticity_token = response.xpath("//form[@action='/session']/div[1]/input[2]/@value").extract_first()
        return scrapy.FormRequest("https://github.com/session",
                                  headers=self.headers,  —>可以在spider中定义，也可以在setting中定义
                                  formdata=dict(
                                      commit="Sign in",
                                      utf8=utf8,
                                      authenticity_token=authenticity_token,
                                      login="user_name",
                                      password="password"
                                  ),
                                  callback=self.after_login  —>登录之后的回调函数
                                  )

    def after_login(self, response):
        print(response.url, "*" * 100, response.status)
```

使用scrapy.Request的时候一直是发送的get请求，那么post请求就是需要使用scrapy.FormRequest来发送，同时使用formdata来携带需要post的数据

注意：github的部分页面只允许一处登录，比如https://github.com/settings/security

scrapy模拟登陆之自动登录

```
class RenrenspiderSpider(scrapy.Spider):
    name = 'renrenspider'
    allowed_domains = ["renren.com"]
    start_urls = [
        "http://www.renren.com/",
    ]

    def parse(self, response):
        yield scrapy.FormRequest.from_response(
            response,  ——>自动从该响应中找到form表单进行登录
            formdata={"email": "user_name", "password": "password"},
            callback=self.parse_page
        )

    def parse_page(self, response):
        print(response.url, "*" * 100, response.status)
        print('*' * 100)
        print(re.findall(r'user_name', response.body.decode()))
                              ——>response.body是bytes类型，需要decode为str
```

from_response的意思就是从响应中找到from表单进行登录

Fiddler手机抓包

Thank You!

改变中国 IT 教育，我们正在行动