

目录

1. 爬虫框架 Scrapy 简介
 - 1.1 Scrapy 项目文件结构
 - 1.2 Scrapy 工作原理
 - 1.2.1 部件组成
 - 1.2.2 数据流动过程
2. 使用教程
 - 2.1 Quick Start
 - 2.2 配置文件结构
3. 通用爬虫构建
 - 3.1 项目文档结构
 - 3.2 主要文件说明
 - 3.3 系统工作流程
4. 用户使用指南
 - 4.1 正则表达式
 - 4.2 xpath 提取方法
5. 其他问题

1. 爬虫框架 Scrapy 简介

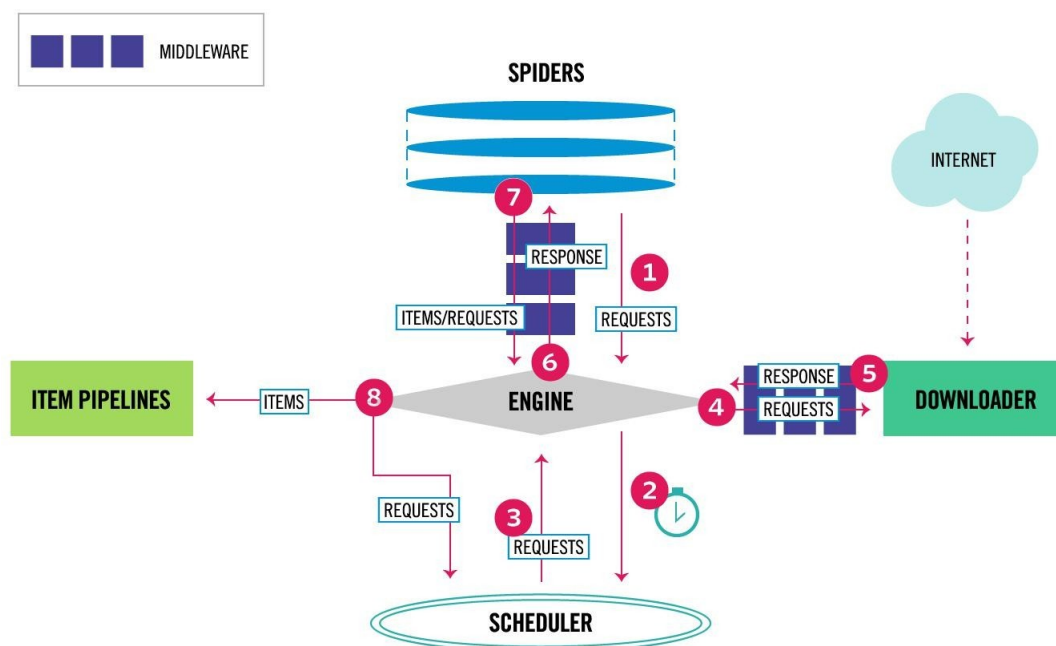
Scrapy 是 Python 开发的一个快速、高层次的 web 抓取框架,用于抓取 web 站点并从页面中提取结构化的数据。Scrapy 用途广泛,可以用于数据挖掘、监测和自动化测试。Scrapy 另一大优势在于它是一个开源框架,任何人都可以根据需求方便的修改。

1.2 Scrapy 总体结构

当我们创建一个新的 Scrapy 项目后,它的文件结构类似于下面的形式:

```
scrapy.cfg
myproject/
  __init__.py
  items.py
  middlewares.py
  pipelines.py
  settings.py
  spiders/
    __init__.py
    spider1.py
    spider2.py
```

1.2 Scrapy 工作原理



1.2.1 部件组成:

- (1) Scrapy Engine: Scrapy 的核心部件是爬虫引擎,它控制爬虫工作时所有部件之间的数据流动,以及当某些行为发生时触发事件;
- (2) Scheduler: 调度器接收来自引擎的请求,并将它们排入队列,以便在引擎请求它们时将它们提供给引擎;

- (3) Downloader:下载器接收来自引擎的请求,并通过引擎返回网页到 Spider 中用于处理;
- (4) Spider:它是 Scrapy 用户可以自行定制化的类,用于解析响应以及提取数据项 items,也可以提取出请求通过引擎放入 Scheduler 的队列当中用于爬取;编写爬虫的大部分工作都是在这里完成;
- (5) Item Pipeline:用于处理 Spider 中提取的 items。主要任务诸如数据清理,数据验证以及数据持久化(比如存储到磁盘或者数据库当中);
- (6) Downloader Middlewares:它是位于引擎和下载器中间位置的特定的钩子(hooks),用于处理引擎向下载器发送的请求,以及下载器向引擎返回的响应;
- (7) Spider Middlewares:它是位于引擎和 Spider 中间位置的特定的钩子(hooks),用于处理爬虫输入(response)和输出(items&requests);

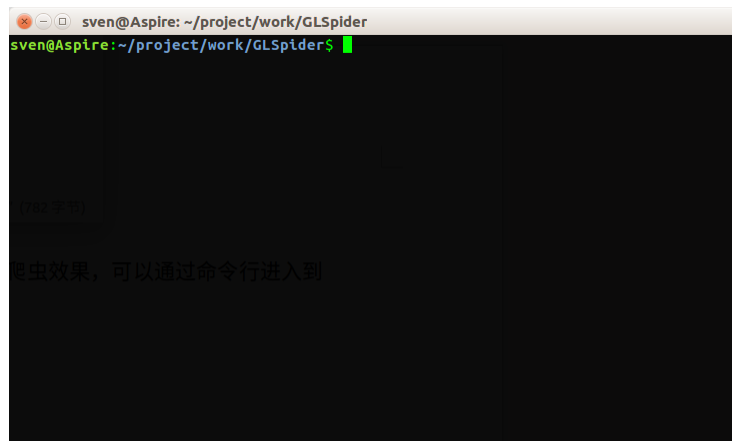
1.2.2 数据流动过程

- (1) Engine 从 Spider 中获取初始 request;
- (2) Engine 调度请求到 Scheduler 中,并继续寻求下一个 request;
- (3) Scheduler 返回下一个 request 给 Engine;
- (4) Engine 通过 download middleware 发送请求到 Downloader;
- (5) 一旦请求页下载完成,Downloader 还要通过 download middleware 将响应 response(以及网页)发送给 Engine;
- (6) Engine 从 Downloader 接收到响应之后,通过 spider middleware 将响应发送给 Spider 用于页面解析处理;
- (7) Spider 处理响应后,通过 spider middleware 将抽取的 items 以及新的 request (可能存在,用于 Engine 的调度)发送给 Engine;
- (8) Engine 发送处理过的 items 给 item pipeline,发送处理过的 request 给 Scheduler 寻求下一个 request;
- (9) 重复步骤(3)-(8),直至调度器中没有任何 request 存在为止。

2. 使用教程

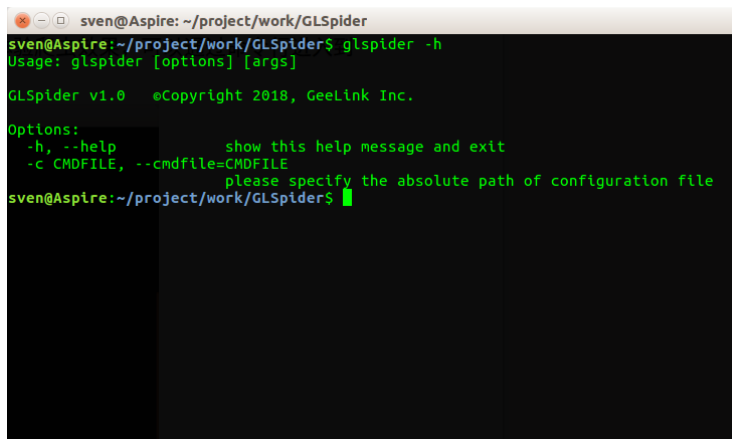
2.1 Quick Start

(1) 假定计算机或远程服务器已经配置 GLSpider 项目，如果你想快速体验爬虫效果，可以通过命令行进入到 GLSpider 项目目录下，例如下图所示。



```
sven@Aspire: ~/project/work/GLSpider
sven@Aspire:~/project/work/GLSpider$
```

(2)通过 glspider 命令查看相关操作。

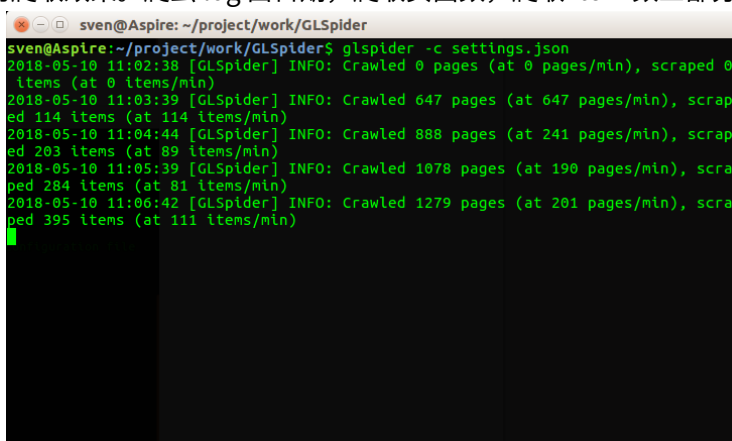


```
sven@Aspire: ~/project/work/GLSpider
sven@Aspire:~/project/work/GLSpider$ glspider -h
Usage: glspider [options] [args]

GLSpider v1.0   ©Copyright 2018, GeeLink Inc.

Options:
  -h, --help          show this help message and exit
  -c CMDFILE, --cmdfile=CMDFILE
                      please specify the absolute path of configuration file
sven@Aspire:~/project/work/GLSpider$
```

(3)通过键入” glspider -c settings.json”快速查看系统内置的配置文件 settings.json 对目标网站” www.epet.com”的爬取效果。爬虫 log 由日期，爬取页面数，爬取 item 数三部分组成。



```
sven@Aspire:~/project/work/GLSpider$ glspider -c settings.json
2018-05-10 11:02:38 [GLSpider] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)
2018-05-10 11:03:39 [GLSpider] INFO: Crawled 647 pages (at 647 pages/min), scraped 114 items (at 114 items/min)
2018-05-10 11:04:44 [GLSpider] INFO: Crawled 888 pages (at 241 pages/min), scraped 203 items (at 89 items/min)
2018-05-10 11:05:39 [GLSpider] INFO: Crawled 1078 pages (at 190 pages/min), scraped 284 items (at 81 items/min)
2018-05-10 11:06:42 [GLSpider] INFO: Crawled 1279 pages (at 201 pages/min), scraped 395 items (at 111 items/min)
```

(4)可以使用快捷键” Ctrl+C”停止爬虫工作，这时可以发现同一路径下多了一个 data 文件夹，这是系统默认的获取的数据的存储路径，之后用户编写的配置文件中可以自行指定 data 文件夹的绝对路径。在 ubuntu 下通过 tree 查看 data 目录结构，每一次重新运行爬虫后都会会在 data 目录下生成一个以类时间戳命名的数据存储目录，子目录 bdf 和 html 分别存储 bdf 文件和 html 网页。

```
sven@Aspire: ~/project/work/GLSpider/data
sven@Aspire:~/project/work/GLSpider/data$ tree
.
├── 20180510_110238
│   ├── bdf
│   │   └── epet.json
│   └── html
│       ├── item.epet.com-112721.html
│       ├── item.epet.com-112748.html
│       ├── item.epet.com-112841.html
│       ├── item.epet.com-112845.html
│       ├── item.epet.com-114016.html
│       ├── item.epet.com-114062.html
│       ├── item.epet.com-114876.html
│       └── item.epet.com-115463.html
└── 3 directories, 9 files
sven@Aspire:~/project/work/GLSpider/data$
```

了一个 data 文件夹，这是系统默认的

2.2 配置文件结构

目前为止，用户还无法自行编写配置文件进行定制化的爬取。下面图示用来说明用户需要配置哪些参数。

```
settings.json (~/.project/work/GLSpider) - VIM
"START_URL": "http://www.epet.com",
"ALLOWED_DOMAINS": "epet.com",

"INCLUDE": ".*item.*.html$",
"EXCLUDE": ".*product.*",

"COLLECTION": "",
"FILENAME": "epet",

"EXTRACTPATTERN": {
  "selOnePt": "ptmatch1",
  "ptmatch1": {
    "field": ".*item.*.html$",
    "title": "//body/div[@class='w-max ct mb20']/div[2]/div[2]/h1[@id='abcde']/text()",
    "subtitle": "//body/div[@class='w-max ct mb20']/div[2]/div[2]/div[@class='ft14 mt c93c']/text()",
    "price": "//body/div[@class='w-max ct mb20']/div[2]/div[2]/div[2]/span[@id='goods-sale-price']/text()"
  },
  "ptmatch2": {
    "field": ".*comment.*.html#$",
    "title": "/html/body/div[3]/div[3]/a[2]/text()",
    "score": "/*[@id='reply_display']/ul/li[1]/center[1]/text()"
  }
}
```

(1) START_URL

类型：字符串

说明：表示起始网址，即爬虫程序开始的起点。这个字段是**必填项**。

(2) ALLOWED_DOMAINS

类型：字符串

说明：允许爬取的网站域。这个字段是**必填项**，允许域应该与起始网址所在的站点域一致。

(3) INCLUDE

类型：字符串

说明：允许爬取的网址字段。这个字段是**可选项**，默认为“*”，即匹配所有字符，内容为正则表达式形式，填写之前需要用户对目标网站 url 结构有所了解，否则获取不到需要的网页内容。

(4) EXCLUDE

类型：字符串

说明：禁止爬取的网址字段。这个字段是**可选项**，默认为空字符串“”，内容为正则表达式形式，填写之前需要用户对目标网站 url 结构有所了解，作用是可以过滤掉无关网页。

(5)**COLLECTION**

类型：字符串

说明：保存数据到磁盘的绝对路径。这个字段是**可选项**，默认为项目根目录。选定路径进行爬虫工作后，会自动在所选目录下生成 data 文件夹（如果不存在）用于存储获取的数据。

(6)**FILENAME**

类型：字符串

说明：用于指定存储 BDF 数据格式的 JSON 文件名。这个字段是**可选项**，默认文件名为“bdf”。

(7)**EXTRACTPATTERN**

类型：python 字典格式

说明：用于对指定字段的网页内容进行页面解析，获取需要的数据。这个字段是**必填项**。如样例 settings.json 中所示，子字段“ptmatch1”、“ptmatch2”表示对应其中的子字段“field”的提取规则，“field”是对网页进行过滤的正则表达式。利用子字段“selOnept”选择其中一个字段进行网页解析，抽取网页内容，用户可以准备多组规则项“ptmatch1,2,...”，每次爬取时可以随时选择所需字段。关于提取规则 xpath 的使用，参见 4.1 节。

3 通用爬虫构建

3.1 项目文档结构

根目录为/GLSpider

GLSpider/

scrapy.cfg

GLSpider/

items.py

middlewares.py

pipelines.py

settings.py

spiders/

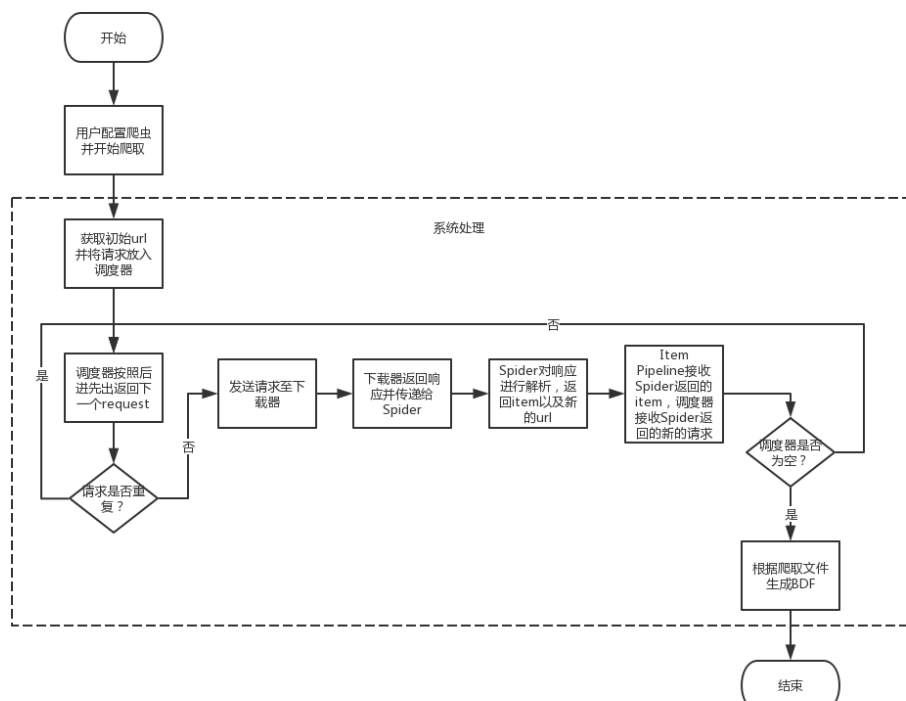
__init__.py

GLSpider.py

3.2 主要文件说明

/GLSpider/settings.py	/GLSpider/items.py	/GLSpider/pipelines.py	/GLSpider/spiders/GLSpider.py
项目配置文件，如实现通用爬虫的起始网址和终止条件，商品页规则，输出文件编码格式等等一些设置	爬虫数据清洗，编写构造 item 的类，设置网页保存一些 Filed 用于 item 的定义，例如 url,title 和 html 三个域	实现 item 的持久化，类 fileWriterPipeline 用于保存 item 的 html 域(即 html.body, 网页内容)以及 item 项 (json 格式的 BDF 文件) 到本地磁盘	通过导入配置文件中的商品页提取规则对响应 response 进行解析，获取指定的数据，返回 item

3.3 系统工作流程



4 用户使用指南

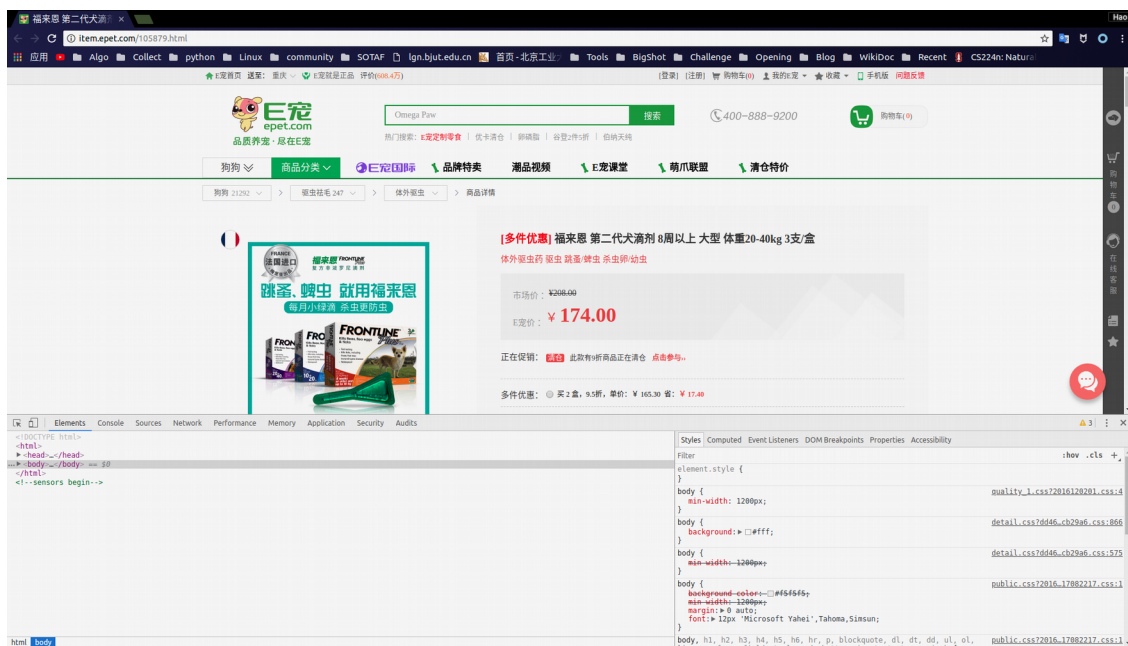
用户在使用项目的时候,主要工作是在配置文件中`进行参数设置`,以及网页的抽取规则的编写。

4.1 正则表达式

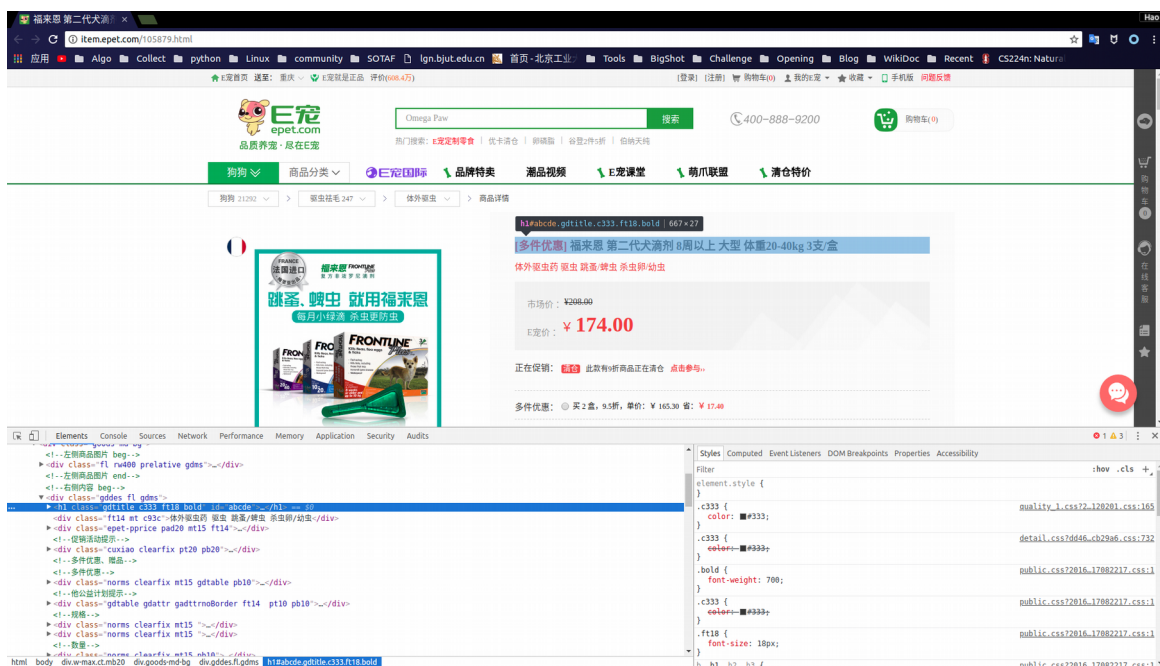
首先需要注意的就是提取网页字段的正则表达式的编写，这对用户的要求是对网站 url 字段的充分掌握以及对正则表达式的熟悉。如果对这两方面不熟悉，可能会很大影响爬虫的工作效果。

4.2 xpath 提取方法

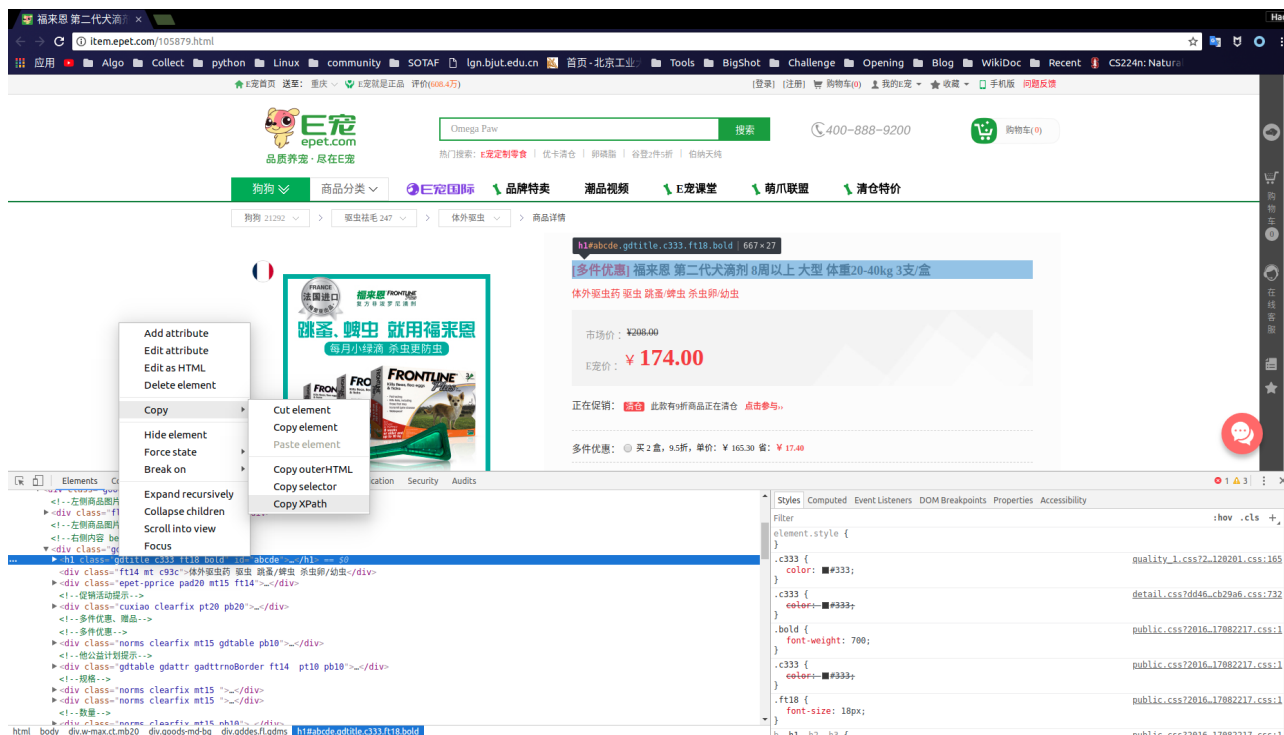
(1) xpath 的提取需要用到浏览器。以谷歌浏览器为例，打开目标网页，鼠标右击空白处，点击检查，打开开发者调试界面，如下图所示。



(2) 点击调试界面左上角鼠标图案，在网页中选择需要提取的字段。比如提取商品标题，如下图所示。



(3) 然后在选择到的标签处右击，选择复制→复制 xpath。如下图所示。



(4) 这时就得到了所需字段对应的 xpath，上面得到的 xpath 为 “//*[@id="abcde"]”。

这里有两点需要**着重注意**：

- 1) 通过复制得到的 xpath 是网页字段对应的标签内容，而我们所需要的是标签当中的内容，所以，我们在填写配置文件当中的提取规则时，需要对复制的 xpath 稍加更改，比如上面的 xpath 更改为” `//*[@id="abcde"]//text()`”，这样才得到标签当中的内容。所以如果需要获取这一字段，在配置文件中填写的完整形式为：`"title": "//*[@id="abcde"]//text()`”。这里的” title”需要用户自行定义。
- 2) 复制得到的 xpath 中如果存在双引号，则更改为单引号，防止出现格式错误。如上面的” `//*[@id='abcde']//text()`”。

(5) 还是以上面网页为例，再提取另一个 price 字段，得到 xpath 为：” `//*[@id='goods-sale-price']//text()`”。以提取 title 和 price 两个字段为例，构造配置文件当中的” EXTRACTPATTERN”域如下图所示。

```
"EXTRACTPATTERN": {
  "selOnePt": "ptmatch1",
  "ptmatch1": {
    "field": ".item.*.html$",
    "title": "//*[@id='abcde']//text()",
    "price": "//*[@id='goods-sale-price']//text()"
  }
}
```

注意” ptmatch1”当中的” field”域提供的是用于目标页面匹配的正则表达式，然后利用下面的网页提取字段” title”和” price”当中的 xpath 表达式进行标题和价格的提取。网页提取字段用户可自行添加，以及与” ptmatch1”同类的域也可以自行添加，用于爬虫的选择。

5. 其他问题

因为 scrapy 框架比较复杂,查阅相关博客,有人提到爬虫最为主要的数据结构“待爬队列”以及调度方式在官方文档中没有详细的介绍,也就是说,即使完全看完官方文档,也未必完全掌握这个框架,所以想要完全的控制爬取策略,就需要在待爬队列上进行操作,这个时候就需要在 scrapy 的源代码的研读基础上去定制化更复杂的功能,比如更换系统自带的调度器。目前因为时间问题,所以实现的功能是有限的,只能在日后不断优化完善。

另外,由于这次实现的是一个通用爬虫,所以具体到某一个目标网站的爬取的时候,爬取规则是需要用户自行指定的,以及另外一些在网页源代码当中获取不到的数据,比如 epet 商品页的”参数&公告”一栏是通过 js 异步加载的,如果需要获取这类信息,必然需要用户具备相应的 web 知识。当然,这不是此项目的设计初衷,所以提高人机交互能力也是 GLSpider 项目不断改进的方面之一。