在前面我们已经学习了多进程、requests、正则表达式、pyquery、PyMongo 等的基本用法，但我们还没有完整地实现一个爬取案例。本课时，我们就来实现一个完整的网站爬虫案例，把前面学习的知识点串联起来，同时加深对这些知识点的理解。

准备工作

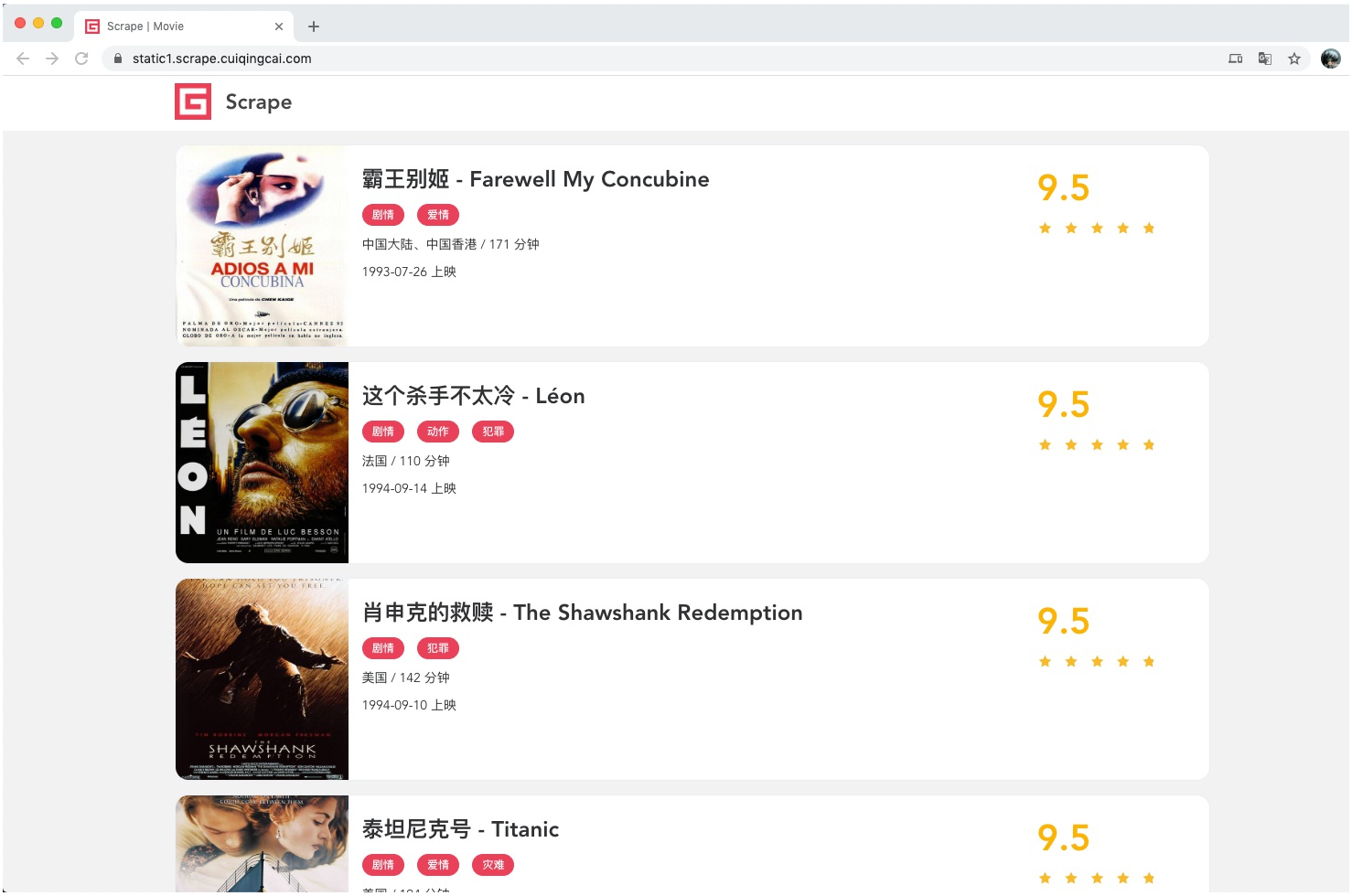
在本节课开始之前，我们需要做好如下的准备工作：

- 安装好 Python3（最低为 3.6 版本），并能成功运行 Python3 程序。
- 了解 Python 多进程的基本原理。
- 了解 Python HTTP 请求库 requests 的基本用法。
- 了解正则表达式的用法和 Python 中正则表达式库 re 的基本用法。
- 了解 Python HTML 解析库 pyquery 的基本用法。
- 了解 MongoDB 并安装和启动 MongoDB 服务。
- 了解 Python 的 MongoDB 操作库 PyMongo 的基本用法。

以上内容在前面的课时中均有讲解，如果你还没有准备好，那么我建议你可以再复习一下这些内容。

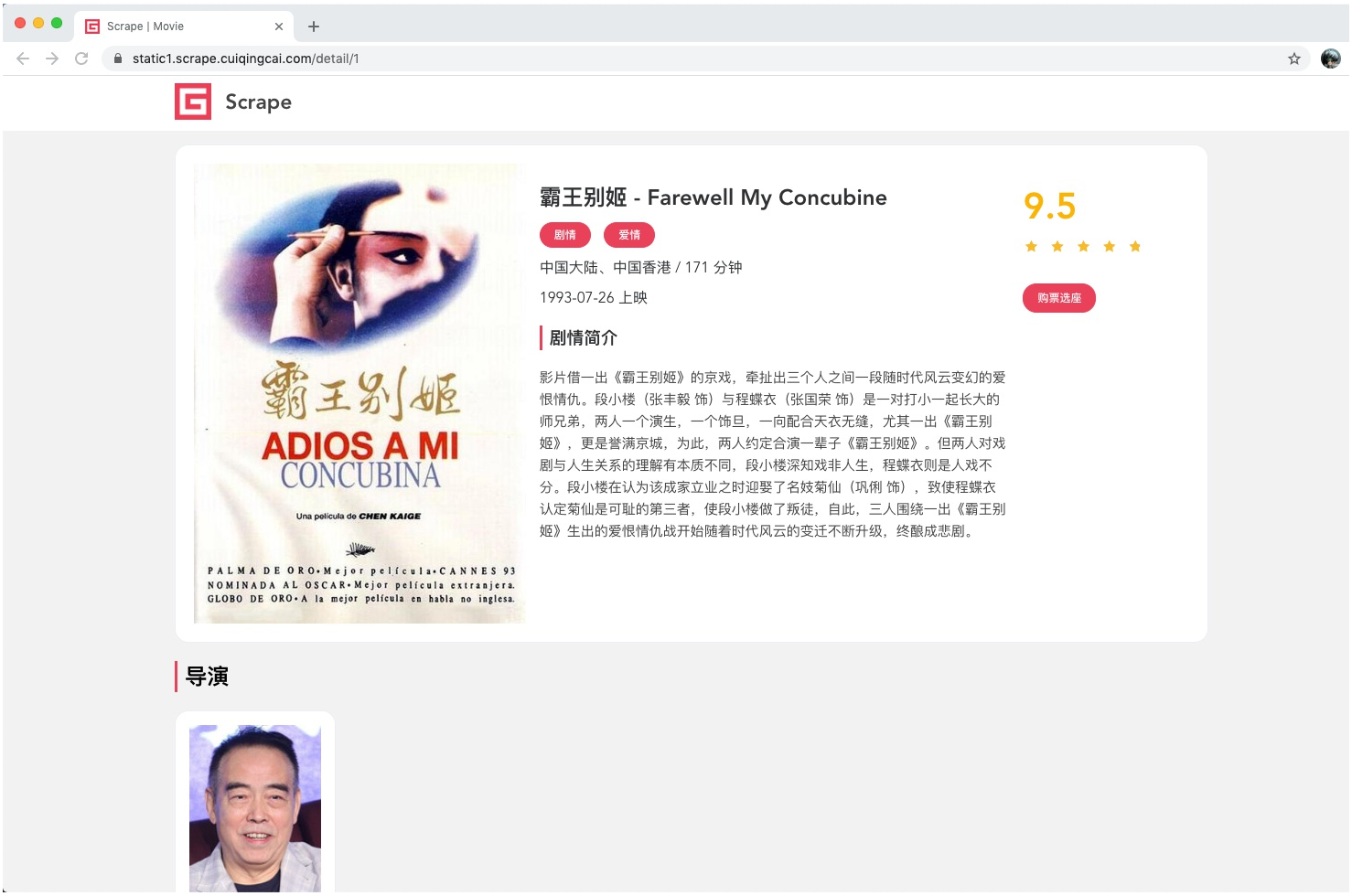
爬取目标

这节课我们以一个基本的静态网站作为案例进行爬取，需要爬取的链接为：https://static1.scrape.cuiqingcai.com/，这个网站里面包含了一些电影信息，界面如下：



首页是一个影片列表，每栏里都包含了这部电影的封面、名称、分类、上映时间、评分等内容，同时列表页还支持翻页，点击相应的页码我们就能进入到对应的新列表页。

如果我们点开其中一部电影，会进入电影的详情页面，比如我们点开第一部《霸王别姬》，会得到如下页面：



这里显示的内容更加丰富、包括剧情简介、导演、演员等信息。

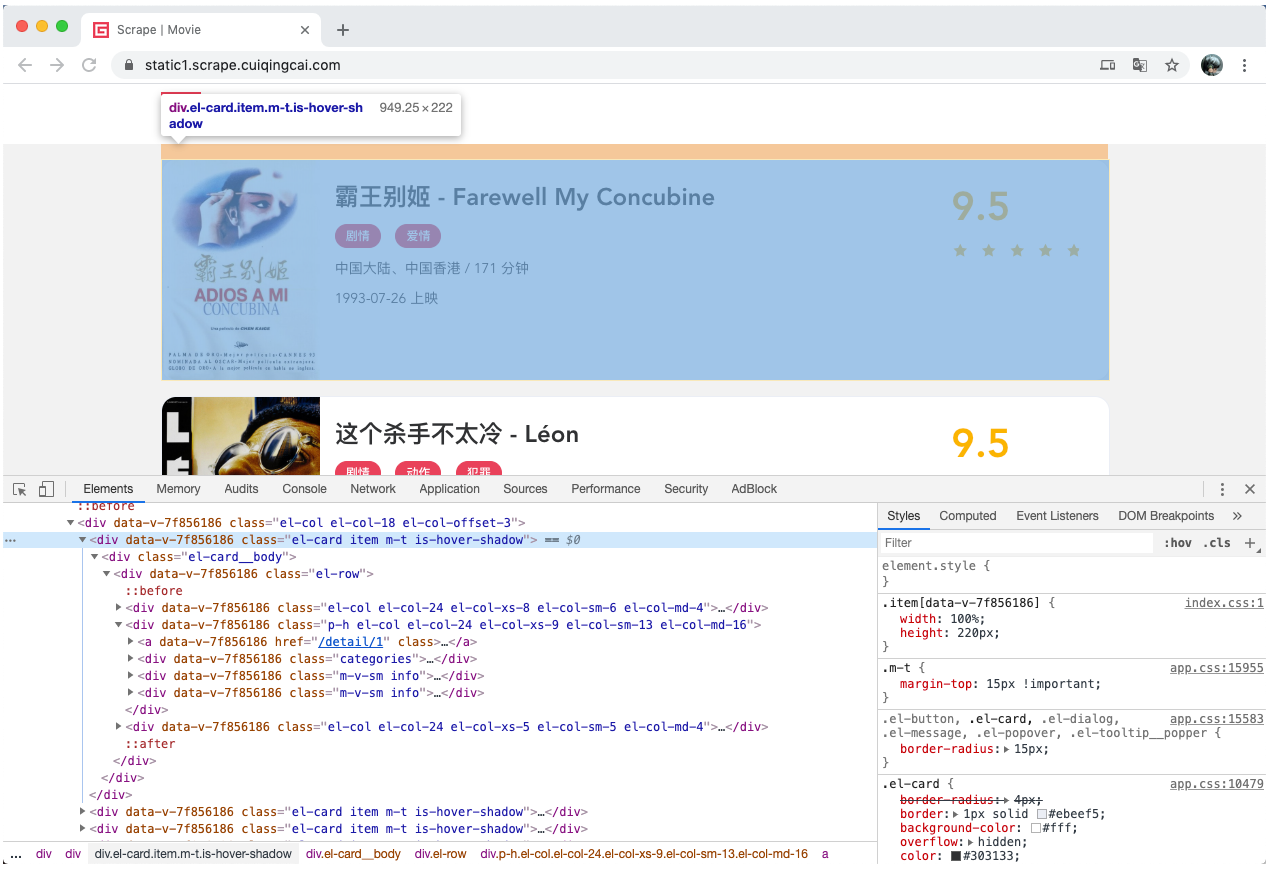
我们这节课要完成的目标是：

- 用 requests 爬取这个站点每一页的电影列表，顺着列表再爬取每个电影的详情页。
- 用 pyquery 和正则表达式提取每部电影的名称、封面、类别、上映时间、评分、剧情简介等内容。
- 把以上爬取的内容存入 MongoDB 数据库。
- 使用多进程实现爬取的加速。

那么我们现在就开始吧。

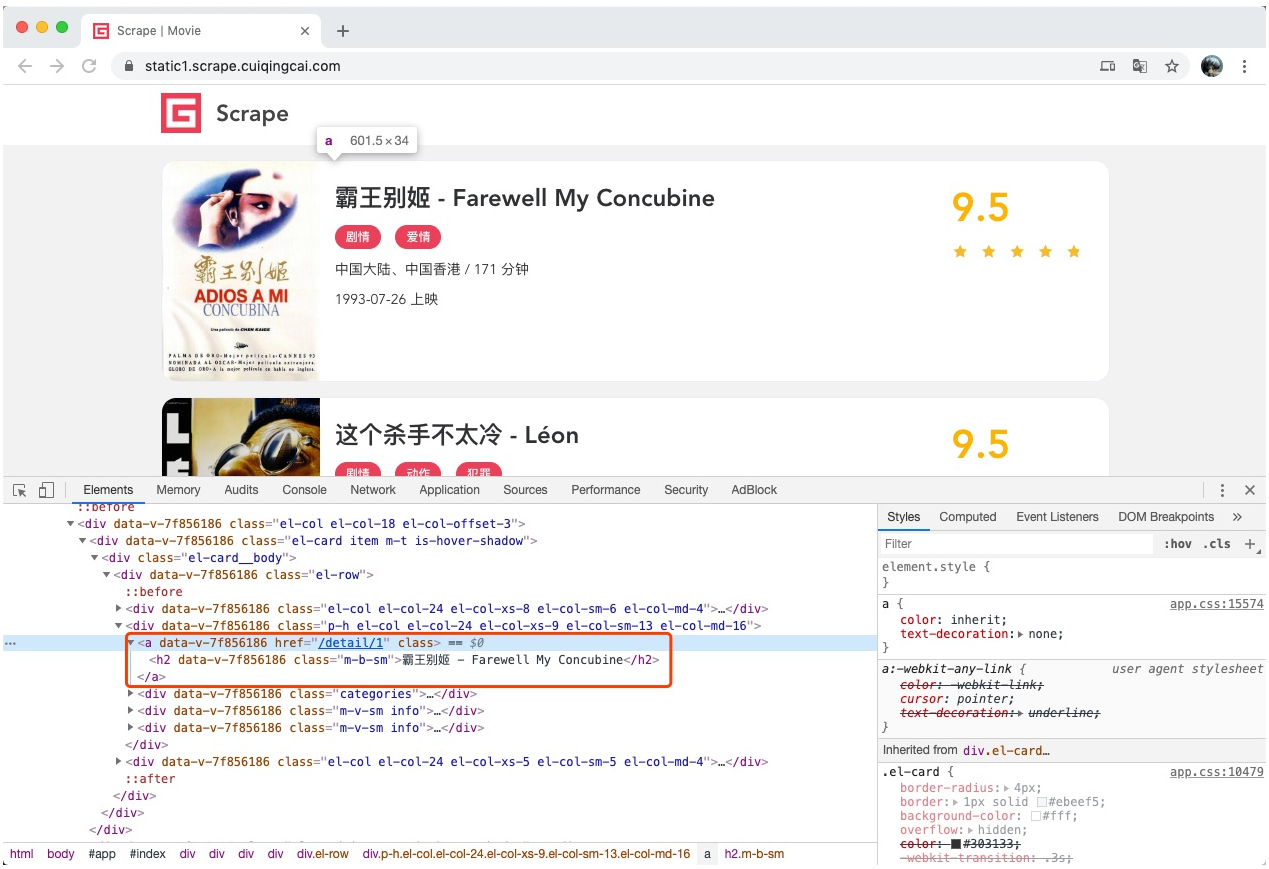
爬取列表页

爬取的第一步肯定要从列表页入手，我们首先观察一下列表页的结构和翻页规则。在浏览器中访问 https://static1.scrape.cuiqingcai.com/，然后打开浏览器开发者工具，观察每一个电影信息区块对应的 HTML，以及进入到详情页的 URL 是怎样的，如图所示：



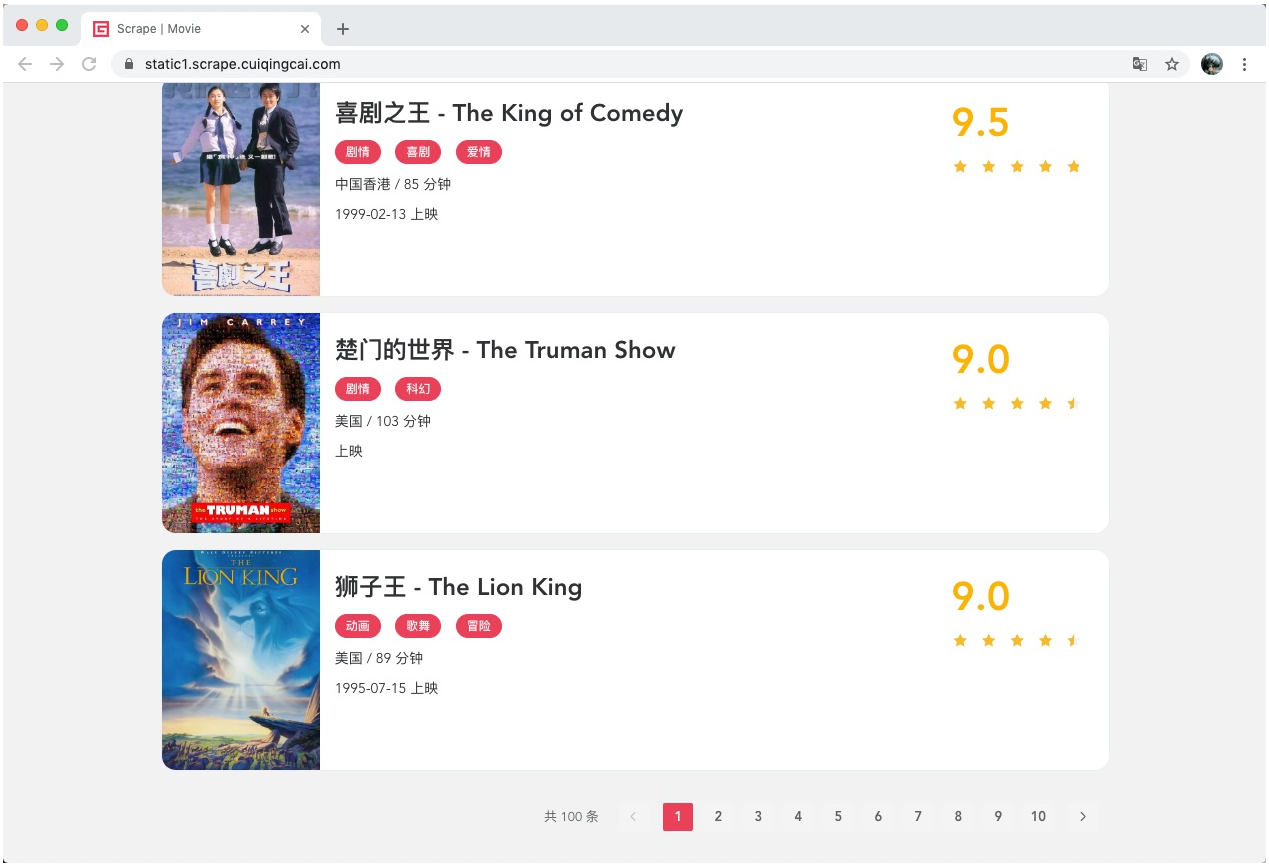
可以看到每部电影对应的区块都是一个 div 节点，它的 class 属性都有 el-card 这个值。每个列表页有 10 个这样的 div 节点，也就对应着 10 部电影的信息。

我们再分析下从列表页是怎么进入到详情页的，我们选中电影的名称，看下结果：

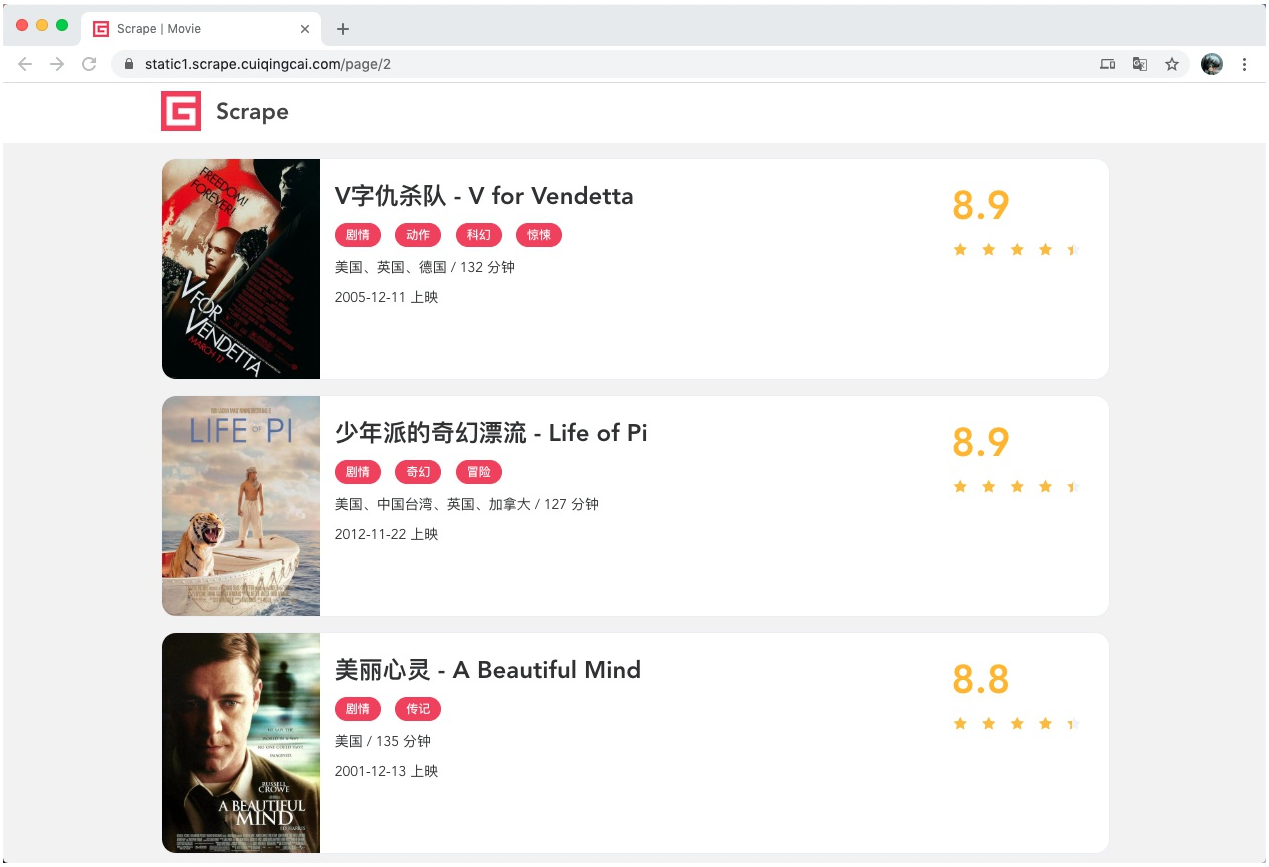


可以看到这个名称实际上是一个 h2 节点，其内部的文字就是电影的标题。h2 节点的外面包含了一个 a 节点，这个 a 节点带有 href 属性，这就是一个超链接，其中 href 的值为 /detail/1，这是一个相对网站的根 URL https://static1.scrape.cuiqingcai.com/ 路径，加上网站的根 URL 就构成了 https://static1.scrape.cuiqingcai.com/detail/1，也就是这部电影详情页的 URL。这样我们只需要提取这个 href 属性就能构造出详情页的 URL 并接着爬取了。

接下来我们来分析下翻页的逻辑，我们拉到页面的最下方，可以看到分页页码，如图所示：



页面显示一共有 100 条数据，10 页的内容，因此页码最多是 10。接着我们点击第 2 页，如图所示：



可以看到网页的 URL 变成了 https://static1.scrape.cuiqingcai.com/page/2，相比根 URL 多了 /page/2 这部分内容。网页的结构还是和原来一模一样，所以我们可以和第 1 页一样处理。

接着我们查看第 3 页、第 4 页等内容，可以发现有这么一个规律，每一页的 URL 最后分别变成了 /page/3、/page/4。所以，/page 后面跟的就是列表页的页码，当然第 1 页也是一样，我们在根 URL 后面加上 /page/1 也是能访问的，只不过网站做了一下处理，默认的页码是 1，所以显示第 1 页的内容。

好，分析到这里，逻辑基本就清晰了。

如果我们要完成列表页的爬取，可以这么实现：

- 遍历页码构造 10 页的索引页 URL。
- 从每个索引页分析提取出每个电影的详情页 URL。

现在我们写代码来实现一下吧。

首先，我们需要先定义一些基础的变量，并引入一些必要的库，写法如下：

```
import requests
import logging
import re
import pymongo
from pyquery import PyQuery as pq
from urllib.parse import urljoin

logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s: %(message)s')

BASE_URL = 'https://static1.scrape.cuiqingcai.com'
TOTAL_PAGE = 10
```

这里我们引入了 requests 用来爬取页面，logging 用来输出信息，re 用来实现正则表达式解析，pyquery 用来直接解析网页，pymongo 用来实现 MongoDB 存储，urljoin 用来做 URL 的拼接。

接着我们定义日志输出级别和输出格式，完成之后再定义 BASE_URL 为当前站点的根 URL，TOTAL_PAGE 为需要爬取的总页码数量。

定义好了之后，我们来实现一个页面爬取的方法吧，实现如下：

```
def scrape_page(url):
    logging.info('scraping %s...', url)
    try:
        response = requests.get(url)
        if response.status_code == 200:
            return response.text
        logging.error('get invalid status code %s while scraping %s', response.status_code, url)
    except requests.RequestException:
        logging.error('error occurred while scraping %s', url, exc_info=True)
```

考虑到我们不仅要爬取列表页，还要爬取详情页，所以在这里我们定义一个较通用的爬取页面的方法，叫作 scrape_page，它接收一个 url 参数，返回页面的 html 代码。

这里我们首先判断状态码是不是 200，如果是，则直接返回页面的 HTML 代码，如果不是，则会输出错误日志信息。另外，这里实现了 requests 的异常处理，如果出现了爬取异常，则会输出对应的错误日志信息。这时我们将 logging 的 error 方法的 exc_info 参数设置为 True 则可以打印出 Traceback 错误堆栈信息。

好了，有了 scrape_page 方法之后，我们给这个方法传入一个 url，正常情况下它就可以返回页面的 HTML 代码了。

在这个基础上，我们来定义列表页的爬取方法吧，实现如下：

```
def scrape_index(page):
    index_url = f'{BASE_URL}/page/{page}'
    return scrape_page(index_url)
```

方法名称叫作 scrape_index，这个方法会接收一个 page 参数，即列表页的页码，我们在方法里面实现列表页的 URL 拼接，然后调用 scrape_page 方法爬取即可得到列表页的 HTML 代码了。

获取了 HTML 代码后，下一步就是解析列表页，并得到每部电影的详情页的 URL 了，实现如下：

```
def parse_index(html):
    doc = pq(html)
    links = doc('.el-card .name')
    for link in links.items():
        href = link.attr('href')
        detail_url = urljoin(BASE_URL, href)
        logging.info('get detail url %s', detail_url)
        yield detail_url
```

在这里我们定义了 parse_index 方法，它接收一个 html 参数，即列表页的 HTML 代码。接着我们用 pyquery 新建一个 PyQuery 对象，完成之后再用 .el-card .name 选择器选出来每个电影名称对应的超链接节点。我们遍历这些节点，通过调用 attr 方法并传入 href 获得详情页的 URL 路径，得到的 href 就是我们在上文所说的类似 /detail/1 这样的结果。由于这并不是一个完整的 URL，所以我们需要借助 urljoin 方法把 BASE_URL 和 href 拼接起来，获得详情页的完整 URL，得到的结果就是类似 https://static1.scrape.cuiqingcai.com/detail/1 这样完整的 URL 了，最后 yield 返回即可。

这样我们通过调用 parse_index 方法传入列表页的 HTML 代码就可以获得该列表页所有电影的详情页 URL 了。

好，接下来我们把上面的方法串联调用一下，实现如下：

```
def main():
    for page in range(1, TOTAL_PAGE + 1):
        index_html = scrape_index(page)
        detail_urls = parse_index(index_html)
        logging.info('detail urls %s', list(detail_urls))

if __name__ == '__main__':
    main()
```

这里我们定义了 main 方法来完成上面所有方法的调用，首先使用 range 方法遍历一下页码，得到的 page 是 1~10，接着把 page 变量传给 scrape_index 方法，得到列表页的 HTML，赋值为 index_html 变量。接下来再将 index_html 变量传给 parse_index 方法，得到列表页所有电影的详情页 URL，赋值为 detail_urls，结果是一个生成器，我们调用 list 方法就可以将其输出出来。

好，我们运行一下上面的代码，结果如下：

```
2020-03-08 22:39:50,505 - INFO: scraping https://static1.scrape.cuiqingcai.com/page/1...
2020-03-08 22:39:51,949 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/1
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/2
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/3
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/4
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/5
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/6
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/7
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/8
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/9
2020-03-08 22:39:51,950 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/10
2020-03-08 22:39:51,951 - INFO: detail urls ['https://static1.scrape.cuiqingcai.com/detail/1', 'https://static1.scrape.cuiqingcai.com/detail/2', 'https://static1.scrape.cuiqingcai.com/
2020-03-08 22:39:51,951 - INFO: scraping https://static1.scrape.cuiqingcai.com/page/2...
2020-03-08 22:39:52,842 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/11
2020-03-08 22:39:52,842 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/12
...
```

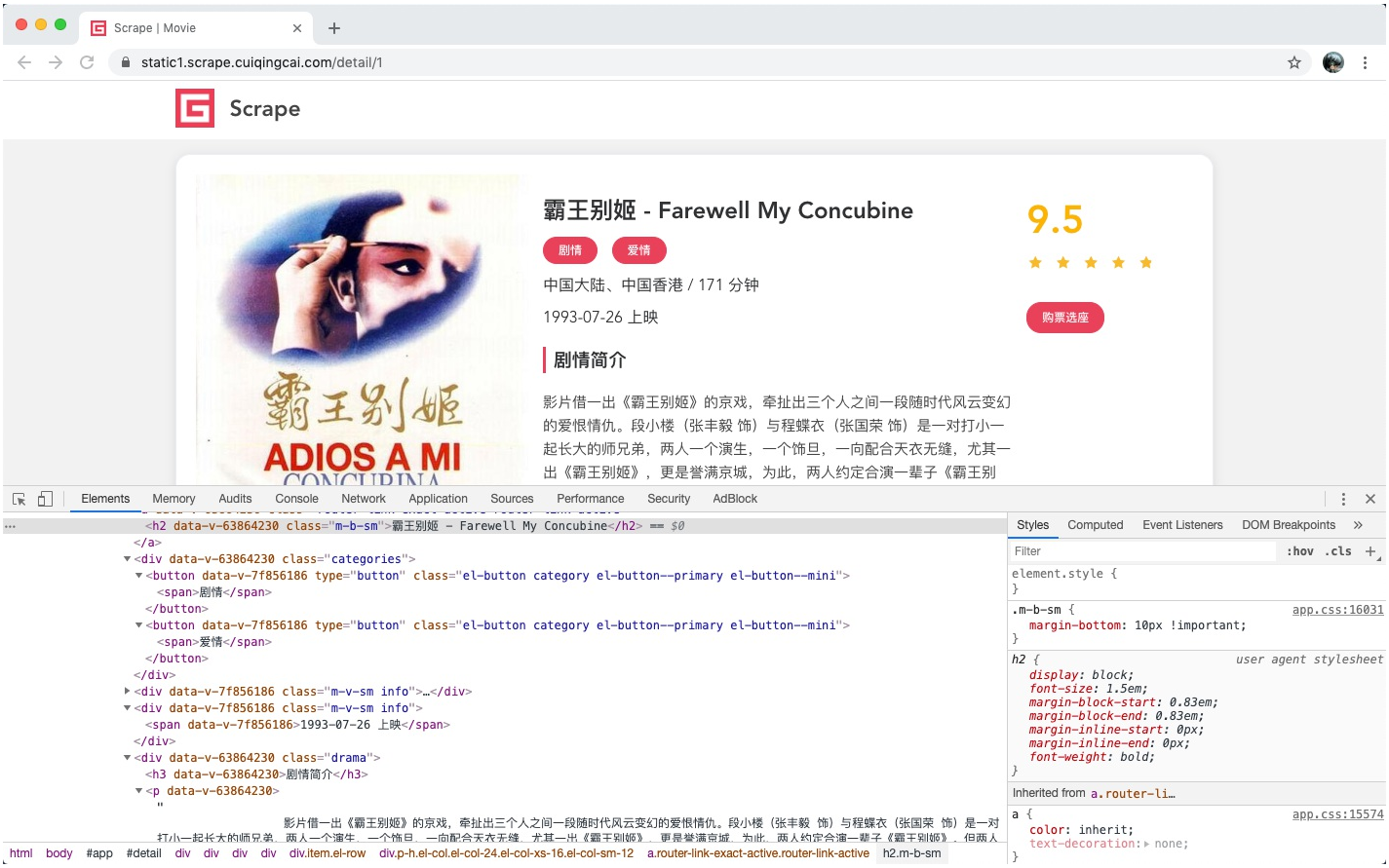
由于输出内容比较多，这里只贴了一部分。

可以看到，在这个过程中程序首先爬取了第 1 页列表页，然后得到了对应详情页的每个 URL，接着再接着爬第 2 页、第 3 页，一直到第 10 页，依次输出了每一页的详情页 URL。这样，我们就成功获取到所有电影详情页 URL 啦。

爬取详情页

现在我们已经成功获取所有详情页 URL 了，那么下一步当然就是解析详情页并提取出我们想要的信息了。

我们首先观察一下详情页的 HTML 代码吧，如图所示：



经过分析，我们想要提取的内容和对应的节点信息如下：

- 封面：是一个 img 节点，其 class 属性为 cover。
- 名称：是一个 h2 节点，其内容便是名称。
- 类别：是 span 节点，其内容便是类别内容，其外侧是 button 节点，再外侧则是 class 为 categories 的 div 节点。
- 上映时间：是 span 节点，其内容包含了上映时间，其外侧是包含了 class 为 info 的 div 节点。但注意这个 div 前面还有一个 class 为 info 的 div 节点，我们可以使用其内容来区分，也可以使用 nth-child 或 nth-of-type 这样的选择器来区分。另外提取结果中还多了「上映」二字，我们可以用正则表达式把日期提取出来。
- 评分：是一个 p 节点，其内容便是评分，p 节点的 class 属性为 score。
- 剧情简介：是一个 p 节点，其内容便是剧情简介，其外侧是 class 为 drama 的 div 节点。

看上去有点复杂，但是不用担心，有了 pyquery 和正则表达式，我们可以轻松搞定。

接着我们来实现一下代码吧。

刚才我们已经成功获取了详情页的 URL，接下来我们要定义一个详情页的爬取方法，实现如下：

```
def scrape_detail(url):
    return scrape_page(url)
```

这里定义了一个 scrape_detail 方法，它接收一个 url 参数，并通过调用 scrape_page 方法获得网页源代码。由于我们刚才已经实现了 scrape_page 方法，所以在这里我们不用再写一遍页面爬取的逻辑了，直接调用即可，这就做到了代码复用。

另外你可能会问，这个 scrape_detail 方法里面只调用了 scrape_page 方法，没有别的功能，那爬取详情页直接用 scrape_page 方法不就好了，还有必要再单独定义 scrape_detail 方法吗？

答案是有必要，单独定义一个 scrape_detail 方法在逻辑上会显得更清晰，而且以后如果我们想要对 scrape_detail 方法进行改动，比如添加日志输出或是增加预处理，都可以在 scrape_detail 里面实现，而不用改动 scrape_page 方法，灵活性会更好。

好了，详情页的爬取方法已经实现了，接着就是详情页的解析了，实现如下：

```
def parse_detail(html):
    doc = pq(html)
    cover = doc('img.cover').attr('src')
    name = doc('a > h2').text()
    categories = [item.text() for item in doc('.categories button span').items()]
    published_at = doc('.info:contains(上映)').text()
    published_at = re.search('(\d{4}-\d{2}-\d{2})', published_at).group(1) \
        if published_at and re.search('\d{4}-\d{2}-\d{2}', published_at) else None
    drama = doc('.drama p').text()
    score = doc('p.score').text()
    score = float(score) if score else None
    return {
        'cover': cover,
        'name': name,
        'categories': categories,
        'published_at': published_at,
        'drama': drama,
        'score': score
    }
```

这里我们定义了 parse_detail 方法用于解析详情页，它接收一个 html 参数，解析其中的内容，并以字典的形式返回结果。每个字段的解析情况如下所述：

- cover：封面，直接选取 class 为 cover 的 img 节点，并调用 attr 方法获取 src 属性的内容即可。
- name：名称，直接选取 a 节点的直接子节点 h2 节点，并调用 text 方法提取其文本内容即可得到名称。
- categories：类别，由于类别是多个，所以这里首先用 .categories button span 选取了 class 为 categories 的节点内部的 span 节点，其结果是多个，所以这里进行了遍历，取出了每个 span 节点的文本内容，得到的便是列表形式的类别。
- published_at：上映时间，由于 pyquery 支持使用 :contains 直接指定包含的文本内容并进行提取，且每个上映时间信息都包含了「上映」二字，所以我们这里就直接使用 :contains(上映) 提取了 class 为 info 的 div 节点。提取之后，得到的结果类似「1993-07-26 上映」这样，但我们并不想要「上映」这两个字，所以我们又调用了正则表达式把日期单独提取出来了。当然这里也可以直接使用 strip 或 replace 方法把多余的文字去掉，但我们为了练习正则表达式的用法，使用了正则表达式来提取。
- drama：直接提取 class 为 drama 的节点内部的 p 节点的文本即可。

- score：直接提取 class 为 score 的 p 节点的文本即可，但由于提取结果是字符串，所以我们需要把它转成浮点数，即 float 类型。

上述字段提取完毕之后，构造一个字典返回即可。

这样，我们就成功完成了详情页的提取和分析了。

最后，我们将 main 方法稍微改写一下，增加这两个方法的调用，改写如下：

```
def main():
    for page in range(1, TOTAL_PAGE + 1):
        index_html = scrape_index(page)
        detail_urls = parse_index(index_html)
        for detail_url in detail_urls:
            detail_html = scrape_detail(detail_url)
            data = parse_detail(detail_html)
            logging.info('get detail data %s', data)
```

这里我们首先遍历了 detail_urls，获取了每个详情页的 URL，然后依次调用了 scrape_detail 和 parse_detail 方法，最后得到了每个详情页的提取结果，赋值为 data 并输出。

运行结果如下：

```
2020-03-08 23:37:35,936 - INFO: scraping https://static1.scrape.cuiqingcai.com/page/1...
2020-03-08 23:37:36,833 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/1
2020-03-08 23:37:36,833 - INFO: scraping https://static1.scrape.cuiqingcai.com/detail/1...
2020-03-08 23:37:39,985 - INFO: get detail data {'cover': 'https://p0.meituan.net/movie/ce4da3e03e655b5b88ed31b5cd7896cf62472.jpg@464w_644h_1e_1c', 'name': '霸王别姬 - Farewell My Concub
2020-03-08 23:37:39,985 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/2
2020-03-08 23:37:39,985 - INFO: scraping https://static1.scrape.cuiqingcai.com/detail/2...
2020-03-08 23:37:41,061 - INFO: get detail data {'cover': 'https://p1.meituan.net/movie/6bea9af4524dfbd0b668eaa7e187c3df767253.jpg@464w_644h_1e_1c', 'name': '这个杀手不太冷 - Léon', 'cate
2020-03-08 23:37:41,062 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/3
...
```

由于内容较多，这里省略了后续内容。

可以看到，我们已经成功提取出每部电影的基本信息，包括封面、名称、类别，等等。

保存到 MongoDB

成功提取到详情页信息之后，下一步我们就要把数据保存起来了。在上一课时我们学习了 MongoDB 的相关操作，接下来我们就把数据保存到 MongoDB 吧。

在这之前，请确保现在有一个可以正常连接和使用的 MongoDB 数据库。

将数据导入 MongoDB 需要用到 PyMongo 这个库，这个在最开始已经引入过了。那么接下来我们定义一下 MongoDB 的连接配置，实现如下：

```
MONGO_CONNECTION_STRING = 'mongodb://localhost:27017'
MONGO_DB_NAME = 'movies'
MONGO_COLLECTION_NAME = 'movies'

client = pymongo.MongoClient(MONGO_CONNECTION_STRING)
db = client['movies']
collection = db['movies']
```

在这里我们声明了几个变量，介绍如下：

- MONGO_CONNECTION_STRING：MongoDB 的连接字符串，里面定义了 MongoDB 的基本连接信息，如 host、port，还可以定义用户名密码等内容。
- MONGO_DB_NAME：MongoDB 数据库的名称。
- MONGO_COLLECTION_NAME：MongoDB 的集合名称。

这里我们用 MongoClient 声明了一个连接对象，然后依次声明了存储的数据库和集合。

接下来，我们再实现一个将数据保存到 MongoDB 的方法，实现如下：

```
def save_data(data):
    collection.update_one({
        'name': data.get('name')
    }, {
        '$set': data
    }, upsert=True)
```

在这里我们声明了一个 save_data 方法，它接收一个 data 参数，也就是我们刚才提取的电影详情信息。在方法里面，我们调用了 update_one 方法，第 1 个参数是查询条件，即根据 name 进行查询；第 2 个参数是 data 对象本身，也就是所有的数据，这里我们用 $set 操作符表示更新操作；第 3 个参数很关键，这里实际上是 upsert 参数，如果把这个设置为 True，则可以做到存在即更新，不存在即插入的功能，更新会根据第一个参数设置的 name 字段，所以这样可以防止数据库中出现同名的电影数据。

注：实际上电影可能有同名，但该场景下的爬取数据没有同名情况，当然这里更重要的是实现 MongoDB 的去重操作。

好的，那么接下来我们将 main 方法稍微改写一下就好了，改写如下：

```
def main():
    for page in range(1, TOTAL_PAGE + 1):
        index_html = scrape_index(page)
        detail_urls = parse_index(index_html)
        for detail_url in detail_urls:
            detail_html = scrape_detail(detail_url)
            data = parse_detail(detail_html)
            logging.info('get detail data %s', data)
            logging.info('saving data to mongodb')
            save_data(data)
            logging.info('data saved successfully')
```

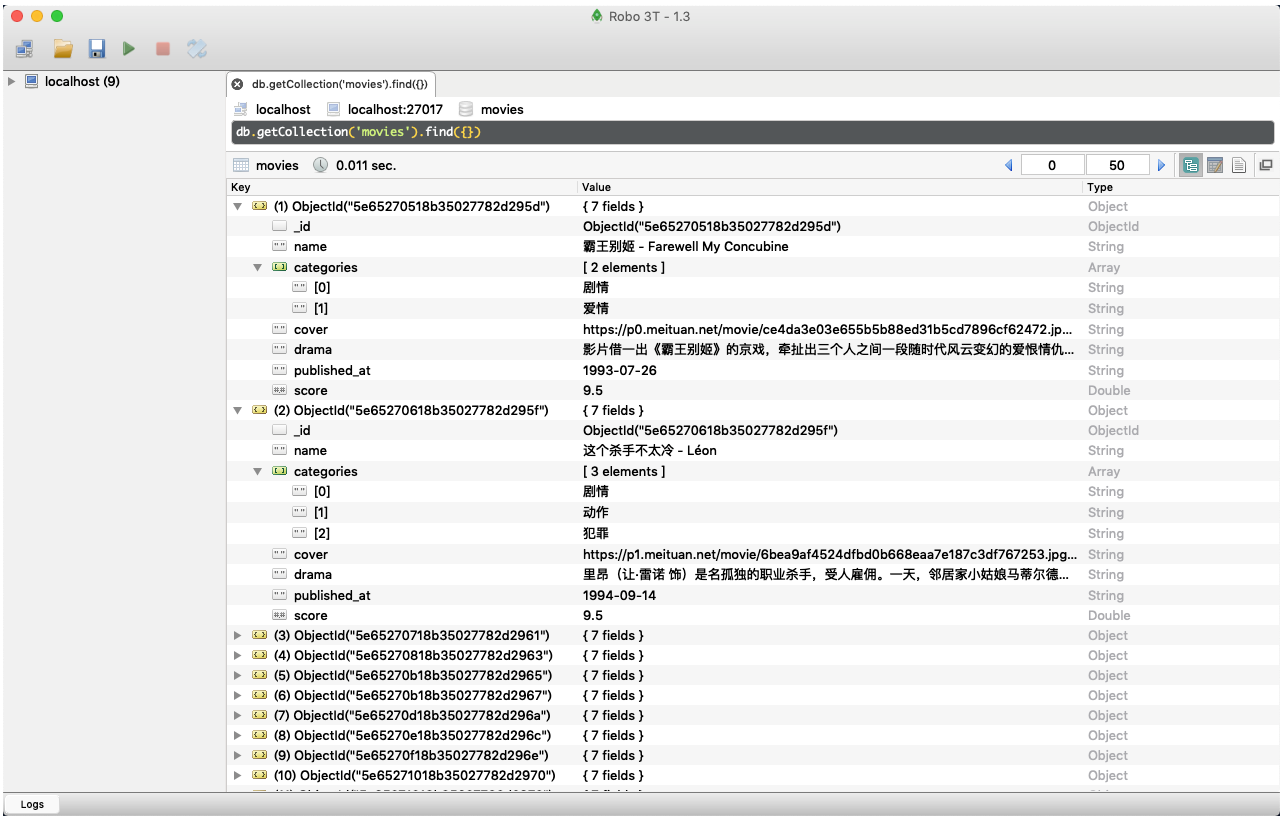
这里增加了 save_data 方法的调用，并加了一些日志信息。

重新运行，我们看下输出结果：

```
2020-03-09 01:10:27,094 - INFO: scraping https://static1.scrape.cuiqingcai.com/page/1...
2020-03-09 01:10:28,019 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/1
2020-03-09 01:10:28,019 - INFO: scraping https://static1.scrape.cuiqingcai.com/detail/1...
2020-03-09 01:10:29,183 - INFO: get detail data {'cover': 'https://p0.meituan.net/movie/ce4da3e03e655b5b88ed31b5cd7896cf62472.jpg@464w_644h_1e_1c', 'name': '霸王别姬 - Farewell My Concub
2020-03-09 01:10:29,183 - INFO: saving data to mongodb
2020-03-09 01:10:29,288 - INFO: data saved successfully
2020-03-09 01:10:29,288 - INFO: get detail url https://static1.scrape.cuiqingcai.com/detail/2
2020-03-09 01:10:29,288 - INFO: scraping https://static1.scrape.cuiqingcai.com/detail/2...
2020-03-09 01:10:30,250 - INFO: get detail data {'cover': 'https://p1.meituan.net/movie/6bea9af4524dfbd0b668eaa7e187c3df767253.jpg@464w_644h_1e_1c', 'name': '这个杀手不太冷 - Léon', 'cate
2020-03-09 01:10:30,250 - INFO: saving data to mongodb
2020-03-09 01:10:30,253 - INFO: data saved successfully
...
```

在运行结果中我们可以发现，这里输出了存储 MongoDB 成功的信息。

运行完毕之后我们可以使用 MongoDB 客户端工具（例如 Robo 3T ）可视化地查看已经爬取到的数据，结果如下：



这样，所有的电影就被我们成功爬取下来啦！不多不少，正好 100 条。

多进程加速

由于整个的爬取是单进程的，而且只能逐条爬取，速度稍微有点慢，有没有方法来对整个爬取过程进行加速呢？

在前面我们讲了多进程的基本原理和使用方法，下面我们就来实践一下多进程的爬取吧。

由于一共有 10 页详情页，并且这 10 页内容是互不干扰的，所以我们可以一页开一个进程来爬取。由于这 10 个列表页页码正好可以提前构造成一个列表，所以我们可以选用多进程里面的进程池 Pool 来实现这个过程。

这里我们需要改写下 main 方法的调用，实现如下：

```
import multiprocessing

def main(page):
    index_html = scrape_index(page)
    detail_urls = parse_index(index_html)
    for detail_url in detail_urls:
        detail_html = scrape_detail(detail_url)
        data = parse_detail(detail_html)
        logging.info('get detail data %s', data)
        logging.info('saving data to mongodb')
        save_data(data)
        logging.info('data saved successfully')

if __name__ == '__main__':
    pool = multiprocessing.Pool()
    pages = range(1, TOTAL_PAGE + 1)
    pool.map(main, pages)
    pool.close()
    pool.join()
```

这里我们首先给 main 方法添加一个参数 page，用以表示列表页的页码。接着我们声明了一个进程池，并声明 pages 为所有需要遍历的页码，即 1~10。最后调用 map 方法，第 1 个参数就是需要被调用的方法，第 2 个参数就是 pages，即需要遍历的页码。

这样 pages 就会被依次遍历。把 1~10 这 10 个页码分别传递给 main 方法，并把每次的调用变成一个进程，加入到进程池中执行，进程池会根据当前运行环境来决定运行多少进程。比如我的机器的 CPU 有 8 个核，那么进程池的大小会默认设定为 8，这样就会同时有 8 个进程并行执行。

运行输出结果和之前类似，但是可以明显看到加了多进程执行之后，爬取速度快了非常多。我们可以清空一下之前的 MongoDB 数据，可以发现数据依然可以被正常保存到 MongoDB 数据库中。

总结

到现在为止，我们就完成了全站电影数据的爬取并实现了存储和优化。

这节课我们用到的库有 requests、pyquery、PyMongo、multiprocessing、re、logging 等，通过这个案例实战，我们把前面学习到的知识都串联了起来，其中的一些实现方法可以好好思考和体会，也希望这个案例能够让你对爬虫的实现有更实际的了解。

本节代码：https://github.com/Python3WebSpider/ScrapeStatic1。