myanimelist.net 数据爬虫

本课程所用的用户评分数据绝大部分来源于Kaggle上的开源数据：https://www.kaggle.com/CooperUnion/anime-recommendations-database . 不过这些数据还是偏少的，很多动漫的信息（如海报url）并未提供。因此为了能获得更多的信息以便推荐系统和后续的UI展示使用，我们可以做一个爬虫直接从myanimelist.net网站上进行信息抓取。

通常而言，编写爬虫获取自己所需的数据也是作为推荐系统工程师的基本能力之一。

原始网页结构分析

要想实现爬虫，首先我们就要熟悉数据源 myanimelist.net 的网页结构。
随便打开一个动漫详情页并且查看源码，我们可以看到类似如下的html标签:

```
<span class="dark_text">Japanese:</span> 錆喰いビスコ
```

这段代码就包含了当前动漫的日语标题信息。类似的，我们还可以找到当前动漫的海报图片url和上映时间等信息，如下：

海报图片url:

```
<a href="https://myanimelist.net/anime/48414/Sabikui_Bisco/pics">
  <img class="lazyload" data-src="https://cdn.myanimelist.net/images/anime/1446/118840.jpg" alt="Sabikui Bisco" class="ac" itemprop="image">
</a>
```

上映时间：

```
<span class="dark_text">Aired:</span>
  Jan 11, 2022 to ?
```

编写scrapy爬虫

在python中想要实现一个爬虫，最经典的框架就是使用scrapy（https://scrapy.org/ ）。如果同学们对scrapy还不是很熟悉，可以先自行在网上学习一下相关的教程，相信只要花几十分钟的时间你就可以掌握基本的概念。下面我们和大家一起基于scrapy实现一下课程项目的爬虫：

1. 安装scrapy并初始化

安装scrapy十分方便，只要运行`pip install Scrapy`即可。
在完成后我们可以使用命令行工具初始化一个空的scrapy项目，方便我们后续的工作。相关命令为`scrapy startproject animelist`，运行后会在当前目录下创建一个animelist文件夹，里面有项目相关的初始文件。

2. 编写anime spider

下面我们就要进入爬虫的编写了，首先我们需要先指定所要爬取的网页url。
对于每个动漫，其在myanimelist上的详情页url由2部分构成：一是myanimelist的基础url，对于所有动漫都是相同的，这个值为`https://myanimelist.net/anime`。另一部分则是动漫的id，这个信息我们可以在Kaggle上下载的anime.csv文件中得到。
有了这两部分信息，只要将他们做字符串拼接就可以得到所有需要爬取的动漫详情页url了。

要实现这些，我们先在`spiders`目录下新建一个`anime_spider.py`文件，作为我们的爬虫入口文件。然后我们在其中定义一个类`AnimeSpider`并让其实现`start_request`方法：

```
BASE_URL = 'https://myanimelist.net/anime'

class AnimeSpider(scrapy.Spider):
    name = "anime"

    def start_requests(self):
        with open('../../data/anime/anime.csv') as csv_file:
            reader = csv.DictReader(csv_file)
            anime_ids = [row["anime_id"] for row in reader]

            with open('output.csv') as output_file:
                output_reader = csv.DictReader(output_file)
                existing_anime_ids = [row['anime_id'] for row in output_reader]

                anime_ids = [id for id in anime_ids if id not in existing_anime_ids]

                for id in anime_ids:
                    yield scrapy.Request(url=f'{BASE_URL}/{id}', callback=self.parse, meta={'anime_id': id})
```

这里我们首先使用csv工具将Kaggle上下载的`anime.csv`文件进行解析和读取，获取到需要处理的动漫id数组。由于数量较大，运行可能会比较慢，因此我这边做了断点恢复的功能，可以从上一次的结果`output.csv`中读取已经解析完毕的动漫列表，将其排除。

随后调用scrapy的Request方法开始真正进行数据爬取。对于每次请求，我们都要告诉scrapy如何来解析网页并获取到我们所需要的数据。

3. 解析网页

在实现了`start_requests`方法后，我们下一步就是要实现网页的元素解析以及信息提取了，我们在`AnimeSpider`类中新增一个方法来完成这个工作：

```
    def parse(self, response):
        anime_id = response.meta.get("anime_id")

        japanese_titles = response.xpath("//span[contains(text(), 'Japanese')]/../text()").getall()
        japanese_titles = [text.strip() for text in japanese_titles]
        japanese_titles = [text for text in japanese_titles if len(text) > 0][0]

        aired = response.xpath("//span[contains(text(), 'Aired')]/../text()").getall()
        aired = [text.strip() for text in aired]
        aired = [text for text in aired if len(text) > 0][0]

        image_url = response.xpath("//a[contains(@href, 'pics')]/img/@data-src").getall()[0]

        print(f"{japanese_titles}    {aired}")
```

```
        yield {
            'anime_id': anime_id,
            'japanese_title': japanese_titles,
            'aired': aired,
            'image_url': image_url
        }
```

这个方法就会按照我们之前观察到的数据结构，从网页上获取想要的元素，并且返回最终的结果。

4. 配置程序

scrapy的爬虫核心代码我们已经有了，下面我们在`settings.py`文件中加入如下几行代码来对项目进行配置：

```
BOT_NAME = 'anime_scrapy'

SPIDER_MODULES = ['anime_scrapy.spiders']
NEWSPIDER_MODULE = 'anime_scrapy.spiders'
```

5. 运行爬虫

最后让我们来运行爬虫看一下效果，运行的命令如下：

```
scrapy crawl anime -o output.csv -t csv
```

如果一切正常，爬虫就会自动运行并且在`output.csv`文件中记录结果。