**课题名称：**基于NoSQL的市场行情数据爬取系统

**简介：**利用爬虫技术，获取海量市场行情信息，支持建设基于NoSQL大数据技术的市场行情系统。

**主要任务及进展：**

1. 掌握编程语言Python的基本用法，掌握主要使用的爬虫py库用法。（√）
2. 掌握Mongodb数据库的基本用法。（√）
3. 完成国内A股的历史行情数据的爬取和每日更新。（√）
4. 完成国内基金的历史行情数据的爬取和每日更新。（√）
5. 完成上证、深证数据的爬取和每日更新。（√）
6. 完成美股的历史行情数据的爬取和每日更新。（√）
7. 完成美国基金的数据爬取。（√）

**现阶段存在的主要问题及解决方法：**

主要问题：

1. 由于要爬取的网站较多，爬虫脚本过于杂乱，缺乏耦合性，需要整理代码。
2. 对有些爬虫算法和架构了解不够透彻，需要进一步优化。
3. 对于twisted框架的非阻塞流程仍不是很熟悉。

解决方法：

1. 上网查找相关资料，掌握相关知识。

**开题以来的进展：**

**使用工具：**云服务器，Python，Pycharm，MongoDB等硬件及软件。

**爬取网站：**

东方财富网（A股、美股、指数每日数据）、

网易财经（A股历史数据、指数历史数据）、

天天基金网（基金每日、历史数据）、

Nasdaq（美股历史数据）、

英为财情（美国基金历史数据和每日数据）

使用框架和主要python库：scrapy、aiohttp、asyncio、selenium、requests、twisted

**历史行情数据的爬取：**

**最初设计思路：**

步骤：

1. 从东方财富网中的股票/基金查询代码页面中将所有股票/基金代码爬取下来并存储在MongoDB。
2. 找到网易财经的数据下载API服务，通过迭代将在数据库的代码拼接成一个个带爬取的URL。（基金页面没有提供数据下载服务，只能通过解析页面获取历史记录）
3. 对url生成器传过来的每个URL封装成一个个request（加上自定义的UESR-AGENT、代理IP等等），发送请求（可使用协程、多线程等手段提高效率），下载csv文件。
4. 对下载下来的文件逐个文件逐行读取，最终存入MongoDB数据库中。

**代码分析：**

1. **连接数据库**
2. **建立对象，补全请求头格式**
3. **request请求代码页面，并对返回结果进行xpath解析**
4. **使用正则表达式筛选代码，并存入数据库**

5、**通过基金代码进入基金历史交易记录页面，补全请求头并发送**request

6、**下载并解析页面，此处使用了aiohttp和asyncio协程**

self.sem = asyncio.Semaphore(32) # 信号量，控制协程数

在控制使用协程函数时前面加上async关键字，如：

async def analy\_page()：

**发送request的函数get\_content：**

此处创建了一个session对象，aiohttp.ClientSession是aiohttp的request对象，结合下面await关键字，当程序运行到await时，为了节省response响应而等待的时间，把该协程挂起跳出，转去执行下一个任务（发送下一个request），但这整个过程都是一个线程在操作，这就是单线程协程，经测试，在限制最多32个协程数时，效率可提升7倍。

**主函数run协程运行代码：**

Tqdm为python可视化进度条，存储在数据库中的数据中有status字段，用于记录每个代码的爬取记录。建立一个以analy\_page函数迭代器提供url的ensure\_future列表，建立一个事件循环对象loop，以future对象列表为参数使用loop.run\_until\_complete函数。

股票和基金爬取方式大同小异，只不过将页面解析换成通过api下载数据方式。

**每日股票基金数据的爬取：**

每日股票基金数据获取采用scrapy框架编写爬虫获取，scrapy爬虫比requests+lxml的定制爬虫方式更稳定、快捷，可以断点重爬，降低了运维的成本。

**Scrapy各文件的展示：**

**1.settings.py设置各种参数参数**

**2.item.py定义数据字段**

**3.middlewares.py：**

对scrapy引擎返回的request添加多种浏览器参数，作用类似于requests的添加请求头。

process\_response函数这一步是因为这次有使用selenium来获取动态数据，判断是否为动态网站地址来返回response类型。

**4.主要文件Spiders/crawler.py：**

**基本思路：**

爬虫从start\_requests开始，从url列表中获取url并判断条件，进入不同的回调函数，

通过selenium爬取动态网页数据表格获得正确的页数页码后，开始修改下一个url，通过抓包获取js请求获取该网站请求数据的api，主要修改的get参数有pn（页码）、pz（每页显示数据数量）、Flelds（股票数据显示的数据类型），迭代发送请求，获取全部数据。

请求返回了json格式的数据，通过json解析将数据与item对象一一对应，返回item对象，完成数据获取。基金数据的获取与股票基本一致，不过基金返回的数据不符合json格式解析，只能通过正则匹配筛选出数据。

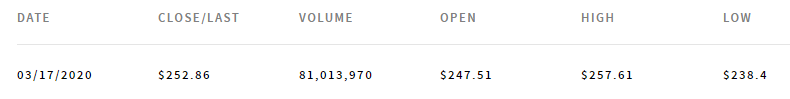
**5.pipelines.py：**

数据的插入主要在这完成，根据data\_type的不同插入不同的数据库集合，后续可以对插入前数据进行修改，减少数据预处理工作量。

**美股历史、每日数据的爬取：**

美股历史数据与A股数据获取大同小异，都是通过网站提供的下载API来获取历史数据，不过由于获取网站不同，数据的字段数量与获取每日数据时的标准不一样，

历史数据：



每日数据：



另外，因为是国外的网站，爬取历史数据时比A股爬取多花了很多的时间，挂VPN爬取时爬取速度也不理想。

美股每日爬取为了获得更多的数据类型，比A股爬取多了一步子页面数据爬取，因为网站传输的get参数有多种类型（如：secid=105./106./107.），使用了多级回调函数来判断返回的json数据是否正确。

在用scrapy框架传输item对象时，发现有爬取数据很多是重复或错乱的，后来发现是由于使用 Request 函数传递 item 时，使用的是浅复制（只复制对象字段值而不复制对象），使用copy的深复制deepcopy解决了问题。

**美国基金历史、每日数据的爬取：**

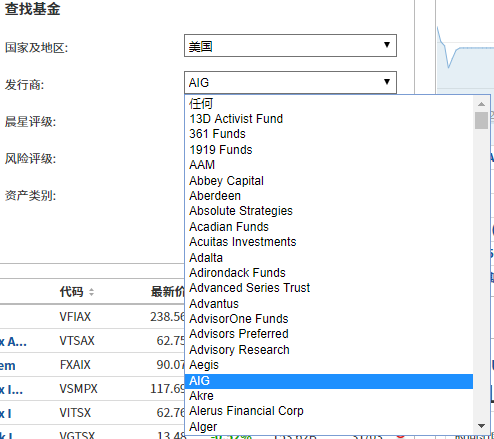
美国基金我爬取选用的网站是英为财情，网站内容齐全，而且网站有设在国内的服务器，所以相对来说更容易获取数据。

设计思路：

（1）该网站的基金页面url：

<https://cn.investing.com/funds/usa-funds?&issuer_filter=0>

观察该url的get参数可知道，issuer\_filter为基金筛选条件，我选择以基金的发行商作为条件。



进入每个发行商的基金页面后，获取基金的页面id（非基金代码）：



data-id的内容就是id。

1. 构建post参数发送到api获得数据。

params = {

"curr\_id": \_id[i], #页面id

"smlID": str(randint(1000000, 99999999)),#随机发送一个数，不可缺少

"st\_date": self.st\_date,#起始时间

"end\_date": self.end\_date,#结束时间

"interval\_sec": 'Daily',#每日数据形式

"sort\_col": "date",

"sort\_ord": "DESC",

"action": "historical\_data" }

1. 对返回的数据进行HTML解析获得有用数据。每日数据的获取不需发送post参数，因为该数据的网页加载不是ajax异步加载的，可直接解析页面获取。

遇到的问题：

1. 使用scrapy.Request请求的页面无法通过网站验证，即便请求头齐全也无法通过，但是用requests库请求可以顺利通过。
2. Post请求失败，参数与网站相同，但是请求后的页面只能定向回主页面无法获得数据。
3. 前两个问题解决后，发现爬虫爬取的速度相较于之前慢了很多。

解决方法：

1. 不同库的请求方式和处理方式也不同，scrapy的Request方式默认不执行JS，对于网站url的重定向也会被默认禁止，导致有些时候无法重定向到需要爬取的网站。

既然requests在测试中可以获取到正确页面内容，那就scrapy结合requests的请求来使用，重写process\_request( )，使用requests请求获取页面内容。

1. post数据没问题，但是请求头缺失了cookie导致请求失败。在请求头加入cookie即可。
2. 因为重写了process\_request( )之后没有考虑请求阻塞的问题，进一步用twisted网络框架重写了非阻塞方法后速度有所提升。但是因为该爬虫有大量的页面解析，cpu计算能力不足，相较于api获取json数据的方法还是慢了许多。