**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，爬虫脚本过于散乱，缺乏耦合性。
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. 爬取数据中可能有较多无用数据，需要进一步筛选。

解决方法：

1. 上网查找相关资料，掌握相关爬虫知识。
2. 找出无用数据的筛选条件，优化数据质量。

**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）

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

**历史行情数据的爬取：**

**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：**

**基金：**

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**

list = pageHTML.xpath('// \*[ @ id = "quotesearch"] / ul / li / a')

解析条件：筛选id为“quotesearch”标签下的ul标签下的li标签下的a标签

1. **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

url=f'http://quotes.money.163.com//fund/jzzs\_{num}\_{i}.html?start={start\_time}&end={end\_time}'

# num为基金代码

# start\_time为开始时间

# end\_time为结束时间

# i为页数

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：

async with self.sem:  
 async with aiohttp.ClientSession() as session:

# cookie字典在clientsession中自定义  
 async with session.get(link, headers={'User-Agent': choice(self.USER\_AGENT)}, timeout=30) as rep:  
 content = await rep.text()  
 pageHTML = etree.HTML(content)  
 return pageHTML

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。

主函数协程运行代码：

for num in tqdm(self.CODES):  
 if num['status'] == 1:  
 num = num['code']  
 tasks = [asyncio.ensure\_future(self.analy\_page(link, num)) for link in self.get\_url(num)]  
 loop = asyncio.get\_event\_loop()  
 loop.run\_until\_complete(asyncio.wait(tasks))

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：**

**Settings.py的参数：**  
USER\_AGENT\_CHOICES # user\_agent集合

ROBOTSTXT\_OBEY = False # 不遵守机器人协议

CONCURRENT\_REQUESTS = 32 # 允许最大request请求数

CONCURRENT\_REQUESTS\_PER\_DOMAIN = 16 # 允许同一域名下最大request请求数

DOWNLOADER\_MIDDLEWARES&ITEM\_PIPELINES #下载中间件和管道的启动

MONGODB\_SERVER = "localhost"

MONGODB\_PORT = 27017

MONGODB\_DB = "keshe"

MONGODB\_GP\_COL = "GP\_test" # 股票

MONGODB\_JJ\_COL = "JJ\_test" # 基金

**Item.py的数据字段定义：**

class DataItem(scrapy.Item):

data\_type = scrapy.Field()

\_id = scrapy.Field()

date = scrapy.Field() # 日期

code = scrapy.Field() # 股票/基金代码

name = scrapy.Field() # 名称

closingPrice = scrapy.Field() # 收盘价

maxPrice = scrapy.Field() # 最高价

minPrice = scrapy.Field() # 最低价

openingPrice = scrapy.Field() # 开盘价

previousClose = scrapy.Field() # 前开盘

change = scrapy.Field() # 涨跌额

quoteChange = scrapy.Field() # 涨跌幅

turnoverRate = scrapy.Field() # 换手率

volume = scrapy.Field() # 成交量

turnover = scrapy.Field() # 成交金额

totalMarketCapitalization = scrapy.Field() # 总市值

marketCapitalization = scrapy.Field() # 流通市值

fundName = scrapy.Field() # 基金名称

unitNetWorth = scrapy.Field() # 单位净值

cumulativeNetWorth = scrapy.Field() # 累计净值

growthRate = scrapy.Field() # 增长率

**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。

def process\_response(self,response,request,spider):

if request.url in ['http://quote.eastmoney.com/center/gridlist.html#hs\_a\_board', f'http://fund.eastmoney.com/data/diyfundranking.html#tall;c0;r;sqjzf;pn50;ddesc;qsd{self.yesterday.strftime("%Y%m%d")};qed{self.today.strftime("%Y%m%d")};qdii']:  
 spider.bro.get(url=request.url)  
 time.sleep(2) # 一定要给与浏览器一定的缓冲加载数据的时间  
 page\_text = spider.bro.page\_source  
 # 篡改响应对象  
 return HtmlResponse(url=spider.bro.current\_url, body=page\_text, encoding='utf-8', request=request)  
 else:  
 return response

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。

**主要文件Spiders/crawler.py：**

**基本思路：**

def start\_requests(self):  
 for each in self.url:  
 if each in ['http://quote.eastmoney.com/center/gridlist.html#hs\_a\_board']:  
 logger.debug(f'现在开始爬取股票数据------')  
 yield scrapy.Request(url=each,callback=self.parse)  
 else:  
 logger.debug(f'现在开始爬取基金数据------')  
 yield scrapy.Request(url=each,callback=self.jijin\_parse)

爬虫从这里开始，从url列表中获取url并判断条件，进入不同的回调函数，

def parse(self,response):  
 page\_num = response.xpath('//span[@class="paginate\_page"]/a/text()')[-1].extract()  
 logger.debug('页数:' + page\_num)  
 self.bro.quit()  
 for page in range(int(page\_num)):  
 url = f'http://26.push2.eastmoney.com/api/qt/clist/get?pn={str(page+1)}&pz=20&po=1&np=1&ut=bd1d9ddb04089700cf9c27f6f7426281&fltt=2&invt=2&fid=f3&fs=m:0+t:6,m:0+t:13,m:0+t:80,m:1+t:2,m:1+t:23&fields=f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f12,f13,f14,f15,f16,f17,f18,f20,f21,f23,f24,f25,f22,f11,f62,f128,f136,f115,f152'  
 yield scrapy.Request(url=url,callback=self.item\_parse,meta={'page': str(page+1),'download\_timeout': 30})

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请求发现该网站请求数据的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），根据获得的总页码数迭代发送请求，获取全部数据

def item\_parse(self,response):  
 item = DataItem()  
 page = response.meta['page']  
 print(f'正在爬取股票第{page}页------')  
 rep = json.loads(response.text)  
 try:  
 data\_list = rep['data']['diff']  
 for each in data\_list:  
 item['date']=self.time  
 item['data\_type']=0 # 股票  
 item['code']=each['f12']  
 item['name']=each['f14']  
 item['closingPrice']=str(each['f2'])  
 item['maxPrice']=str(each['f15'])  
 item['minPrice']=str(each['f16'])  
 item['openingPrice']=str(each['f17'])  
 item['previousClose']=str(each['f18'])  
 item['change']=str(each['f4'])  
 item['quoteChange']=str(each['f3'])  
 item['turnoverRate']=str(each['f8'])  
 item['volume']=str(each['f5'])  
 item['turnover']=str(each['f6'])  
 item['totalMarketCapitalization']=str(each['f20'])  
 item['marketCapitalization']=str(each['f21'])  
 self.num += 1  
 yield item  
 logger.debug('num = '+ str(self.num))  
 except Exception as e:  
 logger.debug(e)

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。

**Pipelines.py：**

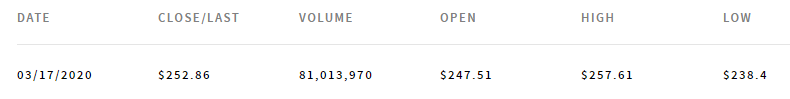
class MongoDBPipeline(object):  
 def \_\_init\_\_(self):  
 # 连接池连接  
 connection = pymongo.MongoClient(  
 settings['MONGODB\_SERVER'],  
 settings['MONGODB\_PORT']  
 )  
 self.db = connection[settings['MONGODB\_DB']]  
 self.GP\_col = self.db[settings['MONGODB\_GP\_COL']]  
 self.JJ\_col = self.db[settings['MONGODB\_JJ\_COL']]  
 # self.time = time.strftime("%Y-%m-%d", time.localtime())  
  
 def process\_item(self, item, spider):  
 try:  
 if item['data\_type'] == 0: # 股票  
 self.GP\_col.insert\_one(dict(item))  
 else: # 基金  
 self.JJ\_col.insert\_one(dict(item))  
 except (Exception) as err:  
 raise DropItem(err)  
 return item

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。