## 版本控制

无论项目大小，我认为但凡属于编程的范畴，你都应该使用版本控制系统，这个编程本身的特点有很大的关系。版本控制系统有很多，其中 git 和 svn 名气和影响比较大，就目前而言 git 是最好的选择，没有之一。

我这里不再累述，如果你有些疑问，下面两篇文章或许可以帮助到你：

[为什么要使用版本控制系统？](https://www.git-tower.com/learn/git/ebook/cn/command-line/basics/why-use-version-control)

[为什么选择 Git？](https://www.git-tower.com/learn/git/ebook/cn/command-line/appendix/why-git)

对 git 还没有任何了解？廖雪峰的教程还是很不错的，可以简要的阅读下：

[Git 教程](https://www.liaoxuefeng.com/wiki/896043488029600)

### Github 的使用

[创建仓库](https://help.github.com/cn/github/getting-started-with-github/create-a-repo)

如果是一个人开发，可以直接在 master 分支下修改提交代码，不过规范的做法是要创建新的分支：

git branch dev # 创建 dev 分支  
git checkout dev # 切换到 dev 分支  
  
# or  
  
git branch -b dev # 等价于上面的两条命令

在本地开发并测试结束以后，需要将 dev 分支 push 到 Github，再通过[创建拉取请求 new pull request](https://help.github.com/cn/enterprise/2.19/user/github/collaborating-with-issues-and-pull-requests/creating-a-pull-request#creating-the-pull-request) 的方式和 master 分支进行合并。

为什么要这么麻烦呢？

两个原因：

1. 如果我们创建了 CI，在提交合并请求的时候会自动触发 CI 来完成你提前规定的任务，比如代码测试、自动部署等，这使得我们在合并代码的时候就能完成很多任务，十分的便捷。

* [关于持续集成](https://help.github.com/cn/actions/automating-your-workflow-with-github-actions/about-continuous-integration)

1. 创建 pull request 之后可以指定队友帮你 review，可以在一定程度上避免严重的 bug，保证代码质量。

## Scrapy 开发

Scrapy 很简单，也很强大。。。

Scrapy 用默认的配置就可以很好地完成任务，如果不符合需求，也很方便的对各种模块进行自定义。

### 定义 Item

首先需要对保存的数据内容进行定义

|  |  |
| --- | --- |
| 名称 | 含义 |
| id | 每条数据的唯一标识符 |
| news\_id | 新闻的唯一标识符 |
| url | 新闻地址 |
| content | 新闻内容 |
| category | 新闻类别 |
| source | 新闻来源 |
| data | 新闻日期 |
| page | 当前页数 |
| entry\_time | 入库时间 |

中国公安网上一则新闻可能分为多页显示，我们对每一页分开进行保存，因为如果合并保存的话有可能页面太长导致存储错误。

id 是 url 和 method 的 sha1 值。

from scrapy.utils.request import request\_fingerprint  
  
id = request\_fingerprint(response.request)

### 解析页面

一共有 25 个类别的新闻数据需要抓取，这也新闻页面之间是有差异的，如果分开编写解析函数工作量太大，在对页面进行分析之后找到了很多共同点，很多数据可以使用相同 Xpath 进行提取。

|  |  |
| --- | --- |
| 数据项 | Xpath |
| title | //\*[@id="newslist"]/h1/gettitle/text() |
| content | //\*[@id="fz\_test"]/div[1]/table |
| category | //\*[@id="source\_report"]/text() |
| date | //\*[@id="pub*time*report"]/text() |

在提取总页数和当前页数的时候，

我们需要从 url 里面提取 news\_id，url 有两种不同的格式，需要用不同的正则表达式进行分别处理：

# http://zhian.cpd.com.cn/n26237006/202001/t20200120\_877942.html  
p\_path1 = re.compile('(.\*/)(.\*?\_.\*?)\.html')  
  
# 'http://jt.cpd.com.cn/n462015/c4191056/content.html'  
p\_path2 = re.compile('(.\*?)content.html')

在提取新闻的总页数和当前页数的时候有两种不同页面，也需要用两种不同的正则表达书进行处理：

# http://jt.cpd.com.cn/n462015/c4191056/content.html  
p\_news1 = re.compile('createPageHTML\((\d+), (\d+),')  
  
# 'http://minsheng.cpd.com.cn/n1448492/c3834444/content.html'  
p\_news2 = re.compile('var maxPageNum=(\d+);.\*?var pageName = (\d+);', re.S)

### 持久化

在获取到数据之后存入到 MySQL 之中，数据表的定义如下：

create schema if not exists news collate utf8mb4\_unicode\_ci;  
  
use news;  
  
create table if not exists cpd\_news  
(  
 id varchar(40) not null,  
 url varchar(255) not null,  
 title varchar(255) not null,  
 content text not null,  
 category varchar(5) not null,  
 source varchar(50) not null,  
 date varchar(30) not null,  
 news\_id varchar(50) not null,  
 page int not null,  
 entry\_time DATETIME not null DEFAULT CURRENT\_TIMESTAMP,  
 constraint data\_id\_uindex  
 unique (id)  
);  
  
alter table data  
 add primary key (id);

利用 PyMysql 库与 MySQL 进行交互。

### URL 去重

每次在启动爬虫的时候，已经抓取的 URL 可以直接跳过不必再重复抓取，Scrapy 提供了去重的方法，但是默认的情况下 URL 的持久化数据存在在了本地，难以迁移，而且请求失败的 URL 也同样被存储了起来，这不符合我们的要求，我们重新定义了 DUPEFILTER\_CLASS 去重方法：

# settings.py  
DUPEFILTER\_CLASS = 'crawler.dupefilters.RFPDupeFilter'  
  
# dupefilters.py  
class RFPDupeFilter(BaseDupeFilter):  
 """Request Fingerprint duplicates filter"""  
  
 def \_\_init\_\_(self, database\_name=None, table\_name=None, filter\_name=None, debug=False):  
 self.fingerprints = set()  
 self.logdupes = True  
 self.debug = debug  
 self.logger = logging.getLogger(\_\_name\_\_)  
 self.fingerprints.update()  
 if database\_name and table\_name:  
 host = settings.get('MYSQL\_HOST', 'localhost')  
 mysql\_user = settings.get('MYSQL\_USER', 'root')  
 mysql\_pwd = settings.get('MYSQL\_PASSWORD', 'news\_crawler')  
 mysql\_port = settings.get('MYSQL\_PORT', 3306)  
 self.db = pymysql.connect(host, mysql\_user, mysql\_pwd, database\_name, mysql\_port)  
 self.cursor = self.db.cursor()  
 sql = "SELECT {0} FROM {1} WHERE 1".format(filter\_name, table\_name)  
 self.cursor.execute(sql)  
 ids = self.cursor.fetchall()  
 ids = map(lambda x: x[0], ids)  
 self.fingerprints.update(ids)  
  
 @classmethod  
 def from\_crawler(cls, crawler):  
 debug = settings.getbool('DUPEFILTER\_DEBUG')  
 return cls(crawler.spider.database\_name, crawler.spider.table\_name, crawler.spider.filter\_name, debug)  
  
 def request\_seen(self, request):  
 fp = self.request\_fingerprint(request)  
 if fp in self.fingerprints:  
 return True  
 self.fingerprints.add(fp)  
  
 def request\_fingerprint(self, request):  
 return request\_fingerprint(request)  
  
 def close(self, reason):  
 if self.db and self.cursor:  
 self.db.close()  
 self.cursor.close()  
  
 def log(self, request, spider):  
 if self.debug:  
 msg = "Filtered duplicate request: %(request)s (referer: %(referer)s)"  
 args = {'request': request, 'referer': referer\_str(request)}  
 self.logger.debug(msg, args, extra={'spider': spider})  
 elif self.logdupes:  
 msg = ("Filtered duplicate request: %(request)s"  
 " - no more duplicates will be shown"  
 " (see DUPEFILTER\_DEBUG to show all duplicates)")  
 self.logger.debug(msg, {'request': request}, extra={'spider': spider})  
 self.logdupes = False  
  
 spider.crawler.stats.inc\_value('dupefilter/filtered', spider=spider)

可以看到，我们在每次启动爬虫的时候从数据库获取 URL 信息，对即将抓去的链接进行检测去重，相对于默认的去重模式，我们重写后的方法不再存在难以迁移的问题，也能更加方便的进行管理。

## 反爬

### 添加代理

由于对方服务器的反爬措施比较严格，添加 IP 代理是最便捷也是最有效的突破反爬的方式。我们使用的是阿布代理，每秒钟最多可以发送 5 次请求，每次请求的代理 IP 会随机变化，所以需要同时限制抓取频率：

# 限制抓取频率  
# settings.py  
CONCURRENT\_REQUESTS = 5  
DOWNLOAD\_DELAY = 0.2  
RANDOMIZE\_DOWNLOAD\_DELAY = False  
  
# 添加代理  
# middlewares.py  
  
# 代理服务器  
proxyServer = "http://http-dyn.abuyun.com:9020"  
  
# 代理隧道验证信息  
proxyUser = ""  
proxyPass = ""  
  
proxyAuth = "Basic " + base64.urlsafe\_b64encode(bytes((proxyUser + ":" + proxyPass), "ascii")).decode("utf8")  
  
  
class ProxyMiddleware(object):  
  
 def process\_request(self, request, spider):  
 request.meta["proxy"] = proxyServer  
 request.headers["Proxy-Authorization"] = proxyAuth

### 随机更换 User-Agent

检测 User-Agent 是反爬的重要手段之一，为了躲避这种反爬手段，我们需要随机更换 User-Agent：

# 随机更换 User-Agent  
# middlewares.py  
  
class RandomUserAgentMiddleware(object):  
 """This middleware allows spiders to override the user\_agent"""  
  
 def \_\_init\_\_(self):  
 self.user\_agent\_list = settings.get('USER\_AGENT\_LIST')  
 self.count = 0  
  
 def process\_request(self, request, spider):  
 request.headers['User-Agent'] = random.choice(self.user\_agent\_list)  
  
# User-Agent 列表  
# settings.py  
USER\_AGENT\_LIST = [  
 'Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10\_6\_8; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50',  
 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-us) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50',  
 'Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0',  
 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0)',  
 'Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)',  
 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:2.0.1) Gecko/20100101 Firefox/4.0.1',  
 'Mozilla/5.0 (Windows NT 6.1; rv:2.0.1) Gecko/20100101 Firefox/4.0.1',  
 'Opera/9.80 (Macintosh; Intel Mac OS X 10.6.8; U; en) Presto/2.8.131 Version/11.11',  
 'Opera/9.80 (Windows NT 6.1; U; en) Presto/2.8.131 Version/11.11',  
 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_7\_0) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Maxthon 2.0)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; TencentTraveler 4.0)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The World)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; SE 2.X MetaSr 1.0; SE 2.X MetaSr 1.0; .NET CLR 2.0.50727; SE 2.X MetaSr 1.0)',  
 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)',  
]

## 部署

### Docker Build

使用 Docker，可以简化部署流程，更轻松的迁移以及更轻松的维护和扩展。

[为什么要用 Docker](https://yeasy.gitbooks.io/docker_practice/introduction/why.html)

FROM python:3.7.3  
  
RUN mkdir /project  
  
WORKDIR /project  
  
ADD crawler/. /project  
  
RUN mkdir error log  
  
RUN pip install -i https://mirrors.aliyun.com/pypi/simple/ -r requirements.txt

### Docker-compose

使用 docker-compose 分别部署数据库和爬虫程序。

数据库：

version: '3'  
  
services:  
  
 db:  
 image: mysql  
 container\_name: mysql-crawler  
 command: mysqld --character-set-server=utf8mb4 --collation-server=utf8mb4\_unicode\_ci #设置utf8字符集  
 restart: always  
 environment:  
 MYSQL\_ROOT\_PASSWORD: news\_crawler  
 ports:  
 - '3306:3306'  
 volumes:  
 - ../data/mysql/db:/var/lib/mysql  
 - ./init/cpd.sql:/docker-entrypoint-initdb.d/init.sql:ro  
 networks:  
 - crawler\_db  
  
 gui:  
 depends\_on:  
 - db  
 image: phpmyadmin/phpmyadmin  
 container\_name: phpmyadmin-crawler  
 restart: always  
 environment:  
 MYSQL\_ROOT\_PASSWORD: news\_crawler  
 PMA\_HOST: db  
 ports:  
 - '8000:80'  
 networks:  
 - crawler\_db  
  
networks:  
 crawler\_db:

爬虫：

version: '3'  
  
services:  
  
 crawler:  
 image: registry.cn-hangzhou.aliyuncs.com/traffic\_news/cpd\_crawler:latest  
 container\_name: crawler  
 command: scrapy crawl cpd  
 volumes:  
 - ./log/:/project/log/  
 networks:  
 - crawler\_net  
  
networks:  
 crawler\_net:

### 自动化构建 Docker 镜像

在每次将代码提交合并到 github 的 master 分支上之后，自动的构建 docker image 并自动上传到镜像仓库。

使用阿里云的[容器镜像服务](https://cr.console.aliyun.com/cn-hangzhou/instances/repositories)，创建自动构建设置，当项目 mater 分支更新则自动构建并上传镜像仓库。

### 定时抓取

利用 Linux crontab 服务，创建定时任务：

crontab -e # 打开服务设置

设置自动定时任务

10,20,30,40,50,59 \* \* \* \* /usr/local/bin/docker-compose -f /root/Projects/traffic\_news/deploy/docker-compose-crawler.yml up -d

重新加载 crontab，生效服务

service cron reload