开源社区开发者流失预测项目爬虫部分说明文档

data\_crawl\_spider部分主要实现了用于爬取GitHub特定仓库的issue、pull request（及review）、commit、fork、star以及各种comment等数据的scrapy爬虫。此文档主要用于说明data\_crawl\_spider的代码实现逻辑、使用方法以及注意事项等，供对该部分代码感兴趣的开发者参考。

data\_crawl\_spider部分基于GitHub REST API实现了两个scrapy爬虫，分别是churnSpider和additionSpider。churnSpider的功能为爬取特定仓库特定时间段的数据，additionSpider设计的初衷是对churnSpider无法正确爬取的数据进行补充，所以其功能为爬取特定API url的数据。下面对项目中GitHub REST API的使用、MySQL数据库、两个爬虫的实现思路以及具体的使用方法进行说明。

1. GitHub REST API

GitHub提供了公开的API来让开发者进行几乎一切的GitHub操作，包括获取仓库公开数据、发issue、修改删除仓库等。data\_crawl\_spider部分的两个scrapy爬虫便是基于GitHub REST API实现的。GitHub REST API官方文档详见<https://docs.github.com/en/rest>。

通过API爬取的数据类型有仓库基本信息、issue数据、issue comment数据、pull request数据、review数据、review comment数据、commit数据、commit comment数据、fork数据、star数据和用户信息等。GitHub REST API提供了对各种数据按照创建/更新时间排序的功能，具体方式是在url的参数中加入sort=created/updated以及direction=asc/desc来控制排序的依据和方向，通过使用该功能可以对数据按照不同时间段进行快捷有效的爬取。上述不同数据的API链接和具体的排序方式如下表所示。

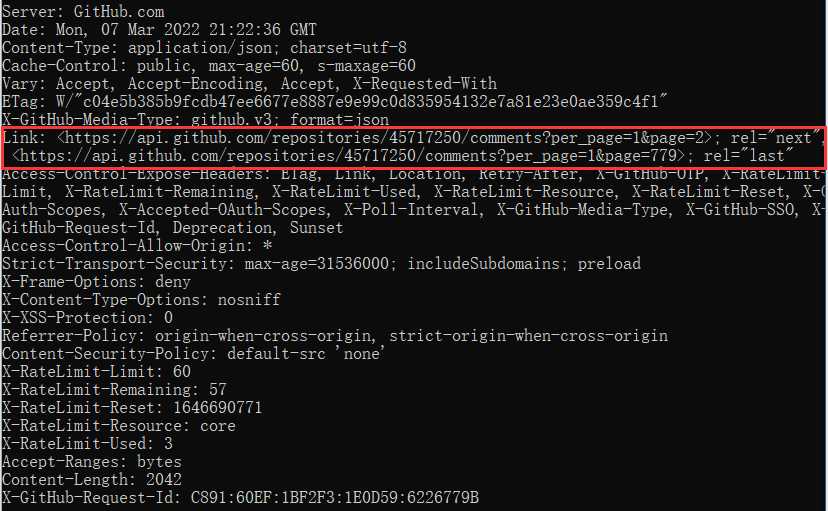
表1 不同数据的API链接和排序方式

|  |  |  |  |
| --- | --- | --- | --- |
| 数据类型 | API链接 | 排序方式 | 备注 |
| 仓库数据 | https://api.github.com/repos/{owner}/{repo} | 无 | 无需排序 |
| issue | https://api.github.com/repos/{owner}/{repo}  /issues | sort=created  direction=desc |  |
| issue comment | https://api.github.com/repos/{owner}/{repo}  /issues/comments | sort=created  direction=desc |  |
| pull request | https://api.github.com/repos/{owner}/{repo}  /pulls | sort=created  direction=desc |  |
| pull request  (detailed) | https://api.github.com/repos/{owner}/{repo}  /pulls/{pull\_number} | 无 | 无需排序 |
| review | https://api.github.com/repos/{owner}/{repo}  /pulls/{pull\_number}/reviews | 无 | 无需排序 |
| review comment | https://api.github.com/repos/{owner}/{repo}  /pulls/{pull\_number}/comments | 无 | 无需排序 |
| commit | https://api.github.com/repos/{owner}/{repo}  /commits | sort=created  direction=desc |  |
| commit comment | https://api.github.com/repos/{owner}/{repo}  /comments | 无 | 自动根据committer提交日期排序 |
| fork | https://api.github.com/repos/{owner}/{repo}/forks | sort=newest |  |
| star | https://api.github.com/repos/{owner}/{repo}/stars | 无 | 自动根据创建时间排序 |
| user | https://api.github.com/users/{username} | 无 | username为用户的登录名（login） |

由于通过API获取的数据资源内容数量庞大，GitHub REST API提供了分页的功能，每个页面可以有最多100条数据（通过per\_page参数设置），且可以根据page参数设置对应页数。同时，在通过API获取数据时，返回的header信息中也会包含前后相邻页和最后一页（部分包含）的链接，如在命令行中执行命令：

curl -i https://api.github.com/repos/tensorflow/tensorflow/comments?per\_page=1

在接受到的header信息中包含有下一页和最后一页的链接：



根据上述分页功能，可以在每次爬取时自动解析下一页的链接从而实现连续的爬取。对于分页功能，还需要注意的是GitHub限制某些数据资源的公开页数，以保证所有用户访问资源的速度。具体限制有：**仓库的issue comment数据最多可以访问400页（每页100条数据）；仓库的star数据最多可以访问400页（每页100条数据）；review和review comment仅可访问2016年9月14日以后数据（具体原因不明）。**其中，issue comment可以通过调整API参数方式将数据爬全（在使用方法中进行详细说明），star和review等数据则无法全部爬取。

GitHub也限制用户对API的访问频率，上图中的X-RateLimit-Limit和X-RateLimit-Remaining分别表示每小时最多可访问次数和剩余可访问次数。这里由于没有进行任何的用户授权操作，所以每小时最多访问60次，超过该次数则会收到403 forbidden的报错提醒。可以通过在请求头中添加用户token进行授权，经过授权后**每个用户**每小时可以访问最多5000次数据。token设置方法可以参考[Creating a personal access token - GitHub Docs](https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token)，该项目中由于只需要访问公共资源，所以申请token时不需要勾选任何的scope，另外需要注意的是如果你**将token包含在代码中提交到了GitHub的公开仓库中，则该token则会被GitHub自动回收。**

1. MySQL数据库

该项目中爬取了30个与AI相关的GitHub仓库，其数据保存在MySQL数据库中。相关的数据库表格和其内容如下表。为了加快后续数据处理时的速度，这里将访问次数较少的comment/message等文本信息单独存放在后缀为extend的表格中。

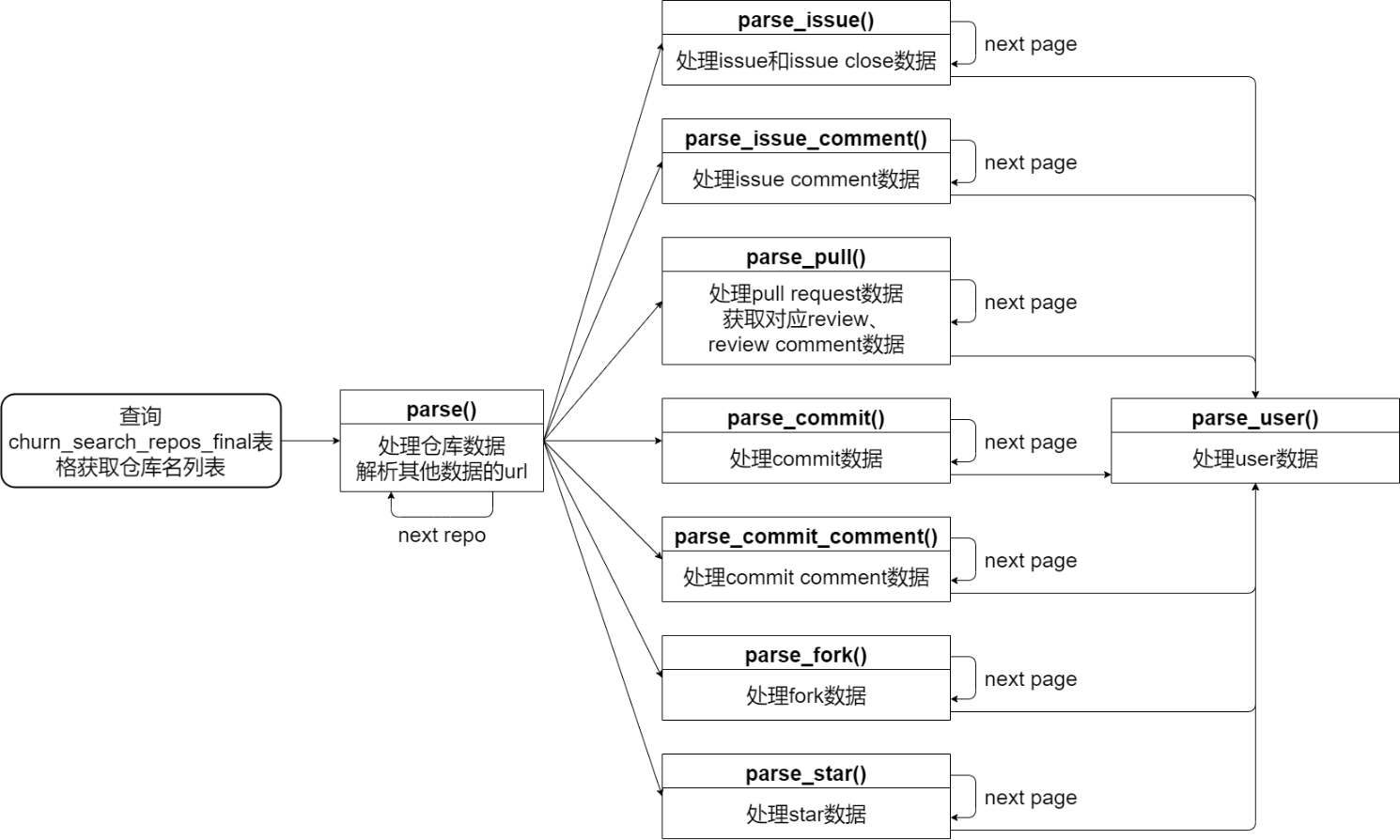
表2. 数据库表格内容

|  |  |
| --- | --- |
| 表名 | 内容 |
| churn\_search\_repos\_final | 选定的30个仓库信息，爬虫爬取的依据 |
| repo\_metadata | 仓库基本信息 |
| repo\_issue | 仓库issue数据 |
| repo\_issue\_extend | issue的title和body等数据 |
| repo\_issue\_closed | 仓库关闭的issue数据 |
| repo\_issue\_comment | issue comment基本数据 |
| repo\_issue\_comment\_extend | issue comment的文本内容数据 |
| repo\_pull | pull request数据 |
| repo\_pull\_extend | pull request的文本数据 |
| repo\_pull\_merged | 合并的pull request的数据 |
| repo\_review | pull request review数据 |
| repo\_review\_comment | review comment数据 |
| repo\_fork | fork数据 |
| repo\_star | star数据 |
| user\_data | 用户信息 |

在scrapy中，为将爬取到的数据存储到对应MySQL表格中，需要根据表格的定义在items.py中定义item（每个item对应一个表格），同时在pipeline中定义对不同item的处理（即数据表格的插入/更新操作）。具体每个表格的定义详见仓库中的数据库定义文件。

1. churnSpider的实现
2. 爬虫实现思路

churnSpider爬虫的基本实现思路如下图所示。在爬虫准备阶段先从数据库中获取需要爬取的仓库名列表，然后设置start\_urls发起爬取过程。parse()函数接收并处理仓库数据，并从中解析出其他数据的url，进而发起新的request。其他数据的爬取结果分别交由对应函数进行处理，parse()函数则请求并处理下一个repo的数据。parse\_issue()和parse\_pull()等函数接收到对应数据后对其进行解析处理，根据其中涉及的user信息发起用户信息爬取的request，并交由parse\_user()处理。同时parse\_issue()等函数会根据返回的header信息解析出下一页，并进行后续页的爬取和处理。



在上述函数中，也会涉及一些其他补充信息或相关信息爬取的功能，如在parse()函数中需要访问仓库对应的languages\_url以获取详细的语言信息，在parse\_pull()函数中需要访问每条pull request的详细信息、以及其review和review comment信息，因此需要单独访问对应的url页面。这些操作的实现并非通过scrapy.Request实现，而是通过requests包中的get函数获取对应页，并在当前函数中对返回的数据进行处理。

另外，在解析获取next page的链接时也需要注意，不同函数的处理是不一样的：对于parse\_issue()、parse\_issue\_comment()、parse\_pull()、parse\_commit()、parse\_fork()，其对应的url有排序功能，且根据创建时间从后往前进行排序，在获取下一页时需要解析header中的rel=next对应的链接；而对于parse\_commit\_comment()和parse\_star()，其对应url没有排序功能，但会根据创建时间自动从前往后排序，为了和其他函数保持一致（从后往前爬取），在parse()函数中需要解析commit comment和star对应url的最后一页，并在parse\_commit\_comment()和parse\_star()中获取下一爬取页时，解析header中的rel=prev对应的链接。

1. 不同API链接的参数设置

在parse()函数中会对接收到的仓库基本数据进行处理，并解析其中包含的其他数据的链接，在发起对其他数据的爬取请求时，为了实现前文所述的从后往前有序爬取、每页100条数据等功能，需要对参数进行设置（排序相关参数可以参考表1的内容）。表3是不同数据API链接的后缀，用于设置参数。

表3. 不同API链接的后缀（参数设置）

|  |  |  |
| --- | --- | --- |
| 数据 | 后缀 | 备注 |
| issue | ?state=all&sort=created&direction=desc&per\_page=100 | 默认page=1 |
| issue comment | ?sort=created&direction=desc&per\_page=100 | 默认page=1 |
| pull | ?state=all&sort=created&direction=desc&per\_page=100 | 默认page=1 |
| commit | ?sort=created&direction=desc&per\_page=100 | 默认page=1 |
| commit comment | ?page=1000&per\_page=100 | 由于排序问题，需设置足够大的page，然后解析尾页链接（rel = last），从后往前爬取 |
| fork | ?per\_page=100&sort=newest | 默认page=1 |
| star | ?per\_page=100 | 然后解析尾页链接（rel = last），从后往前爬取 |

1. 爬取时间段的处理

爬虫爬取仓库数据是根据不同数据的创建时间进行爬取的，而对于一些比较大的仓库，其创建时间久且每年的数据量都很大，很难做到一次性爬取仓库的所有数据，因此需要考虑根据时间段进行分段爬取。

在churnSpider中定义了start\_time和end\_time两个变量用于约束爬取数据的时间段，爬取过程中遇到该时间段以外的数据则会忽略。在parse\_issue()等函数中，也会对接收到的数据提取当前页（每页100条数据）的start time和end time（每一页中的数据是有序的）。如果当前页的start time大于需要爬取时间段的end\_time，则表示还没遍历到对应时间段所在页（注意是从后往前爬取），跳过当前页；如果当前页的end time小于需要爬取时间段的start\_time，则说明已经遍历完对应时间段的所在页，终止该类数据的爬取。这里也需要注意，由于commit comment和star的数据排列顺序和其他数据相反，所以在获取当前页的start time和end time时也和其他函数不同，同时在遍历每页数据时也应从后往前遍历。

1. 日志与错误处理

在爬取过程中，会遇到一些http 错误代码，如401，403，404，502等。这些错误代码需要在settings.py的HTTPERROR\_ALLOWED\_CODES中进行设置，防止代码因遇到这些错误而停止。下面分别说明各种错误的处理方式。

401错误。这种错误一般是因为token过期或使用了错位的token导致的，出现此错误应检查token是否到期，以及是否将token提交到了公开仓库。

502错误。这种错误一般是因为服务器无法响应请求，可以增加settings.py中的RETRY\_TIMES来减少此类错误。

404错误。这种错误是因为找不到对应的页，如查不到某个user的信息。但也可能是GitHub API的一些小漏洞，比如连续的页面中，某一页无法获取，遇到此类错误如果不处理则会中断后续爬取的过程。因此当遇到404报错时，跳过当前页的处理，同时使用代码改写当前url以获取后续页面，并继续爬取。

403错误。这种错误是因为token的使用次数达到了上限（rate limit=5000/user/h），可以通过增加不同用户token的方式解决。具体有两种解决方法：

* 第一种方法是在request\_header.py文件中增加2个备用token，在爬虫代码中判断返回的http code，如果等于403，则更换使用的token（切换churnSpider的参数token\_index），并重新发起scrapy.Request（在scrapy.Requset中设置好切换后的token）,但实际上request的token需要在myUserAgentMiddleware中设置，所以需要在myUserAgentMiddleware中设置token为scrapy.Request中的token（即与spider中使用的token保持一致），这样便实现了token的切换。这一方法在当前token的使用次数达到上限并重复多次失败后会自动切换token，可以保证爬虫持续运行，但由于是在spider中切换的token，每次都是在重复多次失败后才能检测到token失效，所以运行时间会增加；此外，由于很多user数据的request几乎同时发起，切换token也会遇到同步问题。这一方式适合在token数量较少（少于3个）时使用。
* 第二种方法是在middlewares.py文件中设置3个备选token（来自3个不同用户），并在myUserAgentMiddleware中每次设置token时随机选择一个token。这一方法可以充分使用3个token的rate limit，每小时最多可以爬取近15000条数据，因此一般不会遇到403问题。如果某个token使用次数达到上限，每次重试该request时也会随机选择token，因此当设置RETRY\_TIMES较大时一定可以选到一个有效token。这一方法可以有效节约爬取数据时间，适合在token较多（≥3个）时使用。

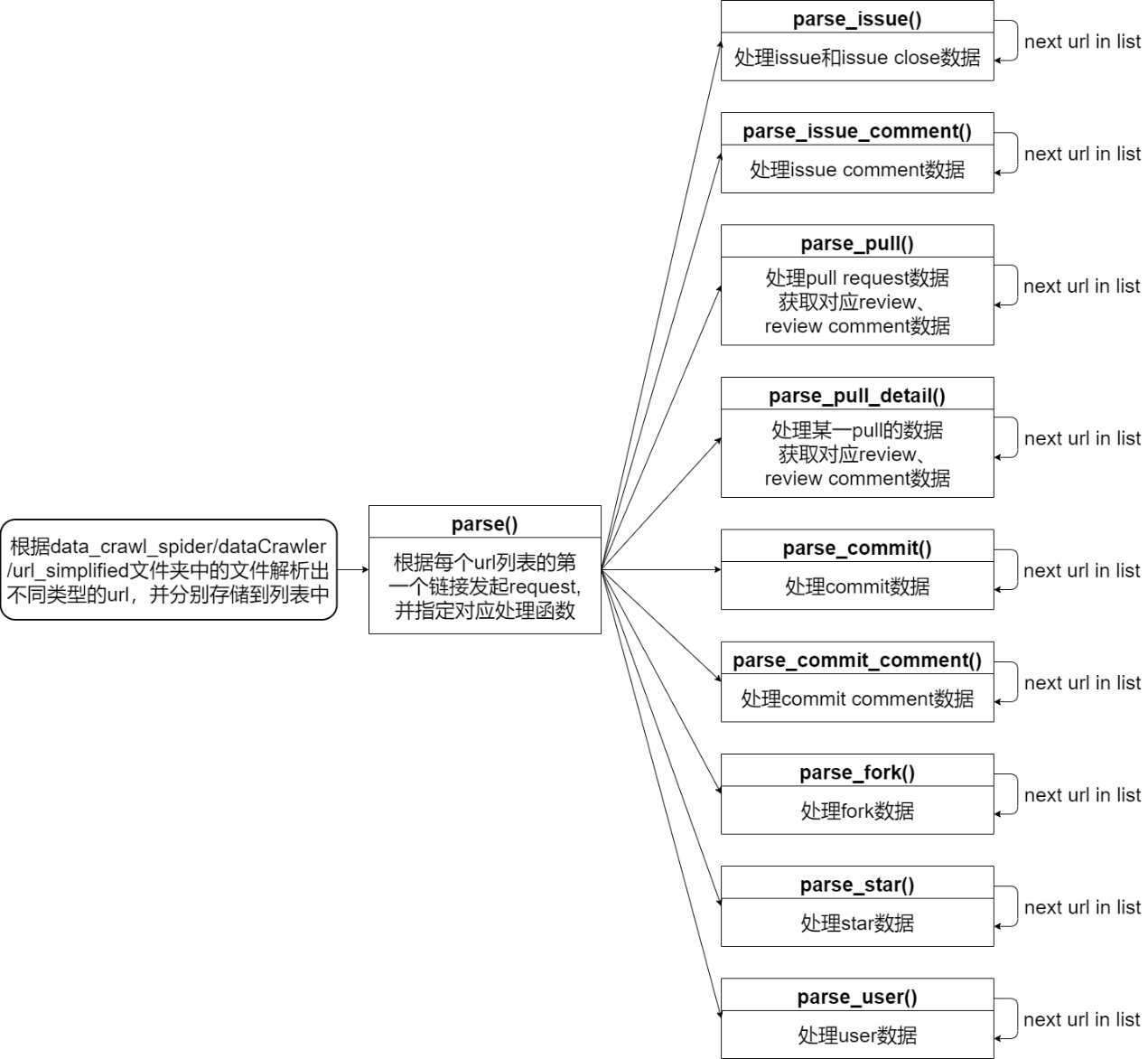
尽管上述几种http error都可以得到有效的解决，但在爬取过程中也会因为网络故障或者代码中未曾考虑到的缺陷而导致某些链接爬取失败，因此我使用python的日志功能记录了所有爬取过程中出现的错误以及对应的url，以供后续在additionSpider中补充爬取。这里我将所有日志记录在all.log文件，将error及以上级别的日志记录在error.log文件。churnSpider中日志存储的文件夹为：data\_crawl\_spider/logs，下面是所有的日志类别和对应的错误内容注解。

表4. 所有日志级别和对应内容注解

|  |  |  |  |
| --- | --- | --- | --- |
| 日志级别 | 日志信息 | 注解 | 影响 |
| INFO | \*\*\* token changed | 403报错，token失效 | 无影响（若同一url连续出现则可能会导致该数据爬取过程中断） |
| INFO | page time before/after period | 不在爬取时间段 | 无影响 |
| WARNING | 404：page is lost | 404错误，某一页缺失 | 当前页爬取失败 |
| WARNING | current url: \*\* next url: \*\* | 404页缺失，寻找下一页，和上一日志同时出现 | 当前页爬取失败 |
| ERROR | Languages url error | 仓库的language 内容获取失败 | 仓库基本信息获取失败，其他数据无法爬取 |
| ERROR | \*\*item error! \*\*\* | item错误 | 后续爬取过程中断 |
| ERROR | \*\*\* url not found | commit comment / star的链接解析失败（可能没有该类数据） | 无法爬取对应数据 |
| ERROR | First page lost! | 404错误，某一类数据首页丢失 | 无法爬取该类数据，后续爬取过程中断 |
| ERROR | page start/end time error! | 无法获取当前页的开始和结束时间 | 后续爬取过程中断 |
| ERROR | data order error! | 当前页的数据排序错误 | 后续爬取过程中断 |
| CRITICAL | repo \*\* data crawl error! | 某一类数据爬取错误 | 后续爬取过程中断 |
| CRITICAL | something wrong with GitHub tokens happened! | token全部过期/使用次数达到上限 | 后续爬取过程中断 |

1. additionSpider的实现
2. 基本实现思路

additionSpider设计的目的是根据API链接爬取对应数据并存储。其主要用途有两个方面，一方面是爬取churnSpider爬取过程中出错或漏爬的url（在日志中记录），另一方面该爬虫也可以用于爬取仓库数据（需要先查找每一类数据的首页和尾页，并生成所有页的url）。其实现思路如下图所示。



additionSpider的代码在data\_crawl\_spider/dataCrawler目录下，首先根据data\_crawl\_spider/ dataCrawler/url\_simplified文件夹下的url文件解析出不同类型的url，并分别存储到列表中。为start\_urls设置一个起始url，并启动爬取过程。在parse()函数中，对不同的url列表进行处理，根据每个列表表首的url发起不同的request，并交由对应函数处理。在对应函数中处理接收的数据后，分别读取对应url列表的后续内容并继续爬取。

1. 关于additionSpider中的时间问题

additionSpider是根据每个API链接进行数据爬取的，一般来说不需要考虑划分时间段的问题，由于该代码是在churnSpider的代码基础上改写的，所以代码中只是简单地扩大了爬取时间段的范围——设置start\_time为2020-01-01，end\_time为2022-01-01。

在每个处理函数中仍保留了对每页首末时间读取和比较的代码（用于简单判断排序是否有问题），因此如果additionSpider爬取的数据链接排序方向是asc（即direction=asc）时，需要修改对应函数中每页开始和结束时间的定义（page\_start\_time和page\_end\_time），否则会报“data order error”的错。

commit数据是根据committer的提交时间进行排序的，但代码中获取的是author的提交时间。author的提交时间和committer的提交时间可能会有比较大的差别，因此在涉及时间的判断时也放宽了限制。如判断page\_start\_time是否大于end\_time（2022-01-01）时，判断的是page\_start\_time - self.end\_time > 36500。这一方式是为了尽可能多地获取commit数据，以避免数据缺失。

1. 日志和错误处理

在additionSpider中也实现了对各类错误的处理和日志记录功能。对于401和403错误，处理方式和churnSpider相同，此处不再赘述。如果遇到了404或502错误，additionSpider会在日志中进行记录，但不做其他处理。additionSpider的日志存储在文件夹data\_crawl\_spider/ dataCrawler/additionSpider/spiderLogs中。所有日志类型和内容注解详见表5。需要注意的是在additionSpider中添加了DEBUG级别的日志，该日志记录每次成功获取内容的url。

表5. 日志级别和内容注解

|  |  |  |  |
| --- | --- | --- | --- |
| 日志级别 | 日志信息 | 注解 | 影响 |
| DEBUG | 链接 | 成功获取内容的url | 成果获取了对应的内容（但未必全部处理完） |
| INFO | \*\*\* token changed | 403报错，token失效 | 无影响（若同一url连续出现则可能会导致该数据爬取过程中断） |
| INFO | page time before/after period | 不在爬取时间段 | 无影响 |
| WARNING | 404/502：page is lost | 404/502错误，某一页缺失或服务器无法响应 | 当前页爬取失败 |
| ERROR | \*\*item error! \*\*\* | item错误 | 当前页爬取失败 |
| ERROR | page start/end time error! | 无法获取当前页的开始和结束时间 | 当前页爬取失败 |
| ERROR | data order error! | 当前页的数据排序错误 | 当前页爬取失败 |
| CRITICAL | repo \*\* data crawl error! | 某一类数据爬取错误 | 当前页爬取失败 |
| CRITICAL | something wrong with GitHub tokens happened! | token全部过期/使用次数达到上限 | 后续爬取过程中断 |

1. 其他辅助工具

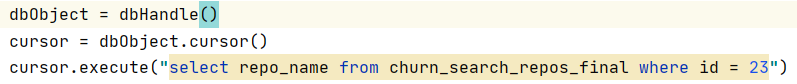
在data\_crawl\_spider/dataCrawler目录中还实现了一些辅助工具，具体如下：

get\_failed\_urls.py，其主要功能是生成根据churnSpider中生成的log文件（应先将文件拷贝至data\_crawl\_spider/dataCrawler/logs目录中）生成不同类型的url文件以供additionSpider补充爬取。生成的文件存储于data\_crawl\_spider/dataCrawler/urls和data\_crawl\_spider/ dataCrawler/urls\_simplified中，前者存储每有消除重复的url文件，后者存储消除重复后的文件。

test\_rate\_limit.py，其主要功能是测试token的当前剩余使用次数（token在同目录下的request\_header.py文件中设置）。可以间断性地测试，以便于在爬虫运行过程中实时监测token的rate limit情况。

1. 使用方法及注意事项
2. churnSpider的使用方法

使用churnSpider爬取所需仓库的数据时，首先需要修改爬虫文件GitHub.py中的MySQL查询语句，搜索需要爬取的仓库名。例如我想爬取churn\_search\_repos\_final表中id=23的仓库，则查询语句为：



然后需要设置爬取的时间段，比如我希望爬取2010年8月1日到2021年10月15日的数据，则设置churnSpider的start\_time和end\_time参数如下：



设置好上述内容即可运行data\_crawl\_spider/churnSpider目录下的run.py来运行爬虫。但在运行前应检查以下内容：

* token的有效性。需要检查GitHub.py所在目录中的request\_header.py，以及middlewares.py中的token列表是否填写且有效，两个文件中的token列表保持一致即可。
* data\_crawl\_spider/logs文件夹是否清空。如果您不希望在上一次运行的logs基础上继续添加日志，则需要在运行前手动清除日志文件。
* 清空data\_crawl\_spider/churnSpider/job\_info/001目录下的内容。该目录是settings.py中设置的job\_dir，如果上一次运行中断后请求队列文件没有删除，可能会影响到此次运行。

1. additionSpider的使用方法

首先需要在data\_crawl\_spider/dataCrawler/urls\_simplified文件夹中添加链接文件，文件名只能选自以下内容：

issue.txt, issue\_comment.txt, pull.txt, pull\_detail.txt, review.txt, review\_comment.txt, commit.txt, commit\_comment.txt, fork.txt, star.txt, user.txt.

如果使用additionSpider爬取churnSpider中爬取失败的链接，则可以将data\_crawl\_spider/ logs中的日志文件拷贝到data\_crawl\_spider/dataCrawler/logs中，然后运行get\_failed\_urls.py（运行前确保data\_crawl\_spider/dataCrawler/urls和data\_crawl\_spider/dataCrawler/urls\_simplified文件夹为空），即可自动提取日志文件中的链接并在urls\_simplified文件夹中生成对应的链接文件。

设置好链接文件后，可以通过运行data\_crawl\_spider/dataCrawler/additionSpider文件夹下的run.py启动爬虫。运行前检查以下内容：

* token的有效性。需要检查GitHub\_addition.py所在目录中的request\_header.py，以及middlewares.py中的token列表是否填写且有效，两个文件中的token列表保持一致即可。
* data\_crawl\_spider/dataCrawler/additionSpider/spiderLogs文件夹是否清空。如果您不希望在上一次运行的logs基础上继续添加日志，则需要在运行前手动清除日志文件。
* 清空data\_crawl\_spider/dataCrawler/additionSpider/job\_info/002目录下的内容。该目录是settings.py中设置的job\_dir，如果上一次运行中断后请求队列文件没有删除，可能会影响到此次运行。
* 检查GitHub\_addition.py中的start\_time和end\_time。确保该时间段涵盖所需爬取数据的时间。
* 如果运行的API链接排序方向为direction=asc, 则需要修改对应处理函数中page\_start\_time和page\_end\_time的定义（默认是direction=desc），否则会出现“data order error”的报错（日志）。

1. token相关注意事项

整个项目中多次出现token，data\_crawler\_spider中的token出现的文件如下表所示。一般情况下，这五个文件中的token列表应保持一致。

表6. token所在文件

|  |  |
| --- | --- |
| 文件路径 | token作用 |
| data\_crawl\_spider/churnSpider/churnSpider/spiders/request\_header.py | churnSpider爬取数据 |
| data\_crawl\_spider/churnSpider/churnSpider/middlewares.py | churnSpider爬取数据 |
| data\_Crawl\_spider/dataCrawler/additionSpider/additionSpider/spiders/  request\_header.py | additionSpider爬取数据 |
| data\_Crawl\_spider/dataCrawler/additionSpider/additionSpider/  middlewares.py | additionSpider爬取数据 |
| data\_Crawl\_spider/dataCrawler/request\_header.py | test\_rate\_limit.py使用 |

在使用token时，需要注意：

* token列表中的token应来自不同用户，同一用户的不同token效果相同，无法起到扩充rate limit的作用；
* 表6所示文件中参数GitHub\_token\_number表示token列表中token的个数，应随token个数进行调整；
* 提交代码到公开仓库时，需要将表6中每个文件中的token删除，否则token会被自动回收。

1. 日志和链接目录说明

项目中多次出现logs、urls等文件夹，分别用于存放/生成日志文件和链接文件等，表7是所有日志和链接目录的说明。

表7. 日志和链接目录说明

|  |  |
| --- | --- |
| 路径 | 作用 |
| data\_crawl\_spider/logs | 存放churnSpider运行生成的日志文件 |
| data\_crawl\_spider/dataCrawler/logs | 一般拷贝自data\_crawl\_spider/logs，用于get\_failed\_urls运行 |
| data\_crawl\_spider/dataCrawler/additionSpider/  spiderLogs | 存放additionSpider运行生成的日志文件 |
| data\_crawl\_spider/dataCrawler/urls | 存放get\_failed\_urls运行生成的文件，即提取自日志的链接文件（未消除重复） |
| data\_crawl\_spider/dataCrawler/urls\_simplified | 存放get\_failed\_urls运行生成的文件，即提取自日志的链接文件(消除重复)；该文件夹的文件用于additionSpider运行 |

1. API页限制问题解决方法

前文提到GitHub限制某些API的访问页数，对于issue comment数据和star数据的某一种排序方式最多可访问40000条内容。然而可以通过调整API链接中的参数来实现issue comment数据的完整爬取（star数据由于缺少排序功能，无法全部爬取）。方法如下：

利用churnSpider爬取仓库的issue comment数据时，最多可以爬取400页（每页100个），此时的排序方式为sort=created&direction=desc，爬到的是最新的40000条issue comment数据；改变排序方式，改为sort=created&direction=asc，并使用additionSpider爬取，此时又可爬取最早的40000条数据；如果前两种方式无法涵盖所有的issue comment数据，可以使用API中的since参数，该参数可以过滤某一时间点以后有更新的issue comment，通过该参数再加上sort=created&direction=asc进行排序，使用additionSpider即可爬取中间缺少的数据。

需要注意的是，如果使用since参数，时间格式必须为YYYY-MM-DDTHH:MM:SSZ；另外通过时间进行数据筛选时服务器响应速度很慢，经常会出现502的报错，应增加additionSpider的RETRY\_TIMES。

1. 代码改进方向

由于编写时间有限，代码仍有很多不足，此处对代码可以改进的方向/细节进行列举，供后续修改。

1. 502问题的处理

churnSpider中缺少对502报错的处理。

1. 日志完善

churnSpider中可以添加DUBUG级别的日志，用于记录已成功爬取的链接，方便检查数据完整性。

1. 代码目录结构调整

代码的目录结构不合理，如logs、urls等文件夹的分布等，需要进一步调整。

1. 自动清空日志/链接文件夹

可以添加运行前自动清空日志/链接文件夹的功能。

1. 爬虫的暂停和恢复

churnSpider暂不支持爬虫的暂停和恢复。