scrapy redis

scrapy redis 简介

Scrapy-redis 是为了更方便地实现 Scrapy 分布式爬取，而提供了一些以 redis 为基础的组件(仅有组件)。主体还是是 redis 和 scrapy 两个库，Scrapy-redis 像胶水一样，把这两个插件粘结了起来。

特点：

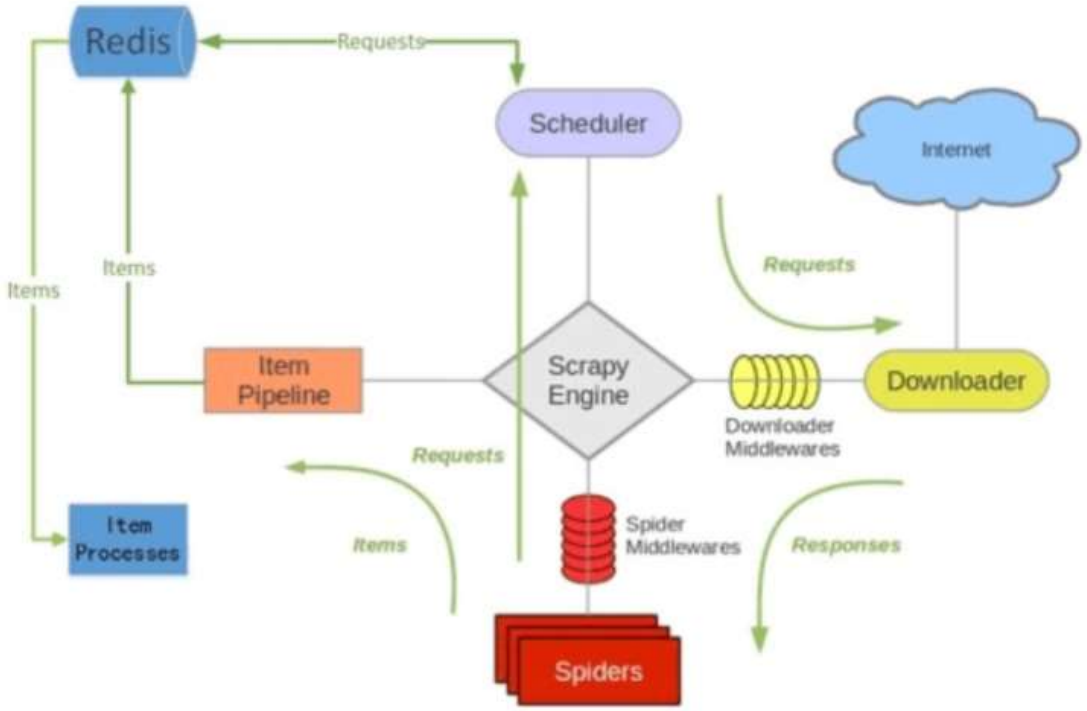
能实现分布式爬取

可实现去重

持续性爬取，可实现增量式爬虫

遵守 Rule 规则，可以实现深度爬虫

scrapy-redis 架构



scrapy-redis 在 scrapy 的架构上增加了 redis，基于 redis 的特性拓展了如下组件：

- **Scheduler**

Scheduler 负责对新的 request 进行入列和出列的操作。

Scrapy 本身不支持爬虫分布式，多个 spider 不能共享待爬取队列 Scrapy queue，**scrapy-redis 把 Scrapy queue 换成 redis 数据库，用同一个 redis-server 存放要爬取的 request，便能让多个 spider 去同一个数据库里读取。**

- **Duplication Filter**

Duplication Filter 利用了 redis 的 set 不重复的特性实现去重

scrapy-redis 调度器从引擎接受 request，将 request 的指纹存入redis 的 set 检查是否重复，并将不重复的 request push 写入redis 的 request queue。

引擎请求 request(Spider 发出的）时，调度器从 redis 的 request queue 队列里里根据优先级 pop 出一个 request 返回给引擎，引擎将此 request 发给 spider 处理。

- **Item Pipeline**

引擎将爬取到的 Item 传给 Item Pipeline，scrapy-redis 的 Item Pipeline 将爬取到的 Item 存入redis 的 items queue。

- **Base Spider**

不在使用 scrapy 原有的 Spider 类，重写的 RedisSpider 继承了 Spider 和 RedisMixin 这两个类，RedisMixin 是用来从 redis 读取 url 的类。

当我们生成一个 Spider 继承 RedisSpider 时，调用 setup_redis 函数，这个函数会去连接 redis 数据库，然后会设置 signals(信号)：

一个是当 spider 空闲时候的 signal，会调用 spider_idle 函数，这个函数调用 schedule_next_request 函数，保证 spider 是一直活着的状态，并且抛出 DontCloseSpider 异常。

一个是当抓到一个 item 时的 signal，会调用 item_scraped 函数，这个函数会调用 schedule_next_request 函数，获取下一个 request。

scrapy 与 scrapy-redis 区别

scrapy-redis 安装

```
pip install scrapy-redis
```

源码下载：https://github.com/rolando/scrapy-redis，其中包含的有官方的案例

Redis 复习

Redis 是一个开源的，内存数据库，它可以用作数据库、缓存和消息中间件。它支持多种类型的数据结构：字符串，哈希，列表，集合，有序集合等

文档：http://www.redis.cn/commands.html

远程连接 redis 数据库：

redis-cli -h<hostname> -p<port>

redis 操作

select 1 切换到 db1，默认 db0

keys * 查看所有的 redis 键

type '键' 查看键的数据类型

flushdb 清空当前 db

flushall 清空所有 db

列表

LPUSH mylist "world" 想 mylist 从左边添加一个值

LRANGE mylist 0 -1 返回 mylist 中所有的值

LLEN mylist 返回 mylist 的长度

set

SADD myset "Hello" 往 set 中添加数据

SMEMBERS myset 获取 myset 中所有的元素

SCARD myset 获取元素数量

zset

向 myzset 中添加一个值和分数，如果存在，就更新分数，分数可以相同

ZADD myzset 1 "one"

ZADD myzset 2 "two" 3 "three"

ZRANGE myzset 0 -1 HITHSCORES 遍历 myzset

ZCARD myzset 返回 myzset 中元素数量

ubuntu 下 redis 安装

官网下载：
https://redis.io/download

解压：
tar zxvf redis-4.0.8.tar.gz

会在当前目录下生成文件夹 redis-4.0.8，我把它移动到了/usr/redis 目录下：
cd /usr/local
sudo mkdir redis
sudo mv /home/tom/桌面/software/redis-4.0.8/* ./redis

进入 redis 目录
cd /usr/local/redis/

如果没有安装 gcc，需要先安装：
sudo apt-get install gcc

生成安装：
sudo make
sudo make install

复制 redis.conf 配置文件：

sudo mkdir /etc/redis

sudo cp /usr/local/redis/redis.conf /etc/redis/

一般配置文件都放在/etc/目录下

打开配置文件 redis.conf

`# bind 127.0.0.1` 注释掉

`protected-mode no` 去掉保护模式

启动服务端的时候没有带上配置文件

```
redis-server redis.conf
```

描述

一台电脑作为安装 redis，能够被远端访问，可以用 ubuntu 系统

其它电脑部署 scrapy-redis 来进行分布式抓取同一个网站

爬虫运行时会把提取到的 url 封装成 request 放到 redis 中的 “dmoz:**requests**”

从该数据库中提取 request 后下载网页，再把网页的内容发送回 redis 中“dmoz:**items**”

redis 中的“dmoz:requests”数据库为空，爬取结束

reids 中“dmoz:dupefilter”用来存储抓取过的 url 的指纹（使用哈希函数将 url 运算后的结果），是防止重复抓取的

dmoz:request:待爬取 request 对象

dmoz:items 爬取保存的条目

dmoz:dupefilter：抓取过的 url 的指纹

注：dmoz：爬虫的名称

分布式设置

在 settings.py 文件的最后增加如下内容：

```
REDIS_HOST = “192.168.10.123” #主机名
REDIS_PORT = 6379 #端口号
REDIS_PARAMS = {
  ‘password’ : ‘123’ ,
}
```

USER_AGENT 设置

USER_AGENT = 'scrapy-redis (+https://github.com/rolando/scrapy-redis)'

PIPELINES 存储设置

```
ITEM_PIPELINES = {
    #'example.pipelines.ExamplePipeline': 300,
    'scrapy_redis.pipelines.RedisPipeline': 400,
}
```

去重组件设置

DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"

调度器组件设置

SCHEDULER = "scrapy_redis.scheduler.Scheduler"

持续化存储设置

SCHEDULER_PERSIST = True

Log 设置

LOG_LEVEL = 'DEBUG'

延迟

DOWNLOAD_DELAY = 1

编码

REDIS_ENCODING = "utf-8" # redis 编码类型默认：'utf-8'

说明：# REDIS_PARAMS 连接参数默认：REDIS_PARAMS = {'socket_timeout': 30,'socket_connect_timeout': 30,'retry_on_timeout': True,'encoding': REDIS_ENCODING,})

数据存储

items 数据直接存储在 Redis 数据库中，这个功能已经由 scrapy-redis 自行实现。除非单独做额外处理(比如直接存入本地数据库等)，否则不用编写 pipelines.py 代码。

爬虫源码分析

1、dmoz (class DmozSpider(CrawlSpider))

特点

- 继承的是 CrawlSpider
- 需要设置 Rule 规则，以及 callback 不能写 parse()方法。
- 它是用来说明 Redis 的持续性，可实现增量式爬虫

当我们第一次运行 dmoz 爬虫，然后 Ctrl + C 停掉之后，再运行 dmoz 爬虫，之前的爬取记录是保留在 Redis 里的。

源码 DmozSpider

```
from scrapy.linkextractors import LinkExtractor
from scrapy.spiders import CrawlSpider, Rule


class DmozSpider(CrawlSpider):
    """Follow categories and extract links."""
    name = 'dmoz'
    allowed_domains = ['dmoz.org']
    start_urls = ['http://www.dmoz.org/']

    rules = [
        Rule(LinkExtractor(
            restrict_css=('.top-cat', '.sub-cat', '.cat-item')
        ), callback='parse_directory', follow=True),
    ]

    def parse_directory(self, response):
        for div in response.css('.title-and-desc'):
```

```
        yield {
            'name': div.css('.site-title::text').extract_first(),
            'description': div.css('.site-descr::text').extract_first().strip(),
            'link': div.css('a::attr(href)').extract_first(),
        }
```

执行方式

scrapy crawl dmoz

2、myspider_redis (class MySpider(RedisSpider))

特点

- 继承了 RedisSpider， 支持分布式抓取
- 需要写 parse 函数。
- 不需要写 start_urls 了，取而代之指定 redis_key，scrapy-redis 将 key 从 Redis 里 pop 出来，成为请求的 url 地址。
- 根据指定的格式，start_urls 将在 Master 端的 redis-cli 里 lpush 到 Redis 数据库里，RedisSpider 将在数据库里获取 start_urls。

参考格式：redis_key = 'myspider:start_urls'

- 不需要写 allowd_domains

源码 myspider_redis.py

```
from scrapy_redis.spiders import RedisSpider


class MySpider(RedisSpider):
    """Spider that reads urls from redis queue (myspider:start_urls)."""
```

```
    name = 'myspider_redis'

    # 注意 redis-key 的格式：
    redis_key = 'myspider:start_urls'

    # 可选：等效于 allowd_domains()，__init__方法按规定格式写，使用时只需要修改 super()里的类名参数即可
    def __init__(self, *args, **kwargs):
        # Dynamically define the allowed domains list.
        domain = kwargs.pop('domain', '')
        self.allowed_domains = filter(None, domain.split(','))

        # 修改这里的类名为当前类名
        super(MySpider, self).__init__(*args, **kwargs)

    def parse(self, response):
        return {
            'name': response.css('title::text').extract_first(),
            'url': response.url,
        }
```

执行方式

✧ 通过 runspider 方法执行爬虫的 py 文件（也可以分次执行多条），爬虫（们）将处于等待准备状态：

```
scrapy runspider myspider_redis.py
```

✧ 在 Master 端的 redis-cli 输入 push 指令，参考格式：

```
$redis > lpush myspider:start_urls http://dmoztools.net/
```

Slaver 端爬虫获取到请求，开始爬取。

3、mycrawler_redis (class MyCrawler(RedisCrawlSpider))

特点

- 爬虫继承了 RedisCrawlSpider
- 能够支持分布式的抓取
- 因为采用的是 crawlSpider，所以需要遵守 Rule 规则，可以实现深度爬虫
- callback 不能设为 parse()方法。
- 不需要写 start_urls 了，取而代之的是 redis_key
- scrapy-redis 将 key 从 Redis 里 pop 出来，成为请求的 url 地址。

源码 mycrawler_redis.py

```
from scrapy.spiders import Rule
from scrapy.linkextractors import LinkExtractor

from scrapy_redis.spiders import RedisCrawlSpider


class MyCrawler(RedisCrawlSpider):
    """Spider that reads urls from redis queue (myspider:start_urls)."""
    name = 'mycrawler_redis'
    redis_key = 'mycrawler:start_urls'

    rules = (
        # follow all links
        Rule(LinkExtractor(), callback='parse_page', follow=True),
    )

    # __init__方法必须按规定写，使用时只需要修改 super()里的类名参数即可
    def __init__(self, *args, **kwargs):
        # Dynamically define the allowed domains list.
        domain = kwargs.pop('domain', '')
        self.allowed_domains = filter(None, domain.split(','))
```

```
        # 修改这里的类名为当前类名
        super(MyCrawler, self).__init__(*args, **kwargs)

    def parse_page(self, response):
        return {
            'name': response.css('title::text').extract_first(),
            'url': response.url,
        }
```

执行方式

✧ 通过 runspider 方法执行爬虫的 py 文件（也可以分次执行多条），爬虫（们）将处于等待准备状态：

```
scrapy runspider mycrawler_redis.py
```

✧ 在 Master 端的 redis-cli 输入 push 指令，参考格式：

```
$redis > lpush mycrawler:start_urls http://www.dmoz.org/
```

爬虫获取 url，开始执行。

附录

runspider 和 crawl 命令

在未创建项目的情况下，运行一个编写在 Python 文件中的 spider

```
scrapy runspider myspider.py
```

项目命令，使用 spider 进行爬取

scrapy crawl <spider>

增量抓虫：指在具有一定量规模的网络页面集合的基础上，采用更新数据的方式选取已有集合中的过时网页进行抓取，以保证所抓取到的数据与真实网络数据足够接近。进行增量式抓取的前提是，系统已经抓取了足够数量的网络页面，并具有这些页面被抓取的时间信息。

深度爬虫：针对起始 url 地址进行数据采集，在响应数据中进行数据筛选得到需要进行数据采集的下一波 url 地址，并将 url 地址添加到数据采集队列中进行二次爬取..以此类推，一致到所有页面的数据全部采集完成即可完成深度数据采集，这里的深度指的就是 url 地址的检索深度。