SCRAPY_REDIS学习

第六部分课程概要

1. Scrapy_redis的基础概念
2. Scrapy_redis的流程
3. Scrapy_redis的使用

什么是scrapy_redis

Scrapy_redis ： Redis-based components for Scrapy.

Github地址：https://github.com/rmax/scrapy-redis

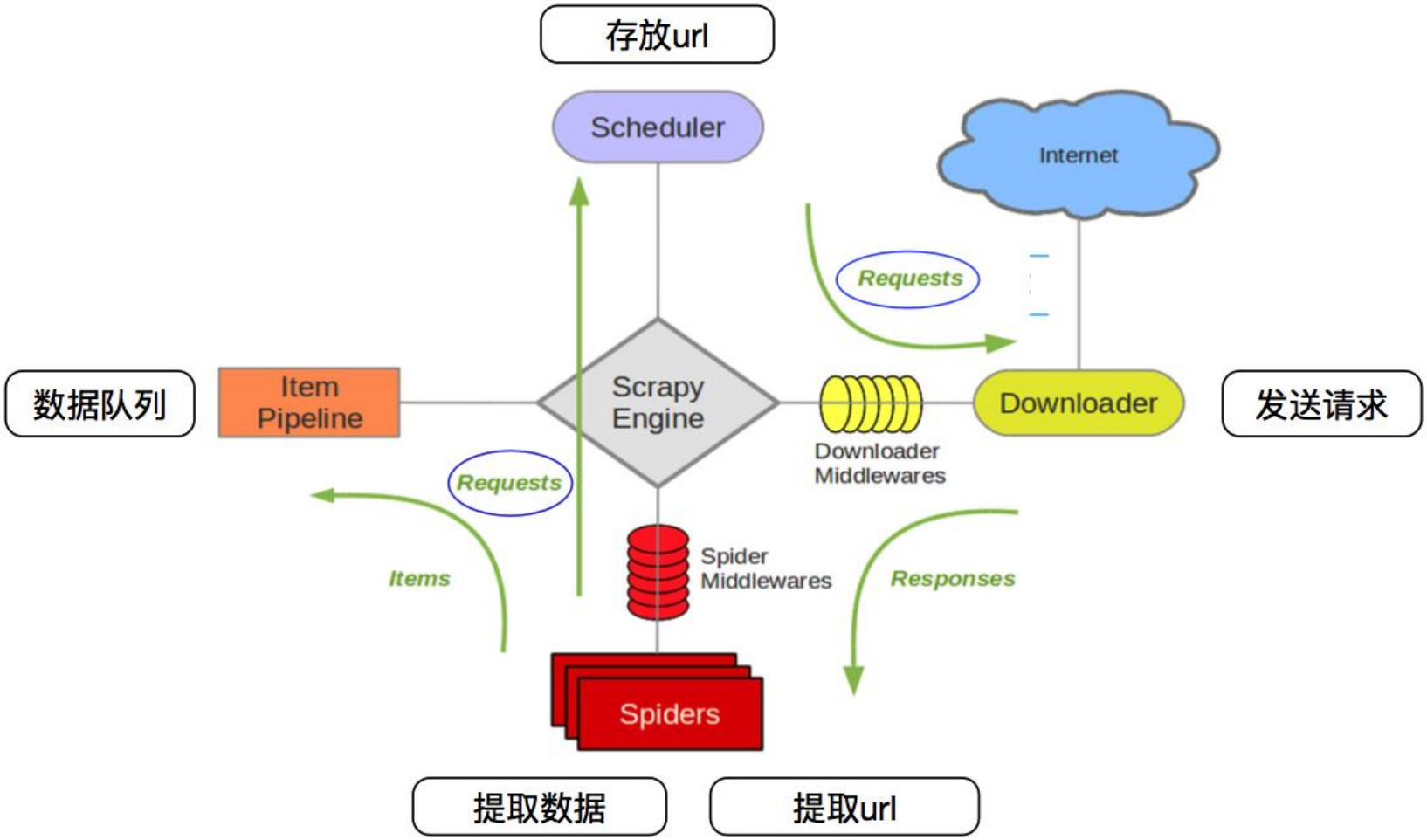
为什么要学习scrapy_redis

Scrapy_redis在scrapy的基础上实现了更多，更强大的功能，具体体现在：reqeust去重，爬虫持久化，和轻松实现分布式

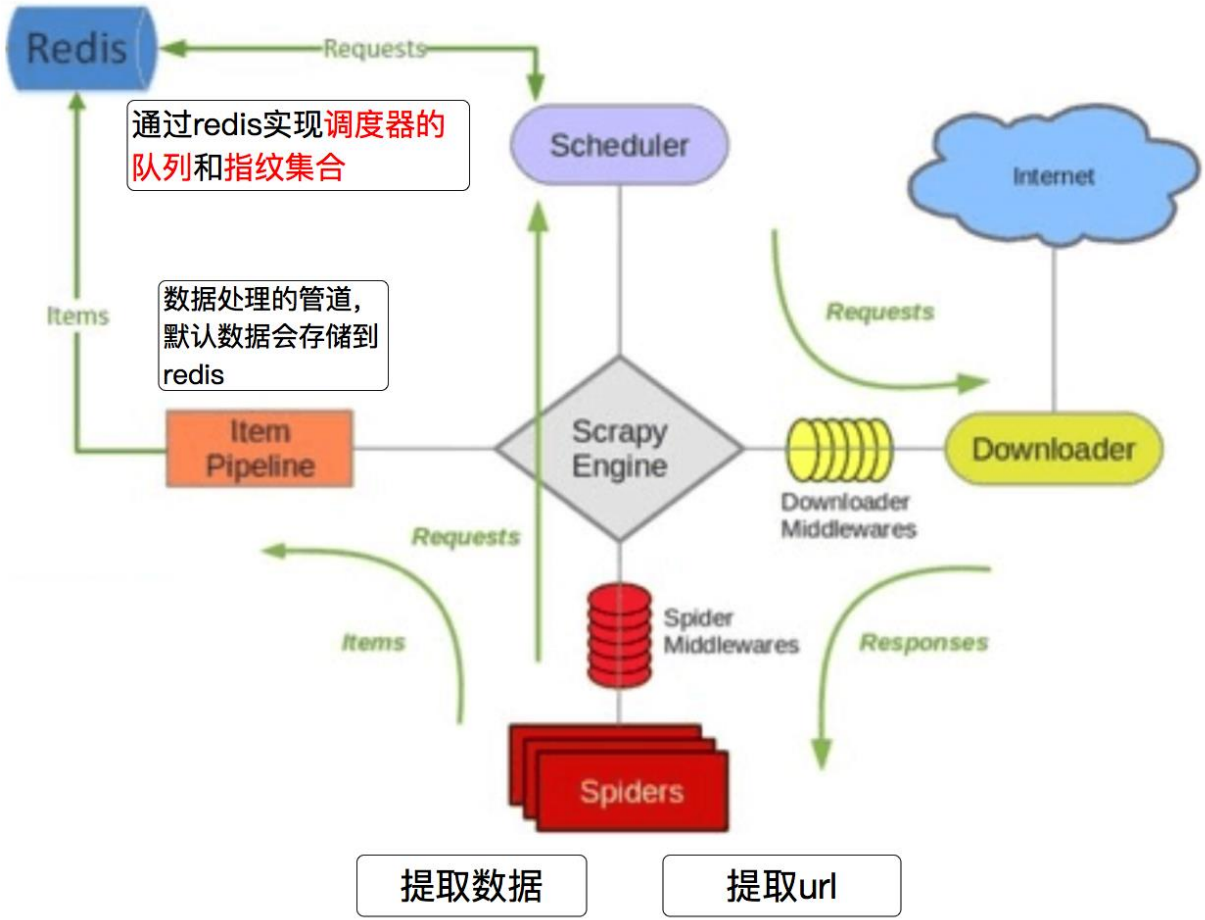
Scrapy_redis是工作流程

那么，scrapy_redis是如何帮助我们抓取数据的呢?

回顾scrapy的爬虫流程



scrapy_redis的爬虫流程



复习redis的使用

redis是什么：

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，如字符串，哈希，列表，集合，有序集合等

常用命令：

```
/etc/init.d/redis-server stop    ——>redis停止
/etc/init.d/redis-server start   ——>启动
/etc/init.d/redis-server restart  ——>重启
```

redis-cli -h <hostname> -p <port> ——>远程连接redis数据库

redis中：

0、select 1　　——>切换到db1，默认在db0
1、keys *　　——>查看所有的redis键
2、type "键"　　——>查看键的数据类型
3、flushdb　　——>清空当前db
4、flushall　　——>清空所有db

中文文档 http://www.redis.cn/commands.html

复习redis的使用

列表：

```
LPUSH mylist "world"      ——>向mylist从左边添加一个值
LRANGE mylist 0 -1        ——>返回mylist中所有的值
LLEN mylis                ——>返回mylist的长度
```

set：

```
redis> SADD myset "Hello"      ——>往set中添加数据
redis> SMEMBERS myset          ——>获取myset中所有的元素
redis> SCARD myset             ——>scrad 获取数量
```

zset：

```
redis> ZADD myzset 1 "one"
(integer) 1
redis> ZADD myzset 2 "two" 3 "three"
(integer) 2
redis> ZRANGE myzset 0 -1 WITHSCORES
1) "one"
2) "1"
3) "two"
4) "2"
5) "three"
6) "3"
redis> ZCARD myzset
```

zadd 向一个zset中添加一个值和分数，如果存在值，就更新分数，分数可以相同
zrange 遍历myzeset
zcard 返回zset中元素的数量

Scrapy_redis使用

1、clone github scrapy-redis源码文件

git clone https://github.com/rolando/scrapy-redis.git

2、研究项目自带的三个demo

mv scrapy-redis/example-project ~/scrapyredis-project

Scrapy_redis之domz

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


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

    rules = [
        Rule(LinkExtractor(    ——>定义了一个url的提取规则，满足的交给callback函数处理
            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 {  ——>yield给引擎
                '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(),
            }
```

domz这个部分和我们自己写的crawlspider没有任何区别

Scrapy_redis之domz

```
SPIDER_MODULES = ['example.spiders']
NEWSPIDER_MODULE = 'example.spiders'

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

DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"    ->指定那个去重方法给request对象去重
SCHEDULER = "scrapy_redis.scheduler.Scheduler"      ——>指定scheduler队列
SCHEDULER_PERSIST = True    ——>队列中的内容是否持久保存，为False的时候还在关闭redis的时候清空redis
#SCHEDULER_QUEUE_CLASS = "scrapy_redis.queue.SpiderPriorityQueue"
#SCHEDULER_QUEUE_CLASS = "scrapy_redis.queue.SpiderQueue"
#SCHEDULER_QUEUE_CLASS = "scrapy_redis.queue.SpiderStack"

ITEM_PIPELINES = {
    'example.pipelines.ExamplePipeline': 300,
    'scrapy_redis.pipelines.RedisPipeline': 400,    ——>scrapy_redis实现的items保存到redis的pipeline
}

LOG_LEVEL = 'DEBUG'

DOWNLOAD_DELAY = 0.3

REDIS_URL = 'redis://192.168.207.130:6379'     ——>指定redis的地址

# REDIS_HOST = "192.168.207.134"          ——>redis的地址可以写成如下形式
# REDIS_PORT = 6379
```

Scrapy_redis之domz

我们执行domz的爬虫，会发现redis中多了一下三个键：

```
127.0.0.1:6379> keys *
1) "dmoz:requests"
2) "dmoz:dupefilter"
3) "dmoz:items"
```

```
2959 )"ph\x94K\x02u\x8c\t_encoding\x9
4\x8c\x05utf-8\x94\x8c\bpriority\x94K\x00\x8c\x
0bdont_filter\x94\x89\x8c\x05flags\x94]\x94u."
3960) "0"
```

Scheduler队列，存放的待请求的request对象，获取的过程是pop操作，即获取一个会去除一个

```
92) "{\"name\": \"F\\u00f8royski …….,
\"spider\": \"dmoz\"}"
93) "{\"name\": \"Nummar.fo \……..
\"spider\": \"dmoz\"}"
```

存放的获取到的item信息，在pipeline中开启RedisPipeline才会存入

```
2185) "9d2a4421f267c672b98093956e864032b0ef6f1f"
2186) "df7ccfc65d1a5818c70a191b058c3fcaa148251c"
2187) "9b8a2c12e911cb8ae3ad00e8455c94616e8e1a99"
2188) "d5a8ef8868b67fe9e3ed6848c1ad305eebc88e5a"
2189) "2d61df39246bf24383b2cb7c516690b93fa97fdd"
2190) "713e31b85b006af1df53e0c489bbe580e6801d4c"
```

指纹集合，存放的是已经进入scheduler队列的request对象的指纹，指纹默认由请求方法，url和请求体组成

Scrapy_redis之domz

我们可以尝试在setting中关闭redispipeline，

观察redis中三个键的存储数据量的变化

Scrapy_redis之domz

变化结果：

dmoz:requests 有变化(变多或者变少或者不变)

dmoz:dupefilter 变多

dmoz:items 不变

变化结果分析:

redispipeline中仅仅实现了item数据存储到redis的过程，我们可以新建一个pipeline（或者修改默认的ExamplePipeline），让数据存储到任意地方

Scrapy_redis之domz

那么问题来了：以上的这些功能scrapy_redis都是如何实现的呢?

Scrapy_redis之RedisPipeline

RedisPipeline代码片段

```
def process_item(self, item, spider):  ——>使用了process_item的方法，实现数据的保存
    return deferToThread(self._process_item, item, spider)
          ——>调用的一个异步线程去处理这个item

def _process_item(self, item, spider):
    key = self.item_key(item, spider)
    data = self.serialize(item)
    self.server.rpush(key, data)  ——>向dmoz:items中添加item
    return item
```

Scrapy_redis之RFPDupeFilter

```
def request_seen(self, request):  —>判断requests对象是否已经存在
    fp = self.request_fingerprint(request)
    added = self.server.sadd(self.key, fp)   ->添加到dupefilter中
    return added == 0   —>返回0表示添加失败，即已经存在，否则表示不存在

def request_fingerprint(self, request):
    return request_fingerprint(request)


def request_fingerprint(request, include_headers=None):
    if include_headers:
        include_headers = tuple(to_bytes(h.lower())
                                for h in sorted(include_headers))
    cache = _fingerprint_cache.setdefault(request, {})
    if include_headers not in cache:
        fp = hashlib.sha1()   —>sha1加密
        fp.update(to_bytes(request.method)) —>请求方法
        fp.update(to_bytes(canonicalize_url(request.url)))  ->请求地址
        fp.update(request.body or b'')   —>请求体，post请求才会有
        if include_headers:   —>添加请求头，默认不添加请求头(因为header的cookies中含有
                              session id，这在不同的网站中是随机的，会给sha1的计算结果带来误差)
            for hdr in include_headers:
                if hdr in request.headers:
                    fp.update(hdr)
                    for v in request.headers.getlist(hdr):
                        fp.update(v)
        cache[include_headers] = fp.hexdigest()   —>返回加密之后的16进制
    return cache[include_headers]
```

Scrapy_redis之Scheduler

Scheduler代码片段：

```
def close(self, reason):
    if not self.persist:    —>如果在setting中设置为不持久，那么在退出的时候回清空
        self.flush()

def flush(self):
    self.df.clear()  ——>指的是存放dupefilter的redis
    self.queue.clear()    —>指的是存放requests的redis

def enqueue_request(self, request):
    if not request.dont_filter and self.df.request_seen(request):  —>不能加入待爬队列的条件
            ——>当前url需要(经过allow_domain)过滤并且request不存在dp的时候
            ——>由此：对于像百度贴吧这种页面内容会更新的网址，可以设置dont_filter为True让其能够被反复抓取
        self.df.log(request, self.spider)
        return False
    if self.stats:
        self.stats.inc_value('scheduler/enqueued/redis', spider=self.spider)
    self.queue.push(request)
    return True
```

Scrapy_redis之dmoz

通过以上知识点的学习，我们会发现：

domz相比于之前的spider多了持久化和request去重的功能

在之后的爬虫中，我们可以模仿domz的用法，使用scrapy_redis实现相同的功能

注意：setting中的配置都是可以自己设定的，意味着我们的可以重写去重和调度器的方法，包括是否要把数据存储到redis(pipeline)

动手练习

需求：抓取京东图书的信息

目标：抓取京东图书包含图书的名字、封面图片地址、图书url地址、作者、出版社、出版时间、价格、图书所属大分类、图书所属小的分类、分类的url地址

url：https://book.jd.com/booksort.html

Scrapy_redis之RedisSpider

```
class MySpider(RedisSpider):
    name = 'myspider_redis'  ——>指定爬虫名
    redis_key = 'myspider:start_urls'   ——>指定redis中start_urls的键，
            ——>启动的时候只需要往对应的键总存入url地址，不同位置的爬虫就会来获取该url
            ——>所以启动爬虫的命令分类两个步骤：
                    ①scrapy crawl myspider_redis(或者scrapy runspider myspider_redis)  让爬虫就绪
                    ②在redis中输入 lpush myspider:start_urls "http://dmoztools.net/" 让爬虫从这个url开始爬取
    allow_doamin = ["dmoztools.net"]  ——>手动指定allow_domain

    def __init__(self, *args, **kwargs):  ——>动态的设置allow_domain，一般不需要，直接手动指定即可
        domain = kwargs.pop('domain', '')
        self.allowed_domains = filter(None, domain.split(','))
        super(MySpider, self).__init__(*args, **kwargs)

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

动手练习

需求：抓取当当图书的信息

目标：抓取当当图书又有图书的名字、封面图片地址、图书url地址、作者、出版社、出版时间、价格、图书所属大分类、图书所属小的分类、分类的url地址

url：http://book.dangdang.com/

Scrapy_redis之RedisCrawlSpider

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

class MyCrawler(RedisCrawlSpider):
    name = 'mycrawler_redis'    ——>爬虫名字
    redis_key = 'mycrawler:start_urls'  ->start_url的redis的键
    allow_domains = ["dmoztools.net"]  ->手动制定all_domains
    rules = (     ——>和crawl一样，指定url的过滤规则
        Rule(LinkExtractor(), callback='parse_page', follow=True)
    )

    def __init__(self, *args, **kwargs):
                    ->动态生成all_domain，不是必须的
        domain = kwargs.pop('domain', '')
        self.allowed_domains = filter(None, domain.split(','))
        super(MyCrawler, self).__init__(*args, **kwargs)

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

动手练习

需求：抓取亚马逊图书的信息

目标：抓取亚马逊图书又有图书的名字、封面图片地址、图书url地址、作者、出版社、出版时间、价格、图书所属大分类、图书所属小的分类、分类的url地址

url：https://www.amazon.cn/%E5%9B%BE%E4%B9%A6/b/ref=sd_allcat_books_l1?ie=UTF8&node=658390051

Crontab爬虫定时执行

安装:apt-get install cron(服务器环境下默认安装的有)
使用:crontab -e 进入编辑页面(第一次会让你选择编辑器)
crontab -l 查看当前的定时任务
编辑:

分	小时	日	月	星期	命令
0-59	0-23	1-31	1-12	0-6	command

例子:

```
30    7     8       *    *    ls          指定每月8号的7：30分执行ls命令
*/15  *     *       *    *    ls
                每15分钟执行一次ls命令 [即每个小时的第0 15 30 45 60分钟执行ls命令 ]
0     */2   *       *    *    ls          每隔两个小时执行一次ls
```

注意点:
1.星期中0表示周日
2.每隔两个小时的时候前面的不能为*,为*表示分钟都会执行

Crontab爬虫定时执行

执行python程序:

1.先把python的执行命令写入.sh脚本
2.给.sh脚本添加可执行权限 → chmod +x myspider.sh
3.把.sh程序写入crontab配置文件中

一个myspider.sh的例子

```
#!/bin/sh
cd `dirname $0` || exit 1
python ./main.py >>run.log 2>&1
```

#!/bin/sh → 使用/bin/sh来执行下面的内容

cd `dirname $0` || exit 1 → cd到当前目录,失败则退出,dirname上面不是引号

python ./main.py >>run.log 2>&1 → 把屏幕输出的内容从定向到run.log,同时把标准错误作物标准输出一起输出到run.log

对应crontab中的编写(注意写绝对路径)

```
#crontab
0 6 * * * /home/ubuntu/***/myspider.sh >> /home/ubuntu/***/run.log 2>&1
```

/home/ubuntu/***/myspider.sh >> /home/ubuntu/***/run.log 2>&1 → 把屏幕输出的内容从定向到run.log,同时把标准错误作物标准输出一起输出到run.log

Thank You!

改变中国 IT 教育，我们正在行动