|  |
| --- |
| 分布式数据库课程报告  分布式网络爬虫设计与实现 |

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 主要工作 |
| 涂浩新 | 31717019 | 代码实现，报告撰写 |
| 赵松辉 | 21717055 | 代码实现，报告撰写 |

2017年12月

目 录

[第1章 引言 3](#_Toc502433340)

[第2章 Redis基础 4](#_Toc502433341)

[2.1 Redis简介 4](#_Toc502433342)

[2.2 在Linux下安装Redis 4](#_Toc502433343)

[2.3 Scrapy集成Redis 4](#_Toc502433344)

[第3章 MongoDB集群 6](#_Toc502433345)

[3.1 简介 6](#_Toc502433346)

[3.2 集群的搭建 6](#_Toc502433347)

[第4章 项目设计 8](#_Toc502433348)

[4.1 设计方案 8](#_Toc502433349)

[4.2 scrapy-redis的源代码分析 9](#_Toc502433350)

[第5章 项目实现 12](#_Toc502433351)

[5.1 实验环境 12](#_Toc502433352)

[5.2 实验步骤 12](#_Toc502433353)

[5.2.1 创建爬虫项目 12](#_Toc502433354)

[5.2.2 定义数据结构——/items.py 12](#_Toc502433355)

[5.2.3 实现爬虫——/spiders/yunqi\_qq\_com.py 12](#_Toc502433356)

[5.2.4 保存数据——/pipelines.py 13](#_Toc502433357)

[5.3 实验结果数据部分展示 13](#_Toc502433358)

[第6章 总结与体会 14](#_Toc502433359)

分布式网络爬虫设计与实现

第1章 引言

分布式是大数据时代流行的一个词，比如常见的分布式计算，分布式存储，分布式爬虫等等。

分布式爬虫，从字面的意义上来说是集群爬虫，就是将爬虫任务分配给多台机器同时进行处理，与之对应的是单机爬虫，单点部署，单点操作。分布式爬虫相当于将多个单机联系起来形成一个整体来完成工作，目的就是提高可用性、稳定性和性能，因为单机操作有CPU、IO和带宽等多重限制。

打造分布式爬虫的关键是调度，本设计采用消息队列的方式，使用scrapy-redis组件作为消息队列，从而实现分布式。

本报告的内容组织如下：第二节介绍Redis基础，包括Redis的安装和使用；第三节说明MongoDB集群的基本操作；分布式网络爬虫设计方案及源码分析是第四部分要陈述的内容；第五部分展示实验环境和最终实现的设计效果；第六部分是总结与展望。

第2章 Redis基础

## 2.1 Redis简介

Redis 是一个使用ANSI标准的C语言编写的开源、支持网络、基于内存的Key-Value存储数据库。根据DB-Engines网站的数据显示，Redis是最流行的键值对数据库。

Redis的外围由一个键、值映射的字典构成。与其他非关系型数据库的主要不同在于：Redis中值的类型不仅限于字符串类型，还支持一些抽象的数据类型，如字符串列表、无序不重复的字符串集合、键和值都为字符串的哈希表等。值的类型决定了值本身支持的操作，Redis支持不同无序、有序的列表，无序、有序的集合间的交集、并集等高级服务器端原子操作。

Redis支持主从同步。数据可以从主服务器向任意数量的从服务器上同步，从服务器可以是关联其他从服务器的主服务器。这使得Redis可执行单层树复制。从盘可以有意无意的对数据进行写操作。由于完全实现了发布/订阅机制，使得从数据库在任何地方同步树时，可订阅一个频道并接收主服务器完整的消息发布记录。同步对读取操作的可扩展性和数据冗余很有帮助。

最值得一提的是Redis的性能。Redis通常将全部的数据存储在内存中。当数据依赖不再需要，Redis这种基于内存的性质，与在执行一个事务时将每个变化都写入硬盘的数据库系统相比就显得执行效率非常高。写与读操作速度没有明显差别。

本项目使用Redis来进行分布式爬虫的调度，并未用它存储数据，存储数据使用的是MongoDB集群。

## 2.2 在Linux下安装Redis

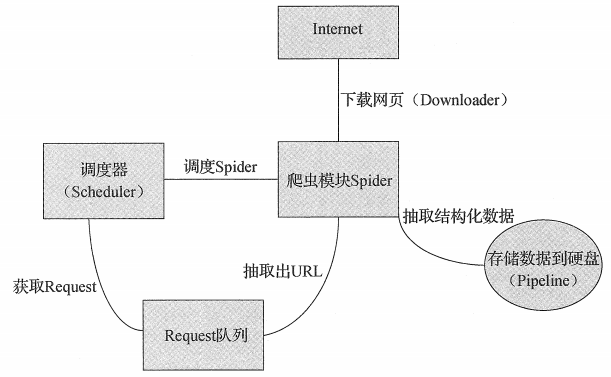
|  |
| --- |
| Linux命令 |
| $ wget http://download.Redis.io/releases/Redis-4.0.6.tar.gz  $ tar xzf Redis-4.0.6.tar.gz  $ cd Redis-4.0.6  $ make |

使用make命令编译成功后，即可使用Redis。进入src目录下，执行./Redis-server命令，启动Redis服务。Redis服务端的默认连接端口是6379。

## 2.3 Scrapy集成Redis

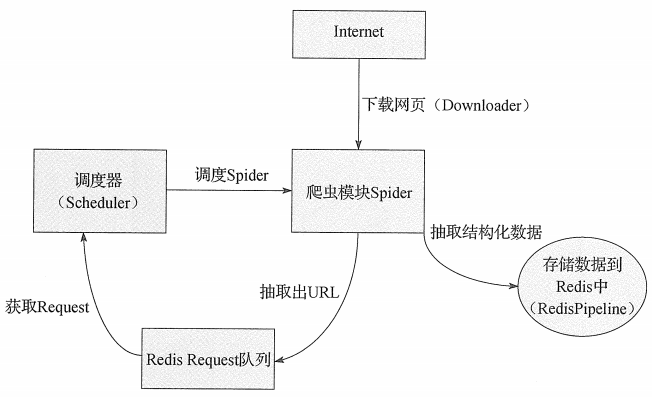
本项目使用开源软件包scrapy\_redis实现分布式爬虫机制。

Scrapy是使用Python开发的一个快速、高层次的屏幕抓取和web抓取框架，用于抓取web站点并从页面中提取结构化的数据。Scrapy用途广泛，可以用于数据挖掘、监测和自动化测试。Scrapy吸引人的地方在于它是一个框架，任何人都可以根据需求方便的修改。它也提供了多种类型爬虫的基类，如BaseSpider、sitemap爬虫等，最新版本又提供了web2.0爬虫的支持。Scrapy框架的工作流程如图2.1所示。



**图2.1 Scrapy框架的工作流程**

在Scrapy中，要想实现分布式，需要使用Redis作为消息队列，通过安装scrapy\_redis组件就可以实现。使用scrapy\_redis后的工作流程，如图2.2所示。前后发生的主要变化是Request队列放到了Redis中，这样多个单机就可以通过Redis获取Request，实现分布式，同时将要存储的结构化数据存到Redis队列中。



**图2.2 Redis Request队列**

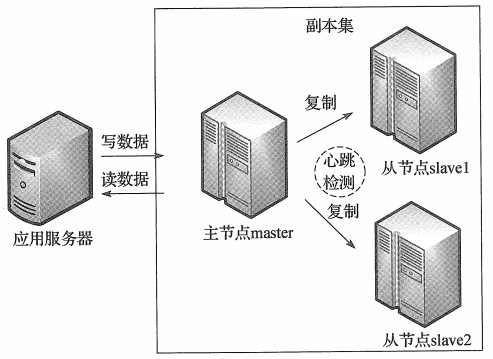
第3章 MongoDB集群

## 3.1 简介

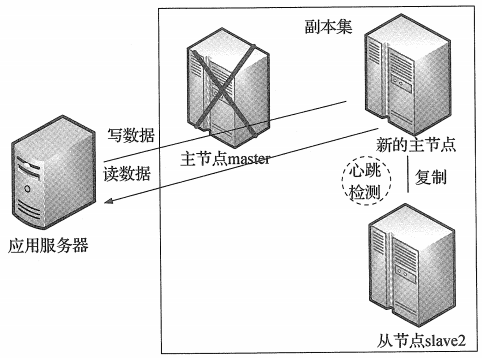
设置MongoDB集群，在很大程度上可以提高系统的稳定性。比如，在分布式爬虫抓取数据的过程中，如果MongoDB的存储服务器因故障而崩溃了，那么整个爬虫系统将立刻陷入瘫痪。为了避免这种情况，可以使用多个MongoDB存储节点，当主节点出现故障时，从属节点可以立刻补充进来，保持系统稳定运行。

## 3.2 集群的搭建

我们使用MongoDB副本集的形式来搭建主从集群，副本集的工作模式如图3.1所示。从图中可以看到，应用服务器与主节点之间进行读写操作，主节点将数据实时同步到从属节点中，主节点与从属节点之间通过心跳检测的方式进行沟通，判断是否存活。假如主节点突然出现故障，两个从属节点会通过仲裁的方式判断谁作为新的主节点，如图3.2所示。



**图3.1 副本集工作模式**



**图3.2 主节点自动切换**

我们采取在同一台主机上开启不同的端口，来实现三台机器的效果。选择localhost:20000作为主节点，localhost:30000和localhost:40000作为从属节点，并且在主机上建立三个不同的文件夹，作为数据库存储。每个文件夹中都创建一个空的data文件夹，作为三个节点的数据目录。接着，开启三个命令行窗口，用以启动三个不同的mongodb服务，不同窗口的输入命令如下所示：

|  |
| --- |
| **主节点master** |
| mongod –-dbpath d:\mongodb\master\data –-replSet repset –-port 20000 |
| **从属节点slave1** |
| mongod –-dbpath d:\mongodb\slave1\data –-replSet repset –-port 30000 |
| **从属节点slave2** |
| mongod –-dbpath d:\mongodb\slave2\data –-replSet repset –-port 40000 |

三个服务启动成功后，需要初始化副本集。任意登录其中一个服务，比如登录主节点，另外启动一个命令行窗口依次输入以下命令：

|  |
| --- |
| **主节点master** |
| > mongo  > use test  > config = { \_id:”repset”, members:[  {\_id:0, host:”127.0.0.1:20000”},  {\_id:1, host:”127.0.0.1:30000”},  {\_id:2, host:”127.0.0.1:40000”}]}  > rs.initiate(config) |

至此，MongoDB集群搭建完成。

第4章 项目设计

## 4.1 设计方案

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，和轻松实现分布式等等。

Scrapy-redis分布式策略如下：

假设有四台电脑：Windows 10、Mac OS X、Ubuntu 16.04、CentOS 7.2，任意一台电脑都可以作为 Master端 或 Slaver端，比如：

Master端(核心服务器) ：使用 Windows 10，搭建一个Redis数据库，不负责爬取，只负责url指纹判重、Request的分配，以及数据的存储。

Slaver端(爬虫程序执行端) ：使用 Mac OS X 、Ubuntu 16.04、CentOS 7.2，负责执行爬虫程序，运行过程中提交新的Request给Master

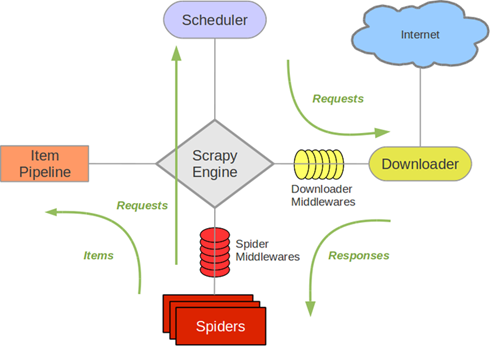
首先Slaver端从Master端拿任务（Request、url）进行数据抓取，Slaver抓取数据的同时，产生新任务的Request便提交给 Master 处理；

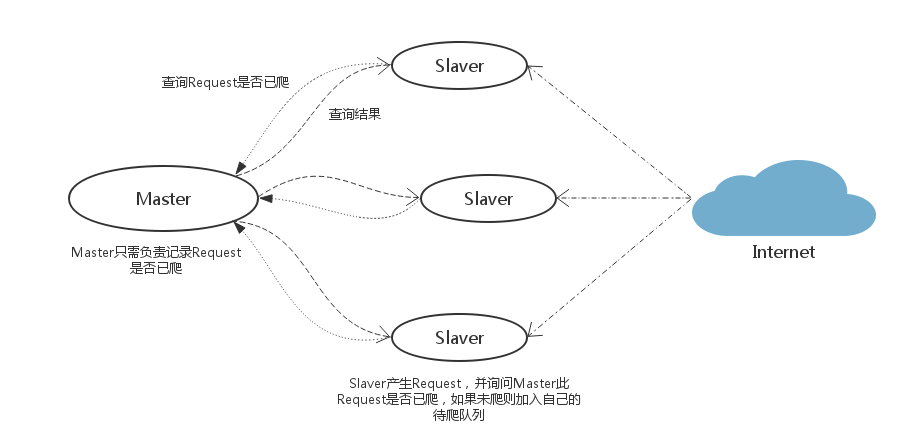
Master端只有一个Redis数据库，负责将未处理的Request去重和任务分配，将处理后的Request加入待爬队列，并且存储爬取的数据。

Scrapy-redis默认使用的就是这种策略，我们实现起来很简单，因为任务调度等工作Scrapy-redis都已经帮我们做好了，我们只需要继承RedisSpider、指定Redis\_key就行了。

缺点是，Scrapy-redis调度的任务是Request对象，里面信息量比较大（不仅包含url，还有callback函数、headers等信息），可能导致的结果就是会降低爬虫速度、而且会占用Redis大量的存储空间，所以如果要保证效率，那么就需要一定硬件水平。

具体流程如下图。





## 4.2 scrapy-redis的源代码分析

scrapy是python的一个非常好用的爬虫库，功能非常强大，但是当爬取的页面非常多时，无论是处理速度还是网络请求的并发数，单个主机的处理能力无法满足我们的需求，此时分布式爬虫的优势就显现出来。scrapy-redis就是结合了分布式数据库Redis，重写了scrapy一些比较关键的代码，将scrapy变成一个可以在多个主机上同时运行的分布式爬虫。

scrapy-redis是Github上的一个开源项目，可以直接下载到他的源代码：

https://github.com/rolando/scrapy-redis。

scrapy-redis的官方文档写的比较简洁，没有提及其运行原理，所以如果想全面的理解分布式爬虫的运行原理，需要研究scrapy-redis的源代码。scrapy-redis工程的主体还是Redis和scrapy两个库，工程本身实现的东西不是很多，这个工程把这两个插件连接到了起来。下面来简单介绍scrapy-redis主要的源代码文件实现的功能。

模块一：connect.py

connect模块引入了Redis模块，这个是Redis-python库的接口，用于通过python访问Redis数据库。这个文件主要是实现连接Redis数据库的功能（返回的是Redis库的Redis对象或者StrictRedis对象，这两个都是可以直接用来进行数据操作的对象）。这些连接接口在其他文件中经常被用到。其中，可以看到，要想连接到Redis数据库，和其他数据库差不多，需要一个IP地址、端口号、用户名密码（可选）和一个整形的数据库编号，同时还可以在scrapy工程的setting文件中配置套接字的超时时间、等待时间等。

模块二：dupefilters.py

这个模块重写了scrapy本身已经实现的request判重功能。因为本身scrapy单机跑的话，只需要读取内存中的request队列或者持久化的request队列（scrapy默认的持久化是JSON格式的文件，不是数据库）就能判断这次要发出的request url是否已经请求过或者正在调度（本地读就行了）。而分布式跑的话，就需要各个主机上的scheduler都连接同一个数据库的同一个request池来判断这次的请求是否是重复的了。

在这个文件中，通过继承BaseDupeFilter重写他的方法，实现了基于Redis的判重。根据源代码来看，scrapy-redis使用了scrapy本身的一个fingerprint接request\_fingerprint，根据scrapy文档所述，通过hash来判断两个url是否相同（相同的url会生成相同的hash结果），但是当两个url的地址相同，get型参数相同但是顺序不同时，也会生成相同的hash结果。所以scrapy-redis依旧使用url的fingerprint来判断request请求是否已经出现过。这个类通过连接Redis，使用一个key来向Redis的一个set中插入fingerprint，这个key对于同一种spider是相同的，Redis是一个key-value的数据库，如果key是相同的，访问到的值就是相同的，这里使用spider名字+DupeFilter的key就是为了在不同主机上的不同爬虫实例，只要属于同一种spider，就会访问到同一个set，而这个set就是他们的url判重池，如果返回值为0，说明该set中该fingerprint已经存在，则返回False，如果返回值为1，说明添加了一个fingerprint到set中，则说明这个request没有重复，于是返回True，还顺便把新fingerprint加入到数据库中了。

DupeFilter判重会在scheduler类中用到，每一个request在进入调度之前都要进行判重，如果重复就不需要参加调度，直接舍弃。

模块三：pipeline.py

pipeline文件实现了一个item pipeline类，和scrapy的item pipeline是同一个对象，通过从settings中拿到我们配置的REDIS\_ITEMS\_KEY作为key，把item串行化之后存入Redis数据库对应的value中，这个value可以看出是个list，我们的每个item是这个list中的一个结点，这个pipeline把提取出的item存起来，主要是为了方便我们延后处理数据。

模块四：scheduler.py

这个文件重写了scheduler类，用来代替scrapy.core.scheduler的原有调度器。其实对原有调度器的逻辑没有很大的改变，主要是使用了Redis作为数据存储的媒介，以达到各个爬虫之间的统一调度。

scheduler负责调度各个spider的request请求，scheduler初始化时，通过settings文件读取queue和dupefilters的类型（一般就用上边默认的），配置queue和dupefilters使用的key（一般就是spider name加上queue或者dupefilters，这样对于同一种spider的不同实例，就会使用相同的数据块了）。每当一个request要被调度时，enqueue\_request被调用，scheduler使用dupefilters来判断这个url是否重复，如果不重复，就添加到queue的容器中（先进先出，先进后出和优先级都可以，可以在settings中配置）。当调度完成时，next\_request被调用，scheduler就通过queue容器的接口，取出一个request，把他发送给相应的spider，让spider进行爬取工作。

同时，如果setting文件中配置了SCHEDULER\_PERSIST为True，那么在爬虫关闭的时候scheduler会调用自己的flush函数把Redis数据库中的判重和调度池全部清空，使得爬取进度完全丢失（但是item没有丢失，item数据在另一个键中储存）。如果设置SCHEDULER\_PERSIST为False，爬虫关闭后，判重池和调度池仍然存在于Redis数据库中，则再次开启爬虫时，可以接着上一次的进度继续爬取。

模块五：spider.py

spider的改动也不是很大，主要是通过connect接口，给spider绑定了spider\_idle信号，spider初始化时，通过setup\_redis函数初始化好和Redis的连接，之后通过next\_requests函数从Redis中取出strat url，使用的key是settings中REDIS\_START\_URLS\_AS\_SET定义的，spider使用少量的start url，可以发展出很多新的url，这些url会进入scheduler进行判重和调度。直到spider跑到调度池内没有url的时候，会触发spider\_idle信号，从而触发spider的next\_requests函数，再次从Redis的start url池中读取一些url。

第5章 项目实现

## 5.1 实验环境

本项目实现环境为：CentOS 6.9,Inter i5-3320m CPU 2.6GHz,scrapy 1.4.0,Redis 3.2.5,mongodb 3.0.6。由于实验室硬件环境有限，采取在一台主机上开启不同的端口，来实现三台机器的效果，选择127.0.0:27017作为主节点，127.0.0.1:27018和127.0.0.1:27019作为从节点，选择三个不同的文件夹，作为数据库的存储。

## 5.2 实验步骤

### 5.2.1 创建爬虫项目

在爬虫之前，首先应该对目标网站——云起书院（yunqi.qq.com）进行分析。我们的目的是抓取网站上小说的名称、作者、分类、状态、更新时间、字数、点击量、人气和推荐等数据。在网站图书列表中，包含小说的名称、作者、所属分类、状态（连载中或已完载）、最近更新时间、字数等信息；任意选择其中一个小说条目，进入小说的详情页，详情页包含小说的点击量、人气、推荐等数据。

对网站的结构分析完成后，就可以创建项目了。使用全局命令startproject创建项目；进入项目文件夹，使用scrapy提供的crawl模板创建一个名为yunqi.qq.com的Spider。命令如下所示：

|  |
| --- |
| **创建爬虫项目：yunqiCrawl** |
| scrapy startproject yunqiCrawl  cd yunqiCrawl  scrapy genspider –t crawl yunqi.qq.com yunqi.qq.com |

### 5.2.2 定义数据结构——/items.py

创建完项目之后，我们根据网站上小说的信息展示方式，在Python中使用类（class）定义了两个Item，分别是小说的基本信息ListItem（即图书列表中展示的信息）和小说的热度信息DetailItem（即小说详情页展示的信息）。

### 5.2.3 实现爬虫——/spiders/yunqi\_qq\_com.py

爬虫的实现主要是对目标页面的解析。我们使用两个函数：parse\_book\_list()用于解析图书列表页，获取小说的基本信息；parse\_book\_detail()用于解析小说详情页，获取小说的点击量和人气等数据。

### 5.2.4 保存数据——/pipelines.py

我们将爬虫获取到的数据保存到MongoDB中。在MongoDB中，我们创建了两个collection，分别是bookInfo和bookhot，对应5.2.2中的两个item。在pipelines.py中，我们定义了一个类YunqicrawlPipeline，并将数据库的连接、插入等操作封装在这个类中。数据库的信息封装在配置文件(settings.py)中。pipelines.py完成后，还要在配置文件中添加如下代码激活YunqicrawlPipeline：

|  |
| --- |
| 激活YunqicrawlPipeline |
| ITEM\_PIPELINES = {  'yunqiCrawl.pipelines.YunqicrawlPipeline': 300,  } |

最后，执行如下命令，爬虫即可启动：

|  |
| --- |
| 启动Spider |
| scrapy crawl yunqi.qq.com |

## 5.3 实验结果数据部分展示



图5.1 bookInfo表中的部分数据



图5.2 bookhot表中的部分数据

第6章 总结与体会

通过本项目的实现，我们熟悉了抓取网站数据的流程，学会了使用现有的爬虫框架——Scrapy抓取数据；在项目中使用了Github仓库进行项目的远程托管和代码的版本控制，极大地提高了团队合作开发的效率；实现分布式爬虫的关键是调用了开源工具包scrapy-redis，将爬虫从单个主机分配到多个主机，并且通过搭建MongoDB集群，增强了系统的稳定性。最重要的一点是，本项目让我们熟练掌握了MongoDB的相关知识，包括MongoDB与关系型数据库的区别与联系、MongoDB中的基本概念（文档、集合等）、使用Python连接MongoDB的方法、在Python中对MongoDB进行基本数据操作的方法、MongoDB的主从集群的搭建以及主节点和从属节点的权限差异等。

本项目的源代码及相关资料保存在Github仓库中：

<https://github.com/Hanseltu/DDB-Pro.git>