15 为网易爬虫配置存储大规模数据存储

更新时间：2019-06-14 14:35:43

> 困难只能吓倒懦夫懒汉，而胜利永远属于敢于等科学高峰的人。
>
> ——茅以升

增量式爬虫面对的是逐渐增加的海量数据，显然用文件作为提取数据的存储媒介是完全不能满足其后续的需要的。本节将介绍如何把从网易提取到的巨量数据存储到与 Scrapy 天生契合的 NoSQL 数据库 MongoDB 中。

- 什么是 Pipeline? 在 Scrapy 中有何作用 ？
- 如何编写 Pipeline - `ItemPipeline` 的介绍
- 编写 `MongoPipeline`
  - 安装 pyMongo
  - 如何在 Pipeline 中读取 Pipeline 的配置
  - 丢弃无效的数据项目 `DropItem`
  - 将 Item 写入 MongoDB
  - log 输出写入结果
- 如何安装 MongoDB - Docker
- 使用MongoDB Compress 查看网易文章爬取结果并进行数据结构调优
- 小结

如果持续运行网易爬虫你会发现的 `result.json` 文件很快就变成一个体积巨大的文本，超过200M以后就基本打不开了。面对像网易爬虫这种要应对海量爬取数据的项目单纯将数据结果写入到一个JSON文本的方式已经显得不可取。在这种情况下我们需要使用数据库来处理这种大规模的数据存储。

为了可以像处理JSON一样方便地逐个处理请求结果，本节将会使用MongoDB作为爬虫项目的后端存储，使用管道(Pipleline)来向MongoDB写入数据。

什么是 **Pipeline?** 在 **Scrapy** 中有何作用 ？

当数据项(`Item`)在蜘蛛(Spider)中被收集之后，它将会被Scrapy传递到管道中（Item Pipeline），Scrapy会按照一定的顺序将数据项（Item）传入管道执行处理，简言之：**管道就是针对单个数据项(Item)的处理器**。

每个管道(“Item Pipeline”)是实现了简单方法的Python类。它们接收Item并对Item执行一些特定的处理，也决定此数据项(Item)是否能继续通过管道向下一个管道传递（直接丢弃）。

以下是管道的一些典型应用：

- 清理HTML数据；
- 验证爬取的数据(检查Item包含某些字段)；
- 清洗不合格规范的数项；
- 将数据项写入到数据库；

除了以上的几种应用，其实管道可以处理任意与Item相关（丢弃与继续）的事务。

如何编写 **Pipeline -** `ItemPipeline`的介绍

编写自定义的管道非常简单，每个管道都是一个具有`process_item`方法的Python类，具体如下所示。

```
class MyPipeline(object):
    def process_item(self, item, spider):
        return item
```

每个管道都需要调用`process_item`方法，而且这个方法**必须**返回一个`Item`对象或是抛出一个`DropItem`异常。以下是`process_item`方法的参数说明：

- `item`——当前处理的Item对象；
- `spider`——产生当前Item对象的蜘蛛实例。

> 注意：管道每次只处理一个Item对象。

编写`MongoDBPipeline`

接下来我们编写一个`MongoDBPipeline`用于向MongoDB插入爬取的文章内容。

MongoDB简介

MongoDB是一个文档型的非关系数据库产品，是非关系数据库当中功能最丰富，最像关系数据库的。它支持的数据结构非常松散，是类似json的bson格式，因此可以存储比较复杂的数据类型。Mongo最大的特点是它支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部分功能，而且还支持对数据建立索引。

MongoDB的基本概念

- 文档：是MongoDB中数据的基本单元，非常类似于关系型数据库系统中的行（但是比行要复杂很多）
- 集合：就是一组文档，如果说MongoDB中的文档类似于关系型数据库中的行，那么集合就如同表
- MongoDB的单个计算机可以容纳多个独立的数据库，每一个数据库都有自己的集合和权限
- MongoDB自带简洁但功能强大的JavaScript shell，这个工具对于管理MongoDB实例和操作数据库作用非常大
- 每一个文档都有一个特殊的键"_id",它在文档所处的集合中是唯一的，相当于关系数据库中的表的主键

安装 **pyMongo**

要在python中访问MongoDB推荐使用pyMongo，这是由MongoDB官方提供的python客户端。在虚环境下安装pyMongo:

```
$ pip install pyMongo
```

pyMongo的使用非常简单，首先需要实例化一个`MongoClient`类，通过这个类与MongoDB进行连接后就能对数据库进行常规的CRUD操作了，具体如下：

```
import pymongo
connection = pymongo.MongoClient('localhost',27017) #连接MongoDB
db = connection['数据库名称'] # 获取数据库实例
collection = db['数据集名称'] # 获取数据集实例(相当于数据表)
```

当获得`collection`就可以进行对象操作了。

与MongoDB的连接可以只在管道实例化时进行，并将`collection`存为管道对象的内部属性这样就无需要每操作一个数据项都与MongoDB发生一次连接，避免额外的资源消耗。具体代码如下：

```
import pymongo

class MongoDBPipeline(object):
    """
    MongoDB数据管道
    """
    def __init__(self, server=None, port=None, db_name=None, col=None):
        connection = pymongo.MongoClient(server, port)
        db = connection[db_name]
        self.collection = db[col]
```

如何在 **Pipeline** 中读取 **Pipeline** 的配置

与我们上两章介绍的去重处理器稍有不同，管道并不是从`from_settings`的类级方法来让Scrapy"注入"配置对象,而改用了`from_cralwer`这个方法，这一点必须注意：

```
@classmethod
def from_crawler(cls, crawler):
  server = crawler.settings.get('MONGODB_SERVER'),
  port = crawler.settings.getint('MONGODB_PORT')
  db_name = crawler.settings.get('MONGODB_DB')
  collection_name = crawler.settings.get('MONGODB_COLLECTION')
  return cls(server, port, db_name, collection_name)
```

这样就为`settings.py`增加了适用于`MongoDBPipeline`的配置项，具体如下所示：

```
# settings.py
MONGODB_SERVER = "localhost"
MONGODB_PORT = 27017
MONGODB_DB = "数据库名"
MONGODB_COLLECTION = "表名"
```

将 **Item** 写入 **MongoDB**

MongoDB的使用有点像传统的SQL数据库，一开始要建立一个数据库连接，然后是获得数据库的实例，接着是得到具体集合(表)的实例，最后才是向集合写入数据　　。如果频繁建立数据库的连接会对服务器的资源产生巨大的消耗，所以我们可以在蜘蛛启动时打开连接，蜘蛛完成爬取任务时关闭数据库连接。这样做就可以避免做无用功消耗不必要的资源。

在`ItemPipline`可以增加`open_spider`和`close_spider`这两个方法分别向蜘蛛打开事件与蜘蛛关闭事件加入特定的处理，具体如下所示：

```
def open_spider(self, spider):
  self.connection = pymongo.MongoClient(self.server, self.port)
  self.db = self.connection[self.db_name]
  self.collection = self.db[self.col]

  def close_spider(self, spider):
    self.connection.close()
```

最后就是向MongoDB插入数据了，向MongoDB插入数据是最简单的操作，只要调用`MongoClient`的`insert`方法并将数据项作为字典对象插入即可，具体如下所示：

```
def process_item(self,item,spider):
   self.collection.insert(dict(item))
    return item
```

这就大功告成了，是不是很简单呢？

丢弃无效的数据项目 `DropItem`

在爬网的过程中经常会爬到一些对我们没有用或者是重复的数据，我们可以在存储这些`Item`数据之前加入一个管道来判断这些数据的可用性，保留有用的、丢弃无用的。只要在`process_item`处理方法中发起`scrapy.exceptions.DropItem`的异常就能完成**丢弃**的动作，并不用担心在`process_item`中引发异常会导致整个爬网过程的中止，因为管道的每次处理是针对单个`Item`对象进行的。

在本示例中，如果文章没有标题(`item['title']`)则我们可以认为这是一个无效的数据，因为不可能出现没有标题的文章。此时，我们就会利用`DropItem`这个异常来跳出当前数据项的处理，只要将`process_item`改为:

```
def process_item(self, item, spider):
  if item['title'] is None:
    raise DropItem()
  self.collection.insert(dict(item))
  return item
```

这里切记一定要在`process_item`的最后返回`Item`对象，否则管道中"流动"的数据项就此被截断了。

log 输出写入结果

为了将爬虫写入数据库这个过程可以被输出到控制台，这样能更便于我们在开发时观察`MongoDBPipline`对象是成功地将数据项写入到MongoDB中。虽然Scrapy提供了内部的日志对象，但其实在python项目中我更喜欢使用python自带的`logging`模块，因为它的通用性可以让你的日志变得更加的灵活可用（更多的用法可以参见Python官方的模块loggingPython 的日志记录工具）。

在导入所有依赖包之后可以加入以下的代码初始化日志模块：

```
logger = logging.getLogger(__name__)
```

作为一个模块级的变量，我们可以在插入数据项之后输出一个简单提示，并记录到Scrapy的状态收集器内。具体代码如下所示：

```
logger.debug("成功将数据插入至MongoDB", extra={'spider': spider})
spider.crawler.stats.inc_value(
            'mongodb/inserted', spider=spider)
```

`MongoDBPipline`的完整代码如下所示：

```
# -*- coding: utf-8 -*-

import pymongo
import logging
from scrapy.exceptions import DropItem

logger = logging.getLogger(__name__)


class MongoDBPipeline(object):
    """
    MongoDB数据管道
    """

    def __init__(self, server=None, port=None, db_name=None, col=None):
        self.server = server
        self.port = port
        self.db_name = db_name
        self.col = None

    def open_spider(self, spider):
        self.connection = pymongo.MongoClient(self.server, self.port)
        self.db = self.connection[self.db_name]
        self.collection = self.db[self.col]

    def close_spider(self, spider):
        self.connection.close()

    @classmethod
    def from_crawler(cls, crawler):
        server = crawler.settings.get('MONGODB_SERVER'),
        port = crawler.settings.getint('MONGODB_PORT')
        db_name = crawler.settings.get('MONGODB_DB')
        collection_name = crawler.settings.get('MONGODB_COLLECTION')
        return cls(server, port, db_name, collection_name)

    def process_item(self, item, spider):
        if item['title'] is None:
            raise DropItem()

        self.collection.insert(dict(item))
        logger.debug("成功将数据插入至MongoDB", extra={'spider': spider})
        spider.crawler.stats.inc_value(
            'mongodb/inserted', spider=spider)
        return item
```

配置`MongoDBPipline`

既然我们已经完成了`MongoDBPipline`的开发，那接下来应如何如何将这个管道接入到我们的爬虫项目中来呢？

首先在`settings.py`中添加MongoDB的基本配置信息，虽然现在还没有安装MongoDB但请不要着急，我们可以先将配置写入，安装MongoDB时根据我们的配置来建立数据库和集合就可以了。

```
MONGODB_SERVER = "localhost"      # MongoDB 服务器地址
MONGODB_PORT = 27017              # MongoDB 服务器的访问端口
MONGODB_DB = "netease"          # MongoDB 采用的数据库名
MONGODB_COLLECTION = "articles"    # 写入的集合名称
```

接下来，最重要的一步就是将`MongoDBPipline`加入到的管道配置信息中了。要将自定义的管道对象接入至Scrapy需要在`ITEM_PIPELINES`配置项内加入管道对象的python包全名称:

```
ITEM_PIPELINES = {
    'netease_crawler.pipelines.MongoDBPipeline': 300
}
```

最后的`300`指的是优先级，这个数值越大优先级将会越低。取值范围在`0`-`1000`之内任意取值。

如何安装 MongoDB - Docker

正如前文安装Redis那样，我推荐使用Docker来安装mongo。首先从docker的官方镜像库中找到由mongoDB官方提供的镜像：

```
$ docker search mongo
```

通过`pull`指令将"mongo"镜像拉取至本地：

```
$ docker pull mongo
```

MongoDB的默认数据访问端口为27017，数据文件的存储路径为`/data/db` ，那么我们就将docker容器的27017端口映射到本机，同时将docker容器的数据目录挂载到当前工作目录下的`db`目录中，具体指令如下所示：

```
$ docker run -p 27017:27017 -v $PWD/db:/data/db -d mongo
```

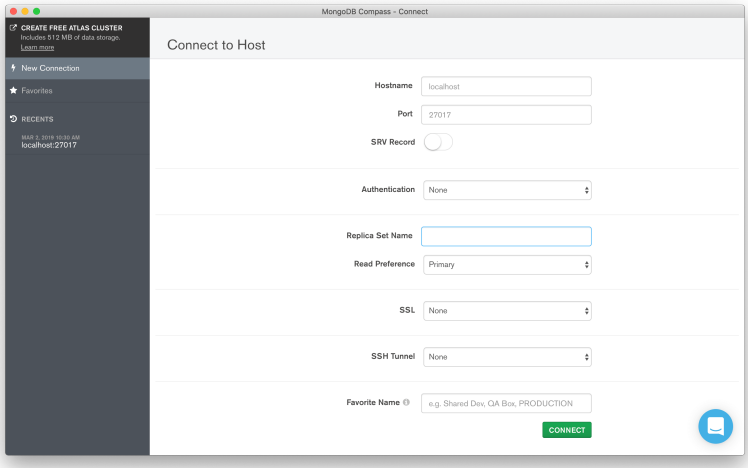
最后，运行的`ps`指令查看一下mongo是否被正常地运行起来：

```
$ docker ps
```

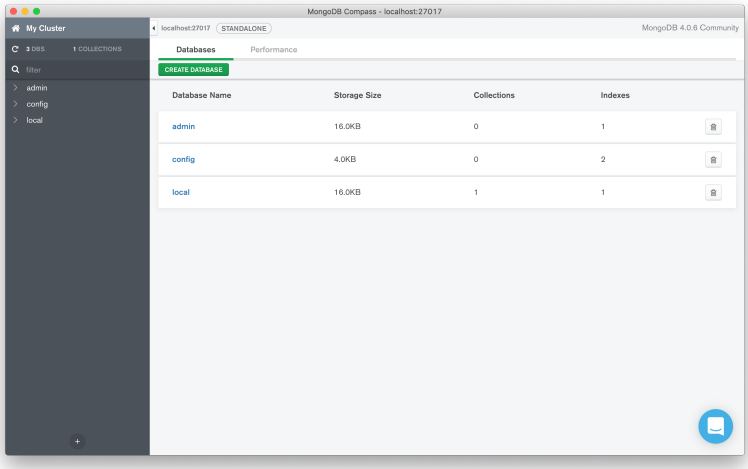
使用MongoDB Compass 查看爬取结果

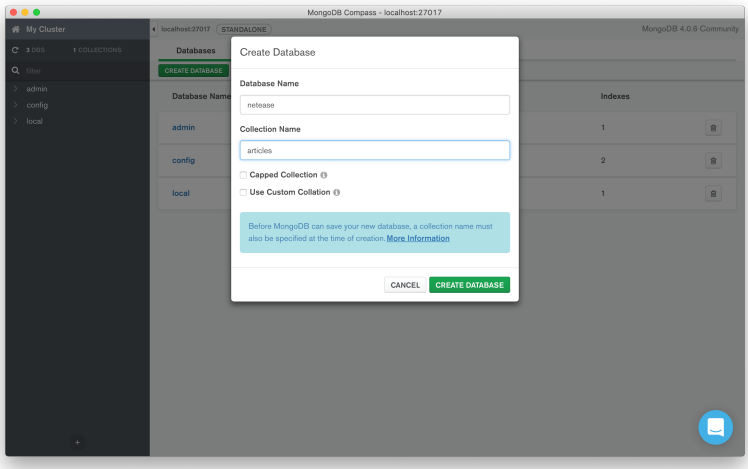
MongoDB官方提供了一个非常实用的数据库管理工具：MongoDB Compass，你可以访问上述链接下载与你操作系统相匹配的运行版本安装使用。

运行的MongoDB Compass, 会看到以下要求连接MongoDB的界面：



我们可以直接按默认值来连接就好了。进入MongoDB Compass的管理界面会看到MongoDB内置的三个系统数据库。直接点+号创建一个新的数据库。



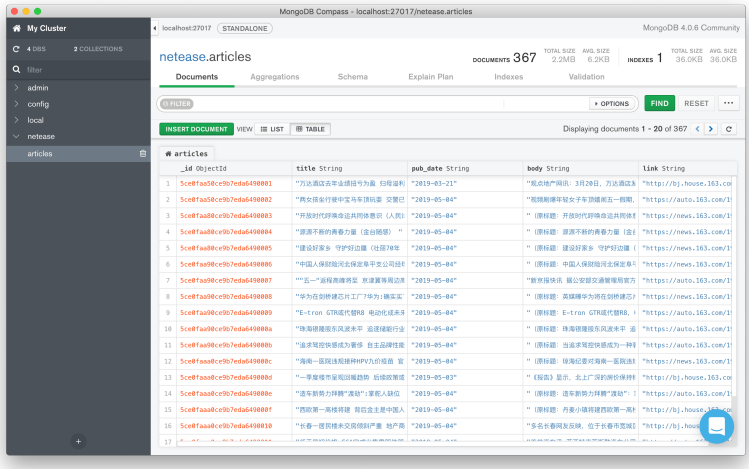


按我们之前的配置的数据库名(Database Name) 就用`netease` 默认集合名称(Collection Name)就使用`articles`，完成输入后点击"CREATE DATABASE"就可以轻松地创建好MongoDB的数据库环境了。

接下来就是直接运行网易爬虫让这个高性能的爬虫全面地运转起来了!

```
$ scrapy crawl netease
```

让爬虫爬一会再打开MongoDB Compass可以很便快速地查看爬取后的结果:

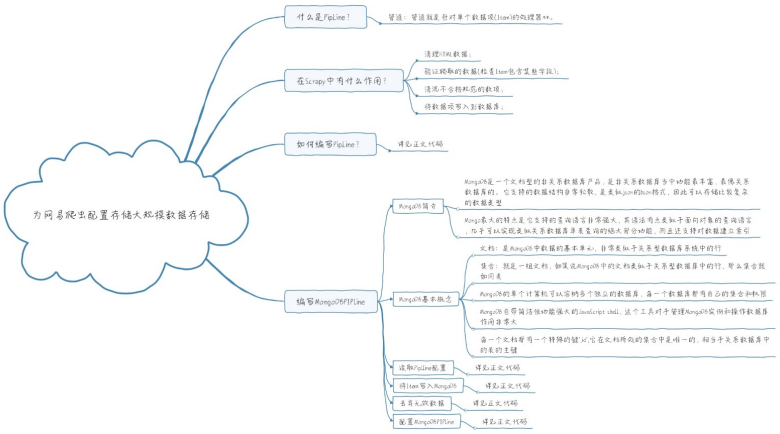


小结

本节的内容是完成一个完整爬虫项目的最后关键性的一环，在面对具有大规模海量数据的爬取目标时，有效地存储爬取结果才能给整个项目的设计画上一个句号。

如果您是跟着本章内的示例一步一步走来，即使没有使用专业工具收集性能数据，你也会明显地发现你的爬虫运行速度是在不断地加快的！这就是有优化与没有优化的最大区别。我们也来回顾一下这个优化的历程：

1. 采用`ItemLoader`有效分离元素结构分析逻辑与数据结构，可以让内容分析的代码变得更为易读，能在一个分析代码中适应多种不同的页面元素结构。
2. 采用基于Redis的去重过滤器可以有效地避免去重文件过大而被撑爆的风险。
3. 在Redis去重过滤器的基本上使用性能更好的布隆算法使去重过程进一步提速，对于网易这类到处充斥着重复链接的网站效果最为明显。
4. 用MongoDB作为后端存储可以避免数据结果文件过大而无法打开的囧境，并且还可以根据日后收集的数据量进行动态扩展，让爬虫项目变得更有弹性。



14 高效的布隆过滤器 - RedisBloomDupeFilter

16 增量式爬虫与增量爬网策略设计