

# 基于 Airflow 的自动化 Scrapy 爬虫系统

肖强

修改日期：2020.2.19

# Contents

<b>1</b>	<b>基本介绍</b>	<b>1</b>
1.1	项目总体设计 . . . . .	1
1.2	本文主要内容 . . . . .	1
<b>2</b>	<b>相关配置</b>	<b>3</b>
2.1	基础配置 . . . . .	3
2.2	Mongo 服务 . . . . .	5
2.2.1	Docker 部署 . . . . .	5
2.3	Mysql 服务 . . . . .	6
2.3.1	Docker 部署 . . . . .	6
2.3.2	容器一直重启原因排查 . . . . .	7
2.3.3	可能出现的问题 . . . . .	8
2.4	WordPress 博客服务 . . . . .	9
2.4.1	Docker 部署 . . . . .	9
2.4.2	可能出现的问题 . . . . .	10
<b>3</b>	<b>Scrapy 爬虫</b>	<b>11</b>
3.1	Scrapy 爬虫框架 . . . . .	11
3.1.1	新建 Scrapy 爬虫工程 . . . . .	11
3.1.2	Scrapy 架构 . . . . .	12
3.1.3	Scrapy 工作流程 . . . . .	14
3.1.4	爬虫流程 . . . . .	15
3.1.5	items 对象 . . . . .	15
3.2	获得文章的 url 链接 . . . . .	15
3.2.1	利用解析 sitemap.xml . . . . .	15
3.2.2	分析修改 url 链接 . . . . .	18

3.2.3	文章上下关系 . . . . .	18
3.3	使用 xpath 获取文章内容 . . . . .	19
3.4	存在的问题 . . . . .	20
3.5	管道中的数据处理 . . . . .	21
3.5.1	上传至 mongo 数据库 . . . . .	21
3.5.2	上传至 WordPress 服务器 . . . . .	22
<b>4</b>	<b>Airflow 调度 Scrapy</b>	<b>24</b>
4.1	airflow 调度系统 . . . . .	24
4.2	Docker 部署 . . . . .	25
<b>附录 A</b>		<b>27</b>
pipeline.py	爬虫管道 . . . . .	27
jiqizhixin.py	(机器之心) . . . . .	30
qbitai.py	(量子位) . . . . .	34
tencent.py	(腾讯云专栏) . . . . .	37

# 1 基本介绍

## 1.1 项目总体设计

本项目设计主要考虑分四个服务模块 (整体架构图见 (1)):

- 监管服务
  - airflow 调度 (Docker1 -p 8080:8080)
- 功能服务
  - scrapy 爬虫 (Docker1)
  - 文本分析处理 (Docker1)
- 存储服务
  - Mongo 数据库服务 (Docker2 -p 8088:3306)
  - MySQL 数据库服务 (Docker3 -P 8089:27107)
- 远程服务
  - Internet 服务
  - WordPress 服务 (Docker4 -p 8081:8080)

## 1.2 本文主要内容

本文主要做一些介绍如何使用 scrapy 创建一个爬虫并能使用 airflow 进行调度管理, 并且未来还可以在该系统上新增文本分析服务。全文内容主要包含如下四个部分:

第一部分主要介绍了我们项目设计的整体框架, 其中包含了整体架构图一幅;

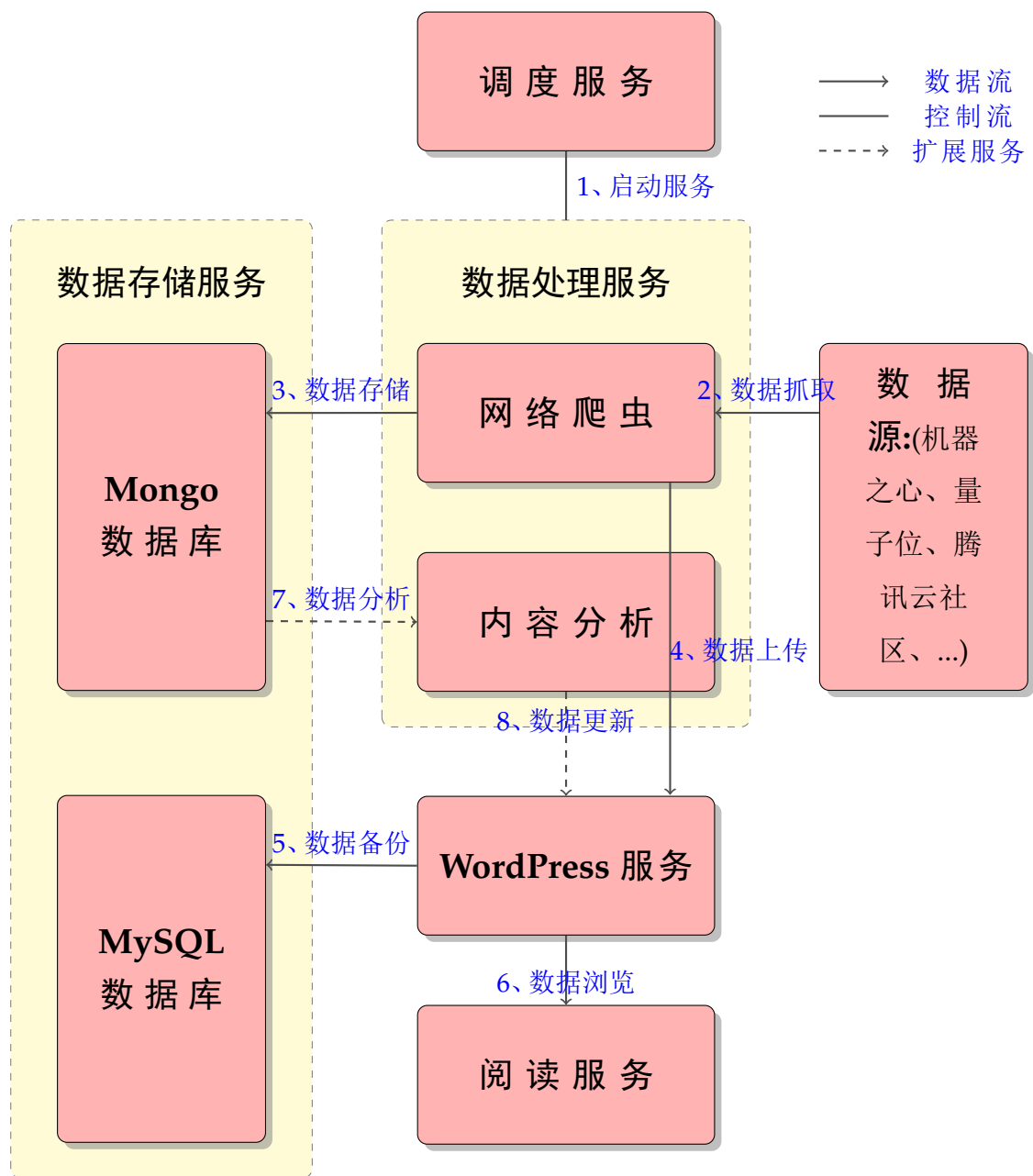


Figure 1: 整体设计架构图

第二部分主要讲述项目所需的第三方依赖和代码目录结构,并详细介绍了容器化部署 mysql、mongo、wordpress 的方法,这也是我们所依赖的相关项;

第三部分主要介绍爬虫的框架、实现,以及如何把文本数据上传到 WORDPRESS 服务器或者 Mongo 服务器,其中包含 pipeline 和 spider 的流程图两幅, scrapy 基础架构图一幅;

第四部分主要是介绍 Airflow 的部署和使用,其中包含一个 UML 时序图。

## 2 相关配置

### 2.1 基础配置

我们使用如下的配置完成这次开发:

#### 1. 开发环境:

- Python:3.7.0
- Docker

#### 2. Python 第三方库:

- scrapy
- airflow
- pymysql
- pymongo
- python-wordpress-xmlrpc

#### 3. Docker 镜像包:

- python:3.7.0-alpine
- wordpress

- mysql
- mongo
- puckel/docker-airflow

项目目录结构:

---

```
get_article/  
    scrapy.cfg          # deploy configuration file  
airflow-scrapy.py      # DAG of airflow  
get_article/          # project's Python module, you'll import your  
    code from here  
    __init__.py  
    items.py           # project items definition file  
    middlewares.py     # project middlewares file  
    pipelines.py       # project pipelines file  
    settings.py        # project settings file  
    spiders/           # a directory where you'll later put your spiders  
    __init__.py  
    jiqizhixin.py      # spider of www.jiqizhixin.com  
    qbitai.py          # spider of www.qbitai.com  
    tencent.py         # spider of  
        cloud.tencent.com/developer/articles?q=timeline  
    aliyun.py          # spider of yq.aliyun.com/articles
```

---

## 2.2 Mongo 服务

### 2.2.1 Docker 部署

网上有许多基于 Docker 部署 Mongo 数据库服务的技术帖<sup>1 2</sup>, 所以我们就简单介绍一下我们实现这个操作的流程吧, 整个过程都是使用的 shell 语句。

1. 在镜像仓库中搜索 mongo 的镜像文件:

- `docker search mongo`

2. 拉取 mongo 镜像至本地;

- `docker pull mongo`

3. 新建一个名为"mongo" 的文件夹用于与之后与容器内 mongoDB 的数据进行挂载;

- `mkdir /Docker/mongo`

4. 以本地的 mongo 镜像作为基础镜像运行一个 Docker, 并将 Docker 内的"/data/db" 文件夹挂载到本地的"\$:/mongo/db" 文件夹 (下列两条命令等价)。

- `docker run -d --restart=always -v $PWD/mongo/db:/data/db --name mongo -p 54001:27017 mongo:latest`
- `docker run -d --restart=always -v $(pwd)/mongo/db:/data/db --name mongo -p 54001:27017 mongo:latest`

通过上面的 4 个步骤我们已经实现了 mongoDB 的容器化部署, 可以通过使用"docker ps" 这一命令查看名为 'mongo' 的容器是否运行成功以及一些相关信息 (见 (2))。或者使用如下命令直接查看容器内的 Mongo 数据库中数据:

---

<sup>1</sup><https://cloud.tencent.com/developer/article/1329170>

<sup>2</sup><https://zgljl2012.com/docker-zhong-shi-yong-mongodb/>



```
[root@localhost Docker]# docker run -d --rm --name mongoDB -v /Docker/mongo/db:/data/db -p 54001:27017 mongo:latest
fbb693612a24002682426804574b8f50cdb1b6f16b525b2d01f80021e024b603
[root@localhost Docker]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
fbb693612a24	mongo:latest	"docker-entrypoint.s..."	4 seconds ago	Up 3 sec
onds	0.0.0.0:54001->27017/tcp		mongoDB	

Figure 2: "docker ps" 这一命令查看名为 ‘mongo’ 的容器是否运行成功

- mongo 10.2.174.40:54001 远程连接 Mongo 数据库: mongo + 宿主机 IP + 容器映射宿主机的端口
- show dbs; 查看 Mongo 数据库中已有数据库 DB

## 2.3 Mysql 服务

### 2.3.1 Docker 部署

我们使用如下指令获取 MySQL 的 docker 镜像, 并运行一个容器部署 MySQL 服务:

- docker pull mysql:latest
- mkdir /Docker/mysql
- docker run --restart=always --name mysql -v \$PWD/mysql:/var/lib/mysql -d -p 54002:3306 -e MYSQL\_ROOT\_PASSWORD=123456 mysql:latest

现在, 我们已经在 Docker 中部署了 MySQL 服务。通过宿主机的 IP 和 54002 端口进行连接, 其中密码 (MYSQL\_ROOT\_PASSWORD) 为: 123456, 实例详见 (3)。通过上述远程连接 mysql 数据库的实例 (3) 可以验证 mysql 服务是否成功部署, 下面列举一些部署 Mysql 服务时可能出现的问题。

```
[root@localhost Docker]# mysql -uroot -p -h10.2.174.40 -P54645
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 683744
Server version: 8.0.18 MySQL Community Server - GPL

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]>
```

Figure 3: 远程连接 mysql 数据库

### 2.3.2 容器一直重启原因排查

在使用如下命令:

- `docker run --restart=always --name mysql -v $PWD/mysql:/var/lib/mysql -d -p 54002:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql:latest`

启动容器后, 发现容器未正常运行, 一直重启, 见 (4). 使用"docker logs container-

```
[root@localhost Read-Blog-Server]# docker run --restart=always --name server-mysql -d -p 54002:3306 mysql:latest
97ae34b5626a0d945cf514597eaf090f551a4e1ad7e4075a5cafee3530a0383d
[root@localhost Read-Blog-Server]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
97ae34b5626a	mysql:latest	"docker-entrypoint.s..."	4 seconds ago	Restarting (1)	Less than a second ago	server-mysql
d827d1c263c2	mongo:latest	"docker-entrypoint.s..."	33 minutes ago	Up 33 minutes	0.0.0.0:54001->27017/tcp	server-mongo

Figure 4: "Docker 一直重启"

ID" 查看运行日志 (5), 这其实是数据库未初始化, 而关键是我们的挂载目录发生错误, 即我们对数据库数据的挂载发生错误导致数据库无法找到数据库配置文件。因此此时填写正确的挂载路径, 如:

- `docker run --restart=always --name mysql -v $PWD/mysql/mysql:/var/lib/mysql -d -p 54002:3306 -e MYSQL_ROOT_PASSWORD=123456 mysql:latest`

此时才能使用我们之前保存的数据库数据挂载到新的数据库, 并且数据库正常启动。

```
[root@localhost Read-Blog-Server]# docker logs server-mysql
2019-12-25 05:59:04+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
2019-12-25 05:59:04+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2019-12-25 05:59:04+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
2019-12-25 05:59:04+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
2019-12-25 05:59:05+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
2019-12-25 05:59:05+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2019-12-25 05:59:05+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
2019-12-25 05:59:05+00:00 [ERROR] [Entrypoint]: Database is uninitialized and password option is not specified
You need to specify one of MYSQL_ROOT_PASSWORD, MYSQL_ALLOW_EMPTY_PASSWORD and MYSQL_RANDOM_ROOT_PASSWORD
2019-12-25 05:59:06+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 8.0.18-1debian9 started.
2019-12-25 05:59:06+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
```

Figure 5: "ERROR:2059"

```
[root@localhost Docker]# mysql -uroot -p -h10.2.174.40 -P54000
Enter password:
ERROR 2059 (HY000): Authentication plugin 'caching_sha2_password' cannot be loaded: /usr/lib64/mysql/plugin/caching_sha2_password.so: cannot open shared object file: No such file or directory
```

Figure 6: "ERROR:2059"

### 2.3.3 可能出现的问题

在连接 MySQL 数据库时, 出现"ERROR:2059", 见 (6)。这是由于新版本特性导致的, 解决办法是进入容器修改 mysql 数据库中 mysql 数据库的数据表。具体过程如下: 进入容器, 连接 mysql 后使用如下两条命令可以看到发生这类错误的源头,

1. use mysql
2. select host,user,plugin from mysql.user;

查询结果见 (7), 因为认证方式改变导致的。在老版本里, 一般使用加密方式为 mysql\_native\_password。所以可以直接修改为老版本的加密方式,

- ALTER USER 'root'@%' IDENTIFIED WITH mysql\_native\_password BY '123456';

使用这条命令将加密方式进行修改 (见 (8)), 就可以解锁远程登陆的方式了。

```
mysql> use mysql
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select host,user,plugin from mysql.user;
+-----+-----+-----+
| host      | user           | plugin           |
+-----+-----+-----+
| %         | root           | caching_sha2_password |
| localhost | mysql.infoschema | caching_sha2_password |
| localhost | mysql.session   | caching_sha2_password |
| localhost | mysql.sys       | caching_sha2_password |
| localhost | root           | caching_sha2_password |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Figure 7: 查询结果

## 2.4 WordPress 博客服务

我们使用已有的 WordPress 来搭建博客系统, 用户可以通过访问指定的"IP"和"端口" 查看该博客。现计划中这个博客上主要包含一些 AI 动态以及一些相关技术贴, 其中 AI 动态主要来源于" 机器之心"、" 量子位" 这两个主要的公众号, 技术贴主要来源于" 腾讯云社区"。

### 2.4.1 Docker 部署

在 docker-hub 上同样也有 wordpress 的官方 Docker 镜像。因此, 很容易使用如下指令完成 wordpress 服务的部署<sup>3</sup>;

- docker search wordpress
- docker pull wordpress
- docker run -d --name wordpress --restart=always -p 8081:80

<sup>3</sup><https://blog.csdn.net/webyzx/article/details/78666344>

```
mysql> ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY '123456'
-> ;
Query OK, 0 rows affected (0.00 sec)

mysql> select host,user,plugin from mysql.user;
+-----+-----+-----+
| host      | user          | plugin          |
+-----+-----+-----+
| %         | root          | mysql_native_password |
| localhost | mysql.infoschema | caching_sha2_password |
| localhost | mysql.session  | caching_sha2_password |
| localhost | mysql.sys      | caching_sha2_password |
| localhost | root          | caching_sha2_password |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

Figure 8: 修改结果

```
-v /Docker/wordpress/html:/var/www/html
-e WORDPRESS_DB_HOST=10.2.174.40:54002
-e WORDPRESS_DB_USER=root
-e WORDPRESS_DB_PASSWORD=123456 wordpress
```

其中"WORDPRESS\_DB\_USER=root"、"WORDPRESS\_DB\_PASSWORD=123456"、"WORDPRESS\_DB\_HOST=10.2.174.40:54000" 是三个用于连接 MySQL 数据库服务的环境变量, 通过这些设置的环境变量将 wordpress 服务器上的数据存储至 MySQL 数据库。

### 2.4.2 可能出现的问题

当我们使用环境变量使得 wordpress 连接上数据库时, wordpress 上所有的数据均会存储在 mysql 数据库。这是 wordperss 保护数据的一种方式, 即: 本地数据的遗失并不会影响 wordpress 服务器上的数据。同时, 对本地数据的修改也只有上传到 MySQL 数据库时才有效。

例如: 我们之前已经使用 Docker 运行过一个 wordpress 服务, 这个服务通过 10.2.174.40:8080 进行访问。我们称这个服务为 wp\_a, 对应 8080 端口的 wp 服务

器。此时我们使用 Docker 再次运行另一个 wordpress 服务, 通过 10.2.174.40:8081 进行访问。我们称这个服务为 wp\_b, 对应 8081 端口的 wp 服务器。现在, 当我们通过 10.2.174:8081 访问 wp\_b 时, 会自动跳转访问 10.2.174:8080 的 wp\_a。即, 此时的 10.2.174.40:8081 只是起到一个重定向的作用, 并不是一个新生的 wordpress 服务。

因此, 当想运行多个不同的 wordpress 服务时应该将 wordpress 连接至不同的 MySQL 数据库。

## 3 Scrapy 爬虫

### 3.1 Scrapy 爬虫框架

为了爬虫开发的需要, 我们需要安装导入几个 Python 第三方库: scrapy, pymongo, python-wordpress-xmlrpc; 具体安装方法如下:

1. pip3 install scrapy
2. pip3 install pymongo
3. pip3 install python-wordpress-xmlrpc

#### 3.1.1 新建 Scrapy 爬虫工程

使用命令 "scrapy startproject get\_articles" 新建一个名为 "get\_articles" 的爬虫工程。这个 "get\_articles" 爬虫工程目录结构如下:

---

```
get_article/  
  scrapy.cfg          # deploy configuration file  
  get_article/        # project's Python module, you'll import  
                        your code from here  
  __init__.py
```

```
items.py          # project items definition file
middlewares.py    # project middlewares file
pipelines.py      # project pipelines file
settings.py       # project settings file
spiders/          # a directory where you'll later put your
    spiders
__init__.py
```

---

其中"items.py" 定义一个数据结构, "pipelines.py" 定义管道, settings.py 是管理 scrapy 爬虫工程的配置文件。"spiders/" 是一个存放爬虫的文件夹, 可以通过在"spiders/" 的同级目录下, 使用"scrapy genspider [spidername] '{url}'" 新建一个 spider 爬虫。

### 3.1.2 Scrapy 架构

Scrapy 框架主要由六大组件组成, 它们分别是调度器 (Scheduler)、下载器 (Downloader)、爬虫 (Spider)、中间件 (Middleware)、实体管道 (Item Pipeline) 和 Scrapy 引擎 (Scrapy Engine)。

- Scrapy Engine(引擎): 引擎负责控制数据流在系统的所有组件中流动, 并在相应动作发生时触发事件。
- Scheduler(调度器): 调度器从引擎接受 request 并将他们入队, 以便之后引擎请求他们时提供给引擎。
- Downloader (下载器): 下载器负责获取页面数据并提供给引擎, 而后提供给 spider。
- Spider (爬虫): Spider 是 Scrapy 用户编写用于分析 response 并提取 item(即获取到的 item) 或额外跟进的 URL 的类。每个 spider 负责处理一个特定 (或一些) 网站。

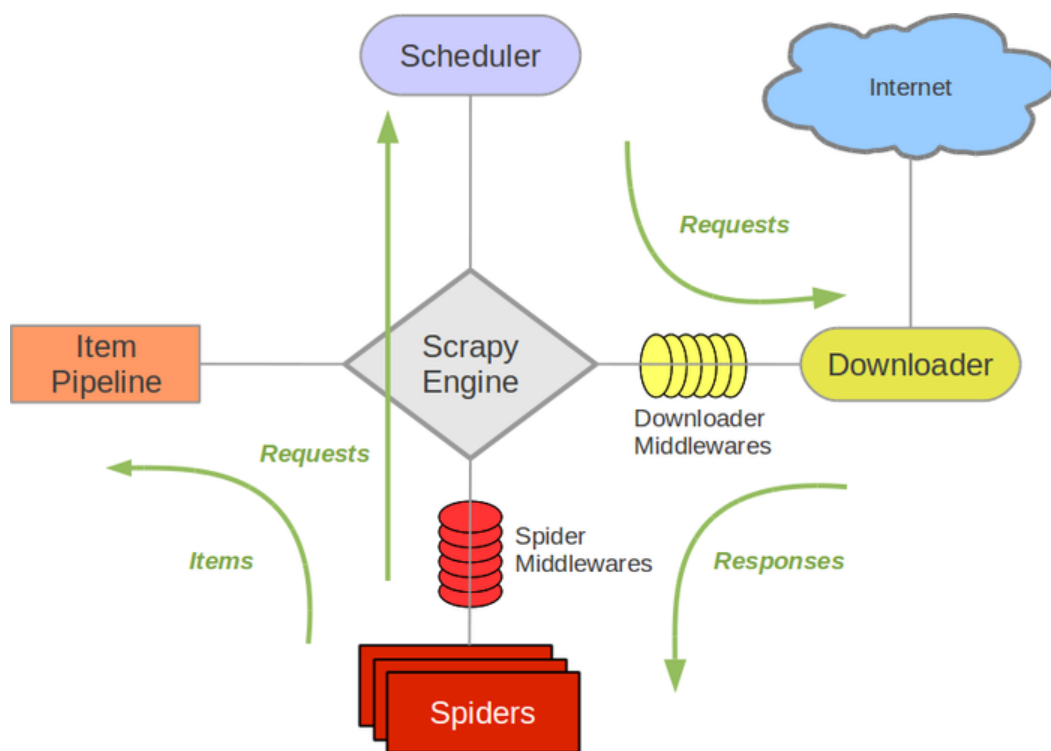


Figure 9: Scrapy 架构图

- **Item Pipeline(管道)**: Item Pipeline 负责处理被 spider 提取出来的 item。典型的处理有清理、验证及持久化 (例如存储到数据库中)。
- **Downloader Middlewares (下载中间件)**: 下载器中间件是在引擎及下载器之间的特定钩子 (specific hook), 处理 Downloader 传递给引擎的 response。其提供了一个简便的机制, 通过插入自定义代码来扩展 Scrapy 功能。
- **Spider Middlewares (Spider 中间件)**: Spider 中间件是在引擎及 Spider 之间的特定钩子 (specific hook), 处理 spider 的输入 (response) 和输出 (items 及 requests)。其提供了一个简便的机制, 通过插入自定义代码来扩展 Scrapy 功能。



### 3.1.3 Scrapy 工作流程

当我们通过 scrapy 框架写好代码并运行后，就会出现如下对话<sup>4</sup>：

1. 引擎：怎么样，爬虫老弟，搞起来啊！
2. Spider：好啊，老哥，来来来，开始吧。今天就爬 xxx 网站怎么样
3. 引擎：没问题，入口 URL 发过来！
4. Spider：呐，入口 URL 是"https://ww.xxx.com"。
5. 引擎：调度器老弟，我这有 request 请求你帮我排序入队一下吧。
6. 调度器：引擎老哥，这是我处理好的 request。
7. 引擎：下载器老弟，你按照下载中间件的设置帮我下载一下这个 request 请求。
8. 下载器：可以了，这是下载好的东西。（如果失败：sorry，这个 request 下载失败了。然后引擎告诉调度器，这个 request 下载失败了，你记录一下，我们待会儿再下载）
9. 引擎：爬虫老弟，这是下载好的东西，下载器已经按照下载中间件处理过了，你自己处理一下吧。
10. Spider：引擎老哥，我的数据处理完毕了，这里有两个结果，这个是我需要跟进的 URL，还有这个是我获取到的 Item 数据。
11. 引擎：管道老弟，我这儿有个 item 你帮我处理一下！
12. 引擎：调度器老弟，这是需要跟进 URL 你帮我处理下。（然后从第四步开始循环，直到获取完需要全部信息）

---

<sup>4</sup>Scrapy 工作原理分析 (简单易懂):<http://ddrv.cn/a/268676>

### 3.1.4 爬虫流程

Scrapy 爬虫是一个采用并发的计算方式, 在爬虫启动时, 各个组件、模块、函数都会运行起来直到爬虫完成。(流程图见 (11)(10))

### 3.1.5 items 对象

---

```
class GetArticlesItem(scrapy.Item):  
    name = scrapy.Field() # str  
    from_url = scrapy.Field() # str  
    title = scrapy.Field() # str  
    author = scrapy.Field() # list  
    publish_time = scrapy.Field() # datetime  
    update_time = scrapy.Field() # datetime  
    summary = scrapy.Field() # "" or True  
    content = scrapy.Field() # list  
    imgs = scrapy.Field() # list  
    category = scrapy.Field() # list  
    tags = scrapy.Field() # list  
    wp_id = scrapy.Field() # str
```

---

我们预定义了一个具有这些属性的 item 对象, 每篇文章只有生成一个这样的 item 才能在 pipeline 中处理。因此在解析文章的 html 内容时, 我们需要获取文章的这些相关信息。

## 3.2 获得文章的 url 链接

### 3.2.1 利用解析 sitemap.xml

sitemap.xml 文件是严格按照 xml 语言编写的网站地图, 用来引导搜索蜘蛛对本站点文章等内容的索引。量子位、机器之心两个网站都有各自的 sitemap.xml

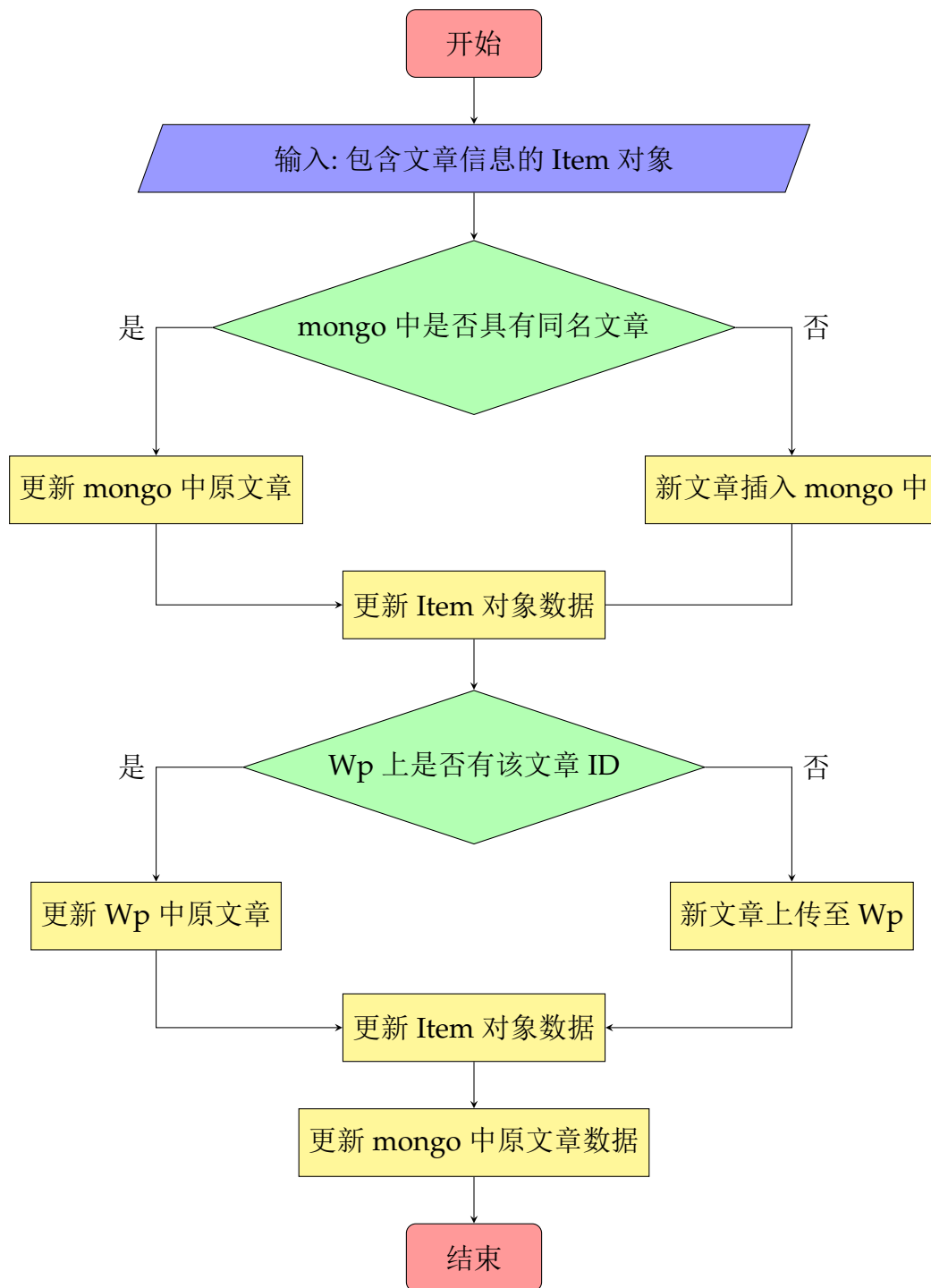


Figure 10: pipeline 处理流程

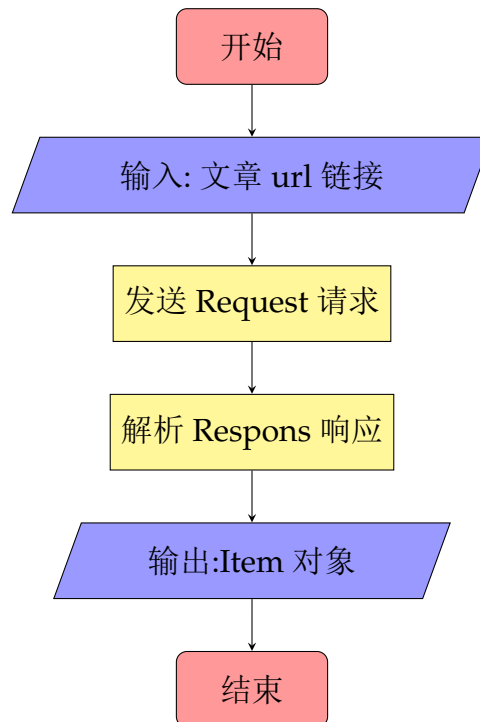


Figure 11: spider 抓取流程

文件, 方便各类爬虫抓取。通过解析 sitemap.xml 文件可以实现快速得到所需要的网页文章 url 链接。样例如下:

```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="text/xsl"
    href="http://www.uedsc.com/wp-content/plugins/google-sitemap-generato
    r/sitemap.xsl"?>
<urlset xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.sitemaps.org/schemas/sitemap/0.9
    http://www.sitemaps.org/schemas/sitemap/0.9/sitemap.xsd"
    xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
<url>
    <loc>http://www.uedsc.com/tag/net</loc>
    <lastmod>2015-02-27T01:12:09+00:00</lastmod>
```

```
<changefreq>weekly</changefreq>
<priority>0.3</priority>
</url>
<url>
  <loc>http://www.uedsc.com/tag/%e9%bd%bf%e8%bd%ae%e5%9b</loc>
  <lastmod>2014-08-08T01:10:39+00:00</lastmod>
  <changefreq>weekly</changefreq>
  <priority>0.3</priority>
</url>
</urlset>
```

---

其中"priority" 是指相对于其他页面的优先权, "changefreq" 则是指内容更新的频率。通过解析这个文件, 就可以快速获取网站的更新情况如何, 以及该网站内容的优先级。

### 3.2.2 分析修改 url 链接

以阿里云栖社区为例, 其上最早文章的 url 链接为:

- url = "https://yq.aliyun.com/articles/" + "1",

很容易发现, 这个 url 的数字部分从"1" 开始随着文章发布时间的增长而增加。因此, 我们只需要得阿里云上最新文章的数字 ID, 就可以得到当前所有可能的文章 url 链接。

例如, 假设云栖社区最新的文章 url 链接为:"https://yq.aliyun.com/articles/" + "x1", 此时云栖社区所有文章 url 链接的数字部分都在"1"~"x1" 之间。

### 3.2.3 文章上下关系

以腾讯云社区的专栏文章为例, 同样这个网站找不到 sitemap.xml 文件。虽然文章的 url 链接也有一串数字 ID, 但是这串数字 ID 最小值是我们未知的。如

上一篇: 拿来就能用! 如何用 AI 算法提高安全运维效率?

下一篇: 科幻元年2020来了! 说好的殖民火星和时光机呢?

Figure 12: 上一篇文章和下一篇文章的信息

果通过修改 url 链接也可以获得文章链接, 可能会出现许多无效请求。因此, 我们通过文章向上向下链接的关系获取其他文章的链接。例如, 腾讯云社区上的专栏文章页面都包含上一篇文章和下一篇文章的信息, 并且这个顺序是按时间严格排序的。因此我们只需要在解析这类网页时, 同时解析其中包含的上下篇文章链接, 就可以获取其他文章的 url 链接。

### 3.3 使用 xpath 获取文章内容

在得到文章的 url 链接后, 我们就可以得到包含文章 html 页的响应体了, 然后我们就需要对网页进行解析, 提取我们需要的资源信息。由于 xpath<sup>5</sup>底层是使用 C 语言实现, 不仅简单高效还具有规则性, 方便解析文章结构, 所以我们使用 xpath 进行网页信息提取, 我们使用 xpath 解析网页信息, 使用如下命令可以测试单个文章网页响应:

---

```
scrapy shell https://www.qbitai.com/2019/12/9606.html
```

```
...
```

```
>>> request.url
```

```
'https://www.qbitai.com/2019/12/9606.html'
```

---

通过检查原网页, 我们知道文章的题目所在的标签为 “h1”, 所以获取 “h1” 标签

---

<sup>5</sup>这里有一些关于 bs4 与 xpath 两种选择器的比较分析, 详细见: <https://blog.mimvp.com/article/25693.html>

下所有的文本内容:

---

```
>>> x = response.xpath('//div[@class="article"]')
>>> x.xpath('./h1/text()').get()
'Scikit-learn新版本发布, 一行代码秒升级'
>>> x.xpath('./h1/text()').getall()
['Scikit-learn新版本发布, 一行代码秒升级']
```

---

通过合理利用节点关系, 可以灵活的使用 xpath 提取文章的内容:

---

```
>>> c1 =
    x.xpath('./div[@class="line_font"]/preceding-sibling::*').getall()
>>> len(c1)
67
>>> c2 = x.xpath('./div[@class="article_info"]/following::*').getall()
>>> len(c2)
319
```

---

比较 c1 列表和 c2 列表中的内容, 其中重复出现的内容就是我们需要的文章内容了。

### 3.4 存在的问题

不同网站文章的 html 格式可能存在很大的结构差异, 因此不同的网站需要使用不同的爬虫进行解析。

此外, 关于动态页面的处理我们可能还需要很多时间的努力。例如, 阿里云栖社区上的文章我们依然无法加载全部文章, 所以没有得到全部文章的响应体, 得到的响应体内只包含部分的文章内容, 这还需要慢慢进行学习处理。

## 3.5 管道中的数据处理

### 3.5.1 上传至 mongo 数据库

将文章信息数据上传至 mongo 数据库需要用到 pymongo 第三方库, 下面是一段管道的示例:

---

```
class GetArticlesPipeline(object):

    def __init__(self, mongo):
        self.mongo = mongo

    @classmethod
    def from_crawler(cls, crawler):
        return cls(mongo=crawler.settings.get('MONGO'))

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo.get('mongodb'))
        self.db = self.client[self.mongo.get('db')]

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

    def process_item(self, item, spider):
        res = self.db[item['name']].insert_one(dict(item))
        item['author'] = res.inserted_id
        return item
```

---

其中关于"Mongo 数据库"的配置信息都写在 settings.py 文件中, 可以直接使用 crawler() 函数进行读取。每个管道中都必须实现"process\_item()"函数, scrapy 引擎会自动调用这个函数对流过管道的 item 对象进行处理。



在这个示例中我们使用"from\_crawler()" 获取数据库配置信息, 使用"open\_spider()" 连接数据库, "close\_spider()" 关闭数据库连接, 使用"process\_item()" 上传数据至数据库。

可以发现, 在"process\_item" 方法中, 我们实现了将爬取的文章数据上传至 Mongo 数据库, 并获取了文章的"\_id"。值得注意的是在后续处理时我们不再需要 item['author'] 这一属性值, 所以我们将文章存在 mongo 中的"\_id" 存在了这个属性上。

### 3.5.2 上传至 WordPress 服务器

将文章信息数据上传至 WordPress 服务器需要用到 python-wordpress-xmlrpc 第三方库, 下面是一段管道的示例:

---

```
import scrapy
from datetime import datetime
from scrapy.utils.project import get_project_settings
from wordpress_xmlrpc import Client
from wordpress_xmlrpc import WordPressPost
from wordpress_xmlrpc.methods import media, posts
class UptoWordPressPipeline(object):

    def __init__(self, wp):
        self.wp = wp

    def open_spider(self, spider):
        self.wpc = Client(
            self.wp.get('url'),
            self.wp.get('username'),
            self.wp.get('password'))
```

```

def close_spider(self, spider):
    pass

@classmethod
def from_crawler(cls, crawler):
    return cls(wp=crawler.settings.get('WORDPRESS'))

def process_item(self, item, spider):
    article = WordPressPost()
    article.title = item['title']
    article.date = datetime.strptime(item['publish_time'], '%Y-%m-%d
        %H:%M:%S')
    article.date_modified = datetime.now()
    article.post_status = 'publish'
    article.terms_names =
        {'post_tag':item['tags'],'category':item['category']}
    article.content = ''.join(item['content'])
    item['wp_id'] = self.wpc.call(posts.NewPost(article))
    return item

```

---

在上传数据至 Wordpress 服务器时, 我们需要先将 item 数据提取到 WordPress 对象中。要想设置文章的发布时间, 则".date" 属性值应该设置为一个 datetime 类型的数据, 否则 Wordpress 服务器只会以上传时间为作为发布时间。

另外, 在上传文档时, ".date\_modified" 属性表示文章状态, 不赋值时默认是草稿, “private” 表示私密的, “draft” 表示草稿, “publish” 表示发布。如果不设置发布状态为"publish", 那么该文章是不会有在站点页面出现的。

还有一个注意的地方是: 当使用 python-wordpress-xmlrpc 查询 wordpress 服务器上的文章时, 每次只能获得 50 个文本数据, 删除文本数据时也是至多 50

个。所以, 如果想大规模查询或者删除 WordPress 上的文章时, 可以使用脚本进行循环处理, 或者删除数据库。

## 4 Airflow 调度 Scrapy

### 4.1 airflow 调度系统

Airflow 是一个调度平台, 用于以编程方式编写, 进行安排和监视的工作流。

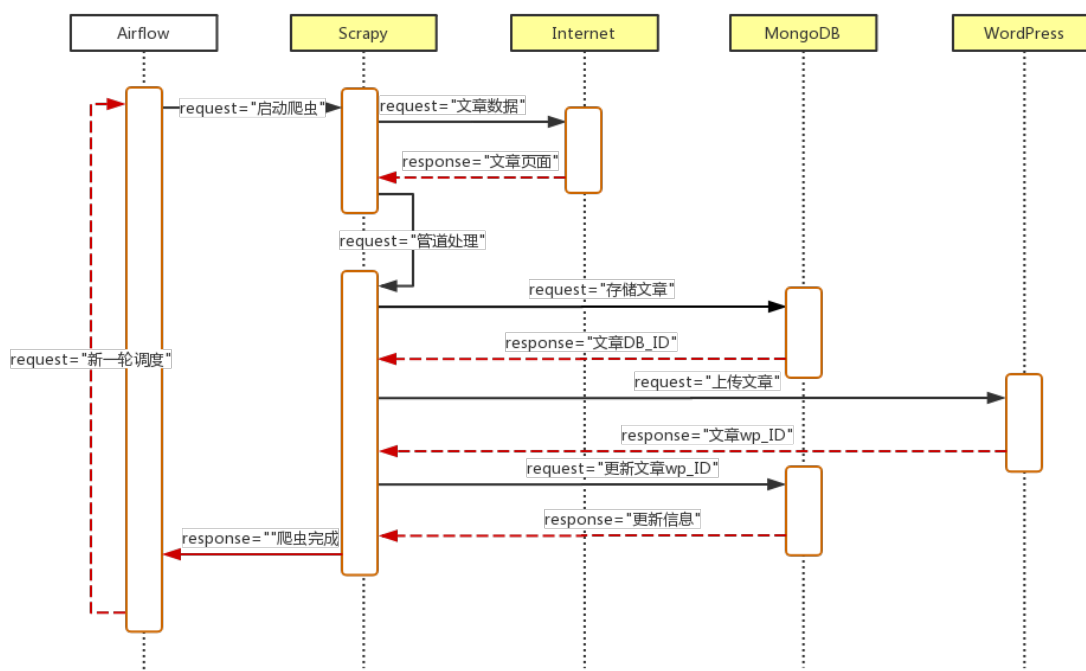


Figure 13: Airflow-scrapy 调度系统 UML 时序图

## 4.2 Docker 部署

在 Dockerhub 上有 airflow 的镜像包, 可以使用如下的命令获得:

### 1. Docker pull puckel/docker-airflow:latest

但是使用这个镜像包运行一个 Docker 时, 默认用户是 `airflow`, 只能执行有限的操作, 并且这个镜像包内只有 `python` 的标准库, 无法满足我们的开发需要。

一个解决办法是以该镜像为基础镜像, 在外层增加一层 `Python` 第三方库的依赖, 并且修改默认的用户为 `root` 用户。但是这种方法不仅很大程度上增加了镜像包的大小, 并且可能还要配置新装依赖的环境变量。


另一个解决办法是重新构建这个 `airflow` 的镜像包, 只需要修改原镜像包的 `Dockerfile`, 在 `Dockerfile` 内增加所需第三方依赖的安装指令, 同时修改默认用户即可。这种方法只依赖于网络质量, 因为在网络不佳时, `Docker` 镜像的构建会因为下载依赖时连接超时而失败。

我们分别实现了两种解决办法, 最后使用第二种办法获得的镜像包进行 `airflow` 的安装部署。使用我们重构的 `Docker` 镜像运行一个 `Docker`:

### 1. `docker run -d --rm --name airflow-scrappy -v $PWD/air-scrappy/dags/:/root/airflow/dags/ -p 8088:8080 air-scrappy:2.1`

其中 `/root/airflow/dags` 是容器的工作目录, 所有脚本都默认在这个目录下执行。并且, `airflow` 的配置文件中 `DAG` 的目录也是 `/root/airflow/dags`, 所有 `DAG` 只有在这个目录或者其子目录下才能被 `airflow` 的调度台进行调度。8080 是 UI 界面的默认端口, 通过映射使得访问宿主机的 8088 进行访问。

关于 UI 界面官网教程上有详细的介绍, 但是没有介绍如何查看任务日志, 我们只介绍一下如何查看任务日志。菜单项目栏中的“Browse”下的“Task Instances”可以查看每个完成的任务详细信息 (14), 在这个列表的最右端的一列就是任务日志, 用于检查以及查看控制台输出。


Airflow
DAGs
Data Profiling ▾
Browse ▾
Admin ▾
Docs ▾
About ▾

SLA Misses

**Task Instances**

Logs

Jobs

DAG Runs

## Task Instances

List (30)
Add Filter ▾
With selected ▾

<input type="checkbox"/>	State	Dag Id	Task Id	Execution Date	Operator	Start Date
<input type="checkbox"/>	success	scrapy_start03	jiqizhixin ▾	12-10T00:00:00+00:00	PythonOperator	12-11T01
<input type="checkbox"/>	success	scrapy_start03	qbitai ▾	12-10T00:00:00+00:00	PythonOperator	12-11T01
<input type="checkbox"/>	success	scrapy_start03	jiqizhixin ▾	12-09T00:00:00+00:00	PythonOperator	12-11T01
<input type="checkbox"/>	success	scrapy_start03	qbitai ▾	12-09T00:00:00+00:00	PythonOperator	12-11T01
<input type="checkbox"/>	success	scrapy_start03	print_the_context ▾	12-10T00:00:00+00:00	PythonOperator	12-11T01
<input type="checkbox"/>	success	scrapy_start03	jiqizhixin ▾	12-08T00:00:00+00:00	PythonOperator	12-11T01

Figure 14: Task Instances 页面

## 附录 A

### pipeline.py 爬虫管道

---

```
import re
import scrapy
import pymongo
from datetime import datetime
from scrapy.utils.project import get_project_settings
from scrapy.exceptions import DropItem
from wordpress_xmlrpc import Client
from wordpress_xmlrpc import WordPressPost
from wordpress_xmlrpc.methods import media, posts
from wordpress_xmlrpc.compat import xmlrpc_client

class GetArticlesPipeline(object):
    def __init__(self, mongo):
        self.mongo = mongo

    @classmethod
    def from_crawler(cls, crawler):
        return cls(mongo=crawler.settings.get('MONGO'))

    def open_spider(self, spider):
        self.client = pymongo.MongoClient(self.mongo.get('mongodb'))
        self.db = self.client[self.mongo.get('db')]

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

```

def process_item(self, item, spider):
    item['wp_id'] = 0
    result =
        self.db[item['name']].find_one({'from_url':item['from_url']})
    item['update_time'] = datetime.now()
    if result:
        self.db[item['name']].update_one(
            {'_id':result.get('_id', 0)},
            {'$set':dict(item)})
        item['author'] = result.get('_id', 0)
        item['wp_id'] = result.get('wp_id', 0)
    else:
        res = self.db[item['name']].insert_one(dict(item))
        item['author'] = res.inserted_id
    return item

class UptoWordPressPipeline(object):
    def __init__(self, wp):
        self.wp = wp

    def open_spider(self, spider):
        self.wpc = Client(
            self.wp.get('url'),
            self.wp.get('username'),
            self.wp.get('password'))

    def close_spider(self, spider):
        pass

```

```

@classmethod
def from_crawler(cls, crawler):
    return cls(wp=crawler.settings.get('WORDPRESS'))

def process_item(self, item, spider):
    article = WordPressPost()
    article.title = item['title']
    article.date = datetime.strptime(item['publish_time'], '%Y-%m-%d
        %H:%M:%S')
    article.date_modified = datetime.now()
    article.post_status = 'publish'
    article.terms_names =
        {'post_tag':item['tags'],'category':item['category']}
    article.content = ''.join(item['content'])
    if item['wp_id']:
        result = self.wpc.call(posts.EditPost(item['wp_id'],
            article))
    else:
        item['wp_id'] = self.wpc.call(posts.NewPost(article))
    return item

class CheckPipeline(object):
    def __init__(self, mongo):
        self.mongo = mongo

    @classmethod
    def from_crawler(cls, crawler):
        return cls(mongo=crawler.settings.get('MONGO'))

```



```

def open_spider(self, spider):
    self.client = pymongo.MongoClient(self.mongo.get('mongodb'))
    self.db = self.client[self.mongo.get('db')]

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

def process_item(self, item, spider):
    result = self.db[item['name']].find_one({'_id':item['author']})
    item['update_time'] = datetime.now()
    item['author'] = result.get('author')
    try:
        self.db[item['name']].update_one(
            {'_id':result.get('_id', 0)},
            {'$set':dict(item)})
    except:
        raise DropItem("Article's Wp_id update failed")

```

---

## jiqizhixin.py(机器之心)

---

```

# -*- coding: utf-8 -*-
import os, re
import gzip, time
import scrapy
import pymongo
import configparser
from get_articles.items import GetArticlesItem

```

```

cfg = configparser.ConfigParser()
cfg.read('scrapy.cfg', encoding='utf-8')

class JiqizhixinSpider(scrapy.Spider):
    name = 'jiqizhixin'
    #allowed_domains = ['www.jiqizhixin.com']
    #start_urls = ['http://www.jiqizhixin.com/']

    def start_requests(self):
        urls = ['http://www.jiqizhixin.com/shared/sitemap.xml.gz']
        for url in urls:
            yield scrapy.Request(url=url, callback=self.pre_parse)

    def pre_parse(self, response):
        if cfg.has_section(self.name) and cfg.has_option(self.name,
            'stop_publish_time'):
            self.stop_publish_time = cfg.get(self.name,
                'stop_publish_time')
        elif cfg.has_section(self.name):
            self.stop_publish_time = '2000-01-01 00:00:00'
            cfg.set(self.name, 'stop_publish_time',
                self.stop_publish_time)
        else:
            self.stop_publish_time = '2000-01-01 00:00:00'
            cfg.add_sections(self.name)
            cfg.set(self.name, 'stop_publish_time',
                self.stop_publish_time)
        path = os.path.join(os.getcwd(), 'sitemap_path')

```

```

with open(path, 'wb') as f:
    f.write(response.body)
gfile = gzip.GzipFile(path)
str_cont = gfile.read()
gfile.close()
pattern = re.compile(r'<url>(.*?)</url>')
loc = re.compile(r'<loc>(.*?)</loc>')
lastmod = re.compile(r'<lastmod>(.*?)</lastmod>')
stop_time = self.stop_publish_time
max_time = self.stop_publish_time
str_cont = str_cont.decode('UTF-8')
for label in pattern.finditer(str_cont):
    update_time = lastmod.findall(label.group(1))[0]
    if (stop_time <= update_time) and ('articles' in
        label.group(1)):
        max_time = max(update_time, max_time)
        yield scrapy.Request(url=loc.findall(label.group(1))[0],
            callback=self.parse)
cfg.set(self.name, 'stop_publish_time', max_time)
cfg.write(open('scrapy.cfg', 'w'))

def parse(self, response):
    item = GetArticlesItem()
    item['name'] = self.name
    item['from_url'] = response.url
    item['title'] =
        response.xpath('//h1[@class="article__title"]/text()').get()
    item['author'] =
        response.xpath('//a[@class="article-author__name"]/text()').getall()

```

```

item['publish_time'] =
    response.xpath('//time[@class="article__published"]/text()').get()
item['publish_time'] = re.sub('/', '-', item['publish_time'])
item['publish_time'] = item['publish_time'] + ':01'
item['category'] = ['机器之心', item['publish_time'][0:7]]
item['summary'] =
    response.xpath('//p[@class="article__summary"]').get()
item['tags'] = response.xpath('//span[@class="u-btn--gray
    category__link article__other--tag"]/text()').getall()
x = response.xpath('//div[@class="article__content"]/child::*')
pa = re.compile(r'<mark.+?>')
pb = re.compile(r'<img(.+?)>')
pc = re.compile(r'<p>')
item['imgs'] = [i.xpath('./img/@src').get() for i in x]
item['content'] = ['',]*len(x)
img = ['<figure><img src="", "", "></figure>']
for i in range(len(x)):
    condi01 = x[i].xpath('./text()').getall()
    condi02 = item['imgs'][i]
    if condi01 and condi02:
        item['content'][i] = re.sub(pa, '', x[i].get())
        img[1] = condi02
        item['content'][i] = re.sub(pb, ''.join(img),
            item['content'][i])
    elif condi01:
        item['content'][i] = re.sub(pa, '', x[i].get())
        if not pc.match(item['content'][i]):
            item['content'][i] = '<p>' + item['content'][i] +
                '</p>'

```

```

        elif condi02:
            img[1] = condi02
            item['content'][i] = ''.join(img)
        else:
            pass
    yield item

```

---

## qbitai.py(量子位)

---

```

# -*- coding: utf-8 -*-
import re
import time
import configparser
import scrapy
import pymongo
from get_articles.items import GetArticlesItem

cfg = configparser.ConfigParser()
cfg.read('scrapy.cfg', encoding='utf-8')

class QbitaiSpider(scrapy.Spider):
    name = 'qbitai'
    # allowed_domains = ['www.qbitai.com']
    def start_requests(self):
        urls = ['https://www.qbitai.com/sitemap_post_1.xml',
                'https://www.qbitai.com/sitemap_post_2.xml']
        for url in urls:
            yield scrapy.Request(url=url, callback=self.pre_parse)

```

```

def pre_parse(self, response):
    if cfg.has_section(self.name) and cfg.has_option(self.name,
        'stop_publish_time'):
        self.stop_publish_time = cfg.get(self.name,
            'stop_publish_time')
    elif cfg.has_section(self.name):
        self.stop_publish_time = '2000-01-01 00:00:00'
        cfg.set(self.name, 'stop_publish_time',
            self.stop_publish_time)
    else:
        self.stop_publish_time = '2000-01-01 00:00:00'
        cfg.add_sections(self.name)
        cfg.set(self.name, 'stop_publish_time',
            self.stop_publish_time)
    stop_time = self.stop_publish_time
    pattern = re.compile(r'www\.qbitai\.com\/\d{4}\/\d{1,2}')
    next_time = stop_time
    for article in response.xpath('//url'):
        lastmod = article.xpath('./lastmod/text()').get()
        next_time = max(next_time, lastmod)
        if (stop_time <= lastmod):
            url = article.xpath('./loc/text()').get()
            if re.findall(pattern, url):
                yield scrapy.Request(url=url, callback=self.parse)
    cfg.set(self.name, 'stop_publish_time', next_time)
    cfg.write(open('scrapy.cfg', 'w'))

def parse(self, response):

```

```

item = GetArticlesItem()
item['name'] = self.name
item['from_url'] = response.url
x = response.xpath('//div[@class="article"]')
item['title'] = x.xpath('./h1/text()').get()
_date = x.xpath('./span[@class="date"]/text()').get()
_time = x.xpath('./span[@class="time"]/text()').get()
item['publish_time'] = _date + ' ' + _time
item['category'] = ['量子位', item['publish_time'][0:7]]
item['author'] = x.xpath('./a[@rel="author"]/text()').getall()
item['summary'] = x.xpath('./p[@class="article_summary"]').get()
c1 =
    x.xpath('./div[@class="line_font"]/preceding-sibling::*').getall()
c2 =
    x.xpath('./div[@class="article_info"]/following-sibling::*').getall()
c3 =
    x.xpath('./blockquote[@class="article-blockquote-first"]/following-sibling::*')
if c3 and c1 and c2:
    item['content'] = c3
    dnode =
        x.xpath('./blockquote[@class="article-blockquote-first"]/following-sibling::*')
    for i in range(len(c3)):
        if (c3[i] not in c1) or (c3[i] not in c2):
            item['content'][i] = ''
            dnode[i] = None
if not c3:
    item['content'] = c1
    dnode =
        x.xpath('./div[@class="line_font"]/preceding-sibling::*')

```

```

        for i in range(len(c1)):
            if c1[i] not in c2:
                item['content'][i] = ''
                dnode[i] = None
            item['imgs'] = [i.xpath('..//img/@src').get() for i in dnode if i]
            item['tags'] =
                response.xpath('//div[@class="tags"]/a[@rel="tag"]/text()).getall()
        yield item

```

---

## tencent.py(腾讯云专栏)

---

```

# -*- coding: utf-8 -*-
import configparser
import scrapy
from get_articles.items import GetArticlesItem

cfg = configparser.ConfigParser()
cfg.read('scrapy.cfg', encoding='utf-8')

class TencentSpider(scrapy.Spider):
    name = 'tencent'
    # allowed_domains = ['cloud.tencent.com/developer']
    start_urls =
        ['https://cloud.tencent.com/developer/articles?q=timeline']
    def parse(self, response):
        if cfg.has_section(self.name) and cfg.has_option(self.name,
            'stop_publish_time'):
            self.stop_publish_time = cfg.get(self.name,

```



```

        'stop_publish_time')
elif cfg.has_section(self.name):
    self.stop_publish_time = '2000-01-01 00:00:00'
    cfg.set(self.name, 'stop_publish_time',
            self.stop_publish_time)
else:
    self.stop_publish_time = '2000-01-01 00:00:00'
    cfg.add_sections(self.name)
    cfg.set(self.name, 'stop_publish_time',
            self.stop_publish_time)
x = response.xpath('//div[@class="col-multi-articles"]/ul/li')[0]
self.start_publish_time = []
url = 'https://cloud.tencent.com' + x.xpath('..//a/@href').get()
yield scrapy.Request(url=url, callback=self.parse_01)

def parse_01(self, response):
    item = GetArticlesItem()
    item['name'] = self.name
    item['from_url'] = response.url
    item['title'] = response.xpath('//h1/text()').get()
    item['publish_time'] = response.xpath('..//time/@datetime').get()
    item['category'] = ['腾讯云社区', item['publish_time'][0:7]]
    item['author'] =
        response.xpath('..//a[@class="author-name"]/text()').getall()
    item['summary'] = ''
    x0 = response.xpath('//div[@class="c-markdown
        J-articleContent"]/child::*')
    item['content'] = [i.get() for i in x0]
    pa1 = re.compile(r'class="lazy-image-holder"

```

```

        dataurl="https://ask.qcloudimg.com/(.+?)>')
pa2 = re.compile(r'class="lazy-image-holder"
        dataurl="https://ask.qcloudimg.com/.+?">')
for i in range(len(x0)):
    resu1 = pa1.findall(item['content'])[i]
    resu2 = pa2.findall(item['content'])[i]
    if (len(resu1) == len(resu2)) and resu1:
        for j in range(len(resu1)):
            re.sub(resu1[j], resu2[j], item['content'][i])
item['tags'] =
    response.xpath('nav/a[@class="col-tag"]/text()').getall()
item['imgs'] = [i.xpath('..//span/@dataurl').get() for i in x0]
if item['publish_time'] >= self.stop_publish_time:
    yield item
    last_article =
        response.xpath('//a[@hotrep="community.column.article-detail.prev"]/@href')
    if last_article:
        url = 'https://cloud.tencent.com' + last_article
        yield scrapy.Request(url=url, callback=self.parse_01)
else:
    cfg.set('tencent', 'stop_publish_number',
            self.start_publish_time)
    cfg.write(open('scrapy.cfg', 'w'))

```

---