护城项目

数据质量问题解决方案

版本号：V1.1

|  |  |  |  |
| --- | --- | --- | --- |
| 拟制: | 丁明会,丁永强,付猛林 | 日期： | 2016/4/16 |
| 审核: | 何俊 | 日期： | 2016/4/16 |
| 批准: |  | 日期： |  |

**文档修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **修订版本** | 日期 | **作者** | 修改内容描述 | **备注** |
|  |  |  |  |  |
| V1.1 | 20160419 | DingMingHui | 添加控制器详细流程图 |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[1 引言 4](#_Toc6748)

[1.1 目的 4](#_Toc19918)

[1.2 预期读者 4](#_Toc14230)

[2 现有问题和解决思路 4](#_Toc30101)

[2.1 完整性 4](#_Toc29319)

[2.1.1 概述 4](#_Toc7185)

[2.1.2 技术解决思路及措施 4](#_Toc11968)

[2.1.3 技术瓶颈问题 4](#_Toc14207)

[2.1.4 资源问题 4](#_Toc20172)

[2.2 及时性 5](#_Toc22644)

[2.2.1 概述 5](#_Toc24643)

[2.2.2 技术解决思路及措施 5](#_Toc24627)

[2.2.3 技术瓶颈问题 5](#_Toc18744)

[2.2.4 资源问题 5](#_Toc1758)

[2.3 正确性 6](#_Toc2616)

[2.3.1 概述 6](#_Toc6136)

[2.3.2 技术解决思路及措施 6](#_Toc318)

[2.3.3 技术瓶颈问题 6](#_Toc16393)

[2.3.4 资源问题 6](#_Toc15684)

[2.4 数据处理 6](#_Toc23661)

[2.5 规范性管理 7](#_Toc12120)

[2.5.1 概述 7](#_Toc1682)

[2.5.2 技术解决思路及措施 8](#_Toc11617)

[3 爬虫系统架构 8](#_Toc7730)

[3.1 系统工作流程图 8](#_Toc7153)

[3.2 模块功能 14](#_Toc29260)

[3.2.1 http 14](#_Toc32737)

[3.2.2 proxy 15](#_Toc26341)

[3.2.3 对http的监控 15](#_Toc32131)

[3.2.4 提供其他接口等 15](#_Toc9690)

[3.2.5 流程图 15](#_Toc15024)

[3.3 相关技术 15](#_Toc6992)

[3.3.1 爬虫整体结构 16](#_Toc509)

[3.3.2 各模块设计 17](#_Toc17021)

[3.3.3 爬虫实现 18](#_Toc14553)

[3.3.4 典型的爬虫开发流程 20](#_Toc11744)

# 引言

## 目的

通过编写方案，解决现有数据完整性、及时性、正确性问题，指导项目实施。

## 预期读者

(1).护城项目爬虫开发人员、产品经理、项目管理人员。

# 现有问题和解决思路

解决方案主要从数据的完整性、及时性、正确性 三个方面进行设计。

## 完整性

### 概述

完整性 主要解决库里面企业抓取量的问题。

### 技术解决思路及措施

<<数据及时性及完整性方案>>确定提高数据完整性。

核心点如下：

1. 通过购买企业名单增加企业名录灌入抓取。
2. 通过抓取相关信息获取的企业名单增加企业名录，灌入抓取。
3. 优化代理服务器逻辑,对代理服务器代码进行评审,优化性能和逻辑,付猛林主要负责代码的调整优化

### 技术瓶颈问题

1. 目前可以通过关键字，蔓延注册号提高覆盖率
2. 蔓延注册号存在大量无效值，会降低效率

### 资源问题

提高服务器带宽和性能，硬件方面提高程序并发执行速度，辅助提高蔓延注册号的抓取次数

缺乏新的公司种子

## 及时性

### 概述

及时性的目标要保证工商数据每一个更新1轮。

### 技术解决思路及措施

抓取量优化达到目前自建代理ip和固定访问次数 利用率最高,个别省份由于公司量超过这个抓取值,暂时不能保证每月抓取一轮,这类特殊的在考虑专门优化.增加代理

数据更新及时性,实现目前爬虫数据入库后的实时索引,实现爬虫数据的实时统计,便于bug公司的批量的搜索快速的解决异常公司数据。

### 技术瓶颈问题

考虑前提要持续更新

1. 北京，广东，深圳信用，因为基数大，封锁ＩＰ比较严格，网站服务器性能限制，难以达到月更一次

比如北京，总量预计200W+, IP抓取超过30次就会封锁数月，1000IP 理论数据每月3W，实际经过多次调整，2W-3W为平均水平。

广东省数据总量300w+，100进程，一次请求返回需要10多秒钟，拿到一次完整数据要2分钟以上。

每天3-5万为平均水平。 加大请求次数会导致对方网站无限变慢。

1. 吉林省份有较严的反爬机制，自身网站下午会故障，持续10小时左右，抓取不稳定
2. 宁夏，浙江，江苏，反爬机制复杂，封锁ＩＰ严格，影响抓取量

目前已经优化过一轮，数据暂时达到要求。

1. 偶尔个别省份网站会关闭(今天广东网站不能打开--20160414)

优化方案：

1.主要是清理爬虫代码，减少不必要的请求

2. 调整代理逻辑和配置，提高代理使用率，能抓取数据而又不被封

### 资源问题

增加代理ＩＰ

存储服务器稳定性和效率问题

## 正确性

### 概述

正确性主要解决 数据源与爬下来的数据的匹配度，比如错位等。

### 技术解决思路及措施

 数据完正确达到入库的数据与网站100%匹配

1. 需要加强爬虫的异常逻辑测试，解决代理不稳定造成的数据字段丢失问题

 2.目前需要投入大量人力和时间排查爬虫代码，清理代码逻辑需要保证代码能力较强人员至少3人专职

### 技术瓶颈问题

可以做到和页面数据一致

### 资源问题

需要人手排查老代码逻辑和测试

## 数据处理

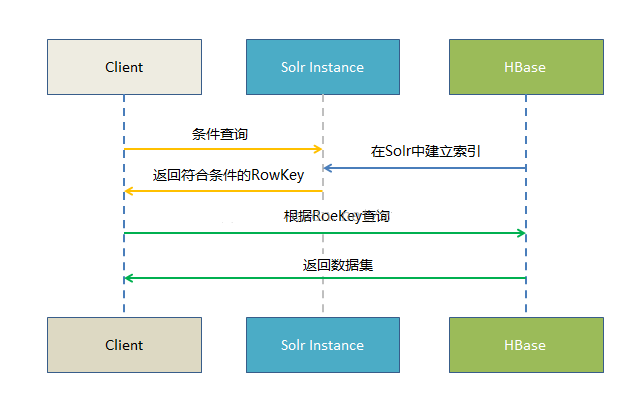
目前爬虫抓取数据以后直接就进入到hbase由于没有实时的统计和跟踪造成数据的统计和测试需要很复杂的步骤来实现，数据的维护性很差，不能满足及时处理的需求。

解决思路:

爬虫增加数据实时抓取量统计，并对数据根据常用统计需要实时创建统计索性方便根据省，公司名，抓取时间对数据进行查询统计

再次基础上爬虫实现抓取和解析分离，发现解析错误可根据情况重新快速解析不用再次抓取加快维护速度

**通过采用入库数据对接solr搜索引擎**

****

1 修改爬虫入库程序,增加一个入库数据消息队列,数据格式[id,company\_name,do\_time,type]

1 每天入库统计和bug公司查询从此solr中检索来实现

2 测试异常公司最近相同省份更新的公司名单统计出来,根重新抓取修复数据异常

## 规范性管理

### 概述

爬虫重构计划解决目前爬虫代码混乱规范不统一的问题,为以后爬虫的更新和维护提供统一的方案,能适应后续对数据质量和抓取量的优化调整

完善开发文档的输出,解决目前研发开发交接困难的问题

解决思路:

1. 完善需求文档便于后来人员了解做的是什么怎么做
2. 完善设计文档和开发强制规范，便于后续人员理解开发结构原理和逻辑，编写统一的接口对接功能
3. 需要尽早并且经常性地进行代码重构，随着需求的增加代码的功能调整，代码会逐渐的复杂，保持整洁的代码胜过巧妙的代码

### 技术解决思路及措施

补全qyxx网站的爬取需求,需要合并新的爬取需求要整合最近提出来的抓取和存储规范,便于合理使用代理调度资源

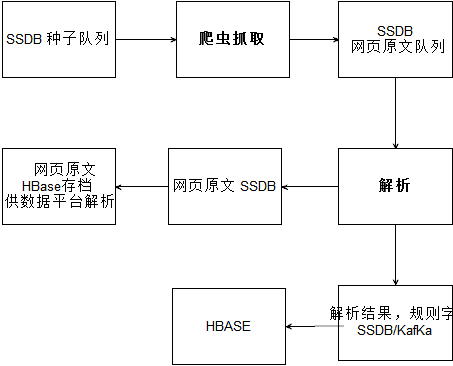
2 爬虫重构需要做详细设计,规范开发代码规范严格要求注释与关键节点的说明.

3 爬虫代码评审,开发的代码要做代码的评审,对规范和注释严格要求,并定期要求开发人员轮流讲解各自代码的逻辑,做到代码的可读性

4 定期对一段时间的产品需求进行汇总(暂定一个月)全体数据组参与会议,提出能否将重复的工作工具化开发,并考虑是否需要迭代爬虫部分逻辑代码.发起自身的研发需求,提高工作效率和数据组自身的处理能力

# 爬虫系统架构

## 系统工作流程图



整体结构分3 大部分

1. 数据存取部分 ，主要利用SSDB，KafKa , HBASE完成爬虫运行过程中的数据流转
2. 爬虫抓取部分，主要功能部分，涉及API 最多，逻辑也最复杂
3. 数据解析部分，主要功能部分，完成页面数据解析，解析逻辑较为复杂

下面就爬虫抓取部分做进一步说明：

此部分牵涉到的技术要点主要有

1. HTTP

主要针对需要抓取的连接，组织请求，获取返回数据

1. 代理

针对不同网站，指定不同的代理使用逻辑，规避反爬措施

1. 验证码解析

标准接口，主要利用图像识别技术，解析验证码图片

1. Log 管理

针对目前log结构比较混乱的情况，进行统一整理，提供标准log

1. 文件存储

以文本形式存放抓取过程中需要的文件在本地

1. DB存取

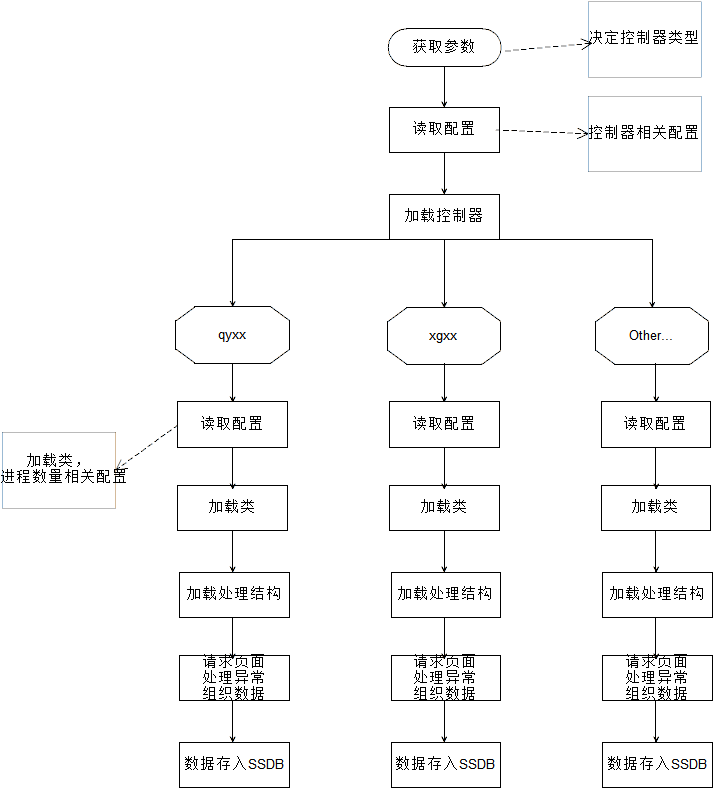
指定统一接口，。利用SSDB/ Kafka存取数据.

Hbase 数据需要单独存取

1. 种子管理

严格控制逻辑，保证一个种子对应一个结果，不能丢失或则判断出错

结构图解



步骤解析：

1. 抓取部分 主要有HTTP 请求+代理处理模块， 控制器模块，相关类和方法定义模块 以及其他公用lib
2. 利用python 自省和设计模式的思想，把处理部分 抽象成统一的步骤，由控制器来统一管理。

考虑到相关信息和其他类型爬虫的接入，设计多个类型的控制器，降低模块耦合性，辅助配置文件，提高扩展性。程序运行接受的参数即为控制器类型。

1. 配置文件1内容主要包括该控制器要启动的爬虫类型，进程数量，DB信息等运行相关信息。
2. 配置文件2 主要包含单独省份（以企业信息为例）的私有信息

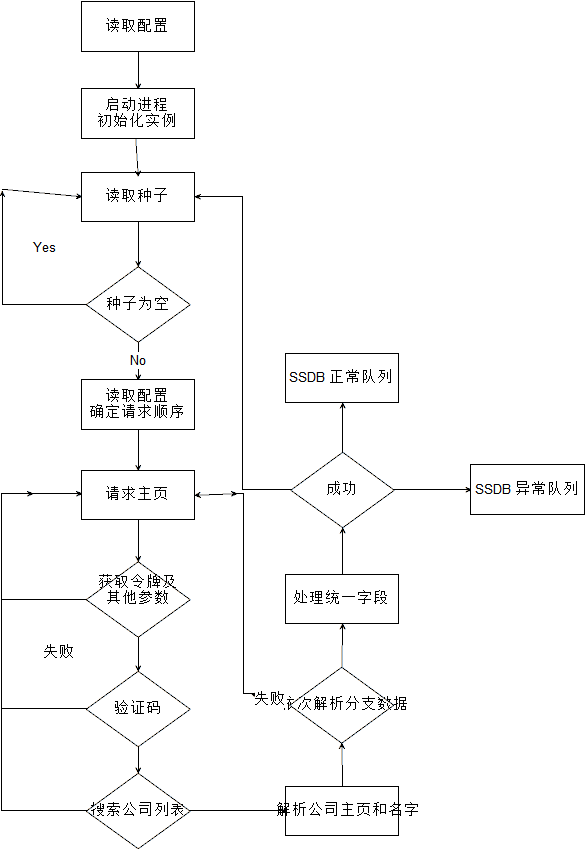
1）省份主页连接

2）爬虫进程数量

3）爬虫处理顺序

1. 请求页面开始归属于抓取流程部分，为抓取的核心， 下面将对抓取流程部分详细说明

抓取流流程图



说明：

1. 设计多进程控制抓取，进程数目由配置文件统一规定
2. 种子队列部分设计单独模块判定种子的类型，内容和多样性
3. 不同省份设置单独的配置文件，用来确定方法的调用次序
4. 获取主页，获取令牌和参数，获取验证码步骤出现异常，需要从头重试
5. 验证码设置单独打码模块，不在作为类实例方法
6. 抓取过程中会涉及到部分解析步骤，例如解析暗链接，公司名称等
7. 分支数据包括 gdxx, baxx,bgxx 等，进行严格管控，不无谓抓取
8. 请求后的网页原文会加一些公共字段供解析，暂定结构为

{

“name”:xxx

“type”:xxxx

“dotime”:xxx

“jbxx”:{}

“gdxx”:{}

“baxx”:

“bgxx”:{}

“fzjg”:{}

“xzcf”:{}

“version”:{}

“url”

“post paras”

}

1. 入库前进行版本判定，版本不对数据，单独入库或不入库，取出公司名称，重新抓取

控制器部分设计

功能要点如下

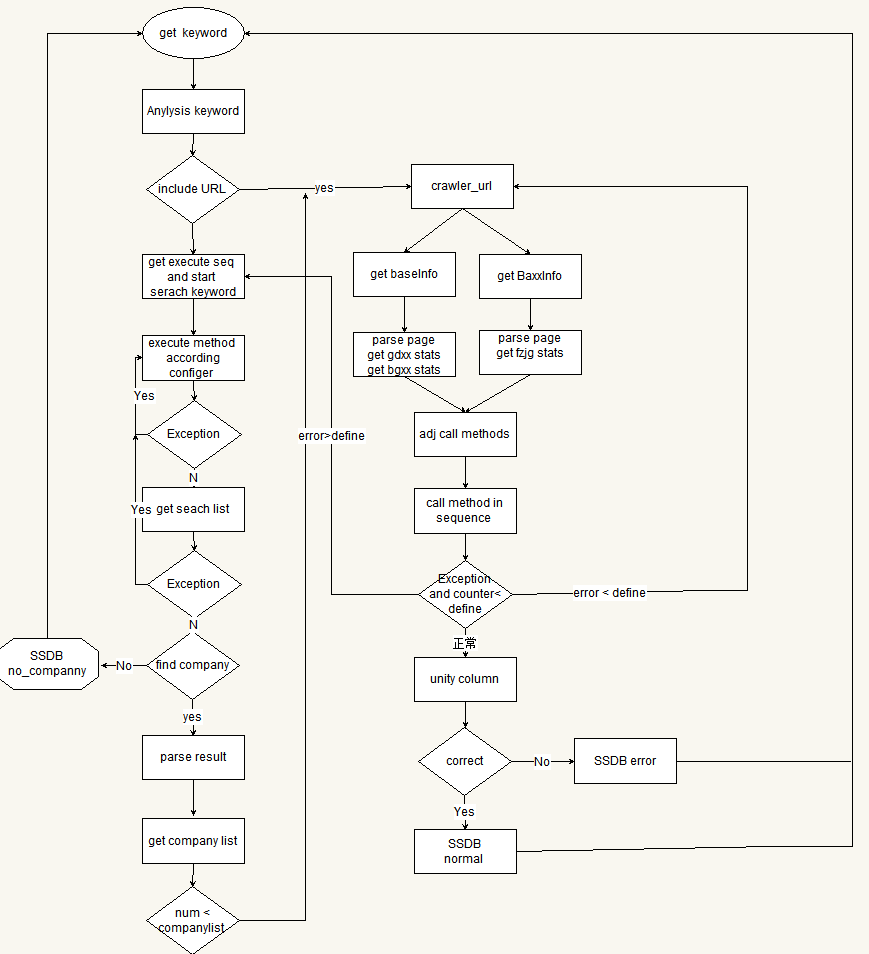
1. 临时队列，保存从队列取出正在抓取的种子，避免重启爬虫造成的数据丢失

每个省份单独维护一个省份名+tmp的set, 种子抓取完毕，代码控制移除

1. 种子类型判断，选择调用函数

判断种子是普通公司名字还是dict格式，包含url走直接抓取url 流程

1. 确定抓取顺序，url->注册号->名字->信用代码
2. 抓取逻辑部分见下面流程图
3. 状态码为9版本，不在入库，而是单独记录
4. 抓取方法有2个接口，一个以搜索方式抓取关键字一个直接抓取url
5. 方法列表从配置列表或配置类里面获得，动态调用
6. 数据读取完毕，设置统一字段模块，用来规整字段



数据解析部分

解析部分结构和抓取部分对应，区别在于下面几方面

1. 业务方面抓取部分主要封装请求，处理网络异常，解析模块主要处理网页原文不同的格式
2. 抓取部分需要大量进程弥补抓取过程中遇到的网络超时和等待情况，解析部分则不需要太多
3. 抓取部分需要严格的重试机制，避免网络异常引起的漏抓。解析部分不需要

此部分牵涉到的技术要点主要有

1. Log 管理

针对目前log结构比较混乱的情况，进行统一整理，提供标准log

1. 文件存储

以文本形式存放抓取过程中需要的文件在本地

1. DB存取

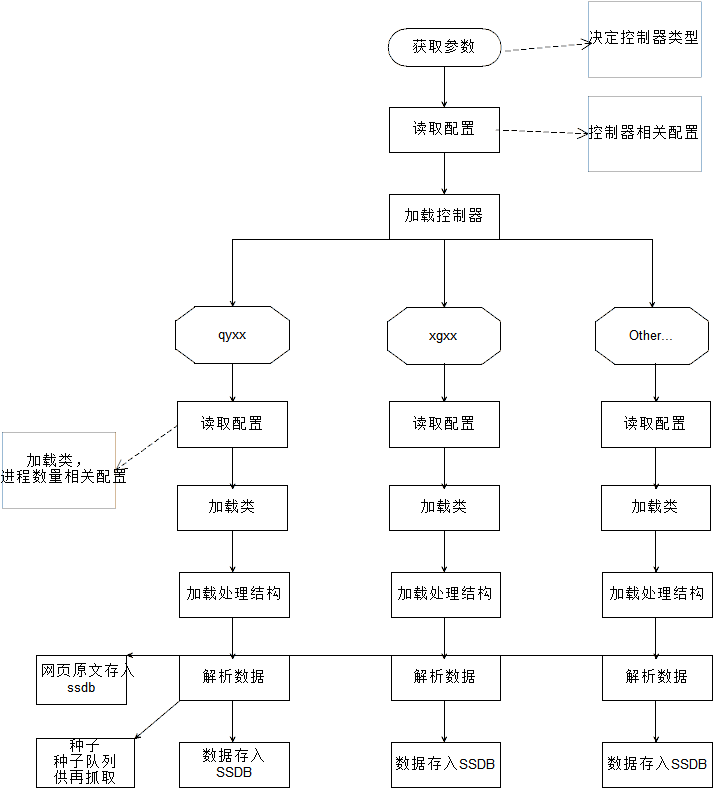
指定统一接口，。利用SSDB/ Kafka存取数据.

Hbase 数据需要单独存取

1. 种子管理

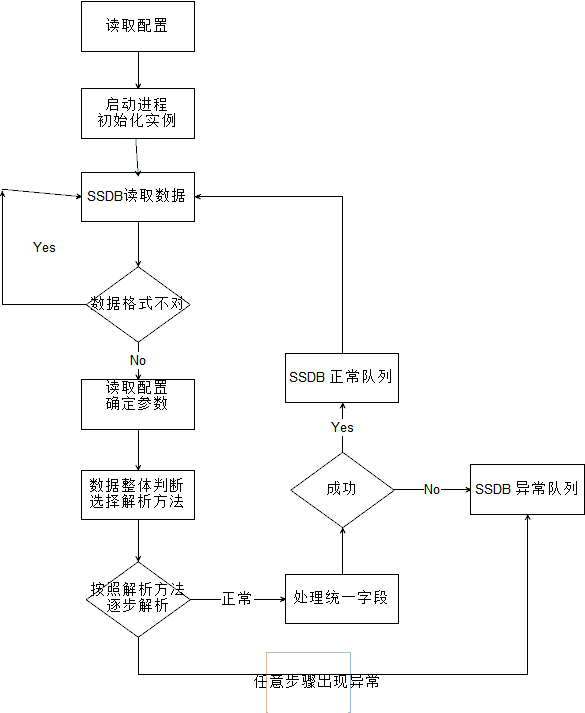
考虑新加队列维护专门的种子

流程图如下



同样，解析数据模块也是整个流程的核心部分

流程如下



## 模块功能

为了规范下载过程，作为爬虫的核心，特将下载和代理封装在一起，方便管理和减少代理重复，该模块包含http下载和proxy代理更换，减少爬取数据流程中对代理的处理，简化处理流程，清晰逻辑，其中http主管下载，proxy负责代理的更新，怎么更新。

### http

该方法负责接受http的参数，包括url，cookies，proxy，headers，data，func(传入函数)，retry(错误后重试次数)，wait\_time（每次访问的间隔时间）,\*\*kwargs等等参数，自建包装header，将此次产生的set\_cookie和上次的cookie封装在一起，然后通过requests包的get和post方法获取下载页面，获取response对象，通过twisted将每次访问derfer后形成异步访问，或者通过gevent协程包jion后，加快访问数据，封装func\_time函数，计算每次访问所需的时间

### proxy

此代理为一个http接口，在第一次获取首页或者cookies时候，筛选出正确的验证码，在下面的获取页面中是否重试时是否更换代理，更换自建还是非自建的代理等

### 对http的监控

对每次http请求一个监控程序，如是否成功，失败次数，连续失败次数，在筛选优质代理时候，连续更换次数，下载次数，在哪一步失败率比较大，等等，方便后面对网站进行分析，

### 提供其他接口等

提供其他外部接口，预留一个接口等

### 流程图



## 相关技术

1、封装requests的访问网络的模块

2、更换代理的逻辑

3、通过twisted或者gevent封装http下载，形成异步下载模式

4、对访问返回错误的处理

### 爬虫整体结构

爬虫整体结构如下图所示，分为爬虫基类，爬虫基础模块库和基础模块类库。



#### 基础模块类库

提供通用模块的封装并为爬虫基础模块库和爬虫基类提供支持，提供诸如Proxy、Browser、Log、Monitor、DB、Mail、验证码识别等功能。

#### 爬虫基类模块库

提供所有爬虫所通用的基础模块，例如读取首页获取令牌，读取验证码，获得搜索列表，读取公司基本信息等。

#### 爬虫基类

通常为某一垂直领域的爬虫基类，例如企业信息的爬虫基类为QyxxCrawler，相关信息的爬虫基类为XgxxCrawler等。该基类可为子类提供运行时需要的所有依赖，例如运行时的状态、中间结果、通用功能模块支持等。

爬虫基类通过对引入的爬虫基础模块进行裁剪，提供典型的抓取流程实现，使爬虫子类可以直接使用。

#### 爬虫子类

负责具体垂直领域下的具体站点爬取，例如对于企业信息下的北京企业信息站点的爬取。由于经过爬虫基础模块和爬虫基类的两次抽象和封装，爬虫子类的实现将变得简单且易于维护。

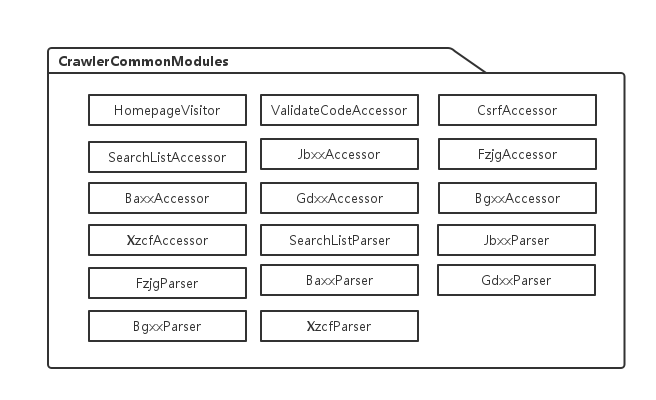
### 各模块设计

#### 爬虫基础模块库

爬虫基础模块库包结构图如下图所示，该模块库可为不同垂直领域的爬虫提供通用方法支持，但因其原型抽取于企业信息爬虫系统，因此其通用性需要随着其他垂直领域爬虫项目的推进不断抽象和优化。

本模块包设计的指导思想是高内聚、低耦合、高复用，将不同的处理逻辑切分为独立的处理单元，每个处理单元为一个独立的模块，各个模块之间完全解耦，模块之间不用关心其他模块是否存在，关注并处理好指定的处理逻辑。每个模块在接口不变的情况下，可以各自独立优化和使用，各个模块可在复用过程中不断优化和打磨，沉淀为可靠的基础组件。

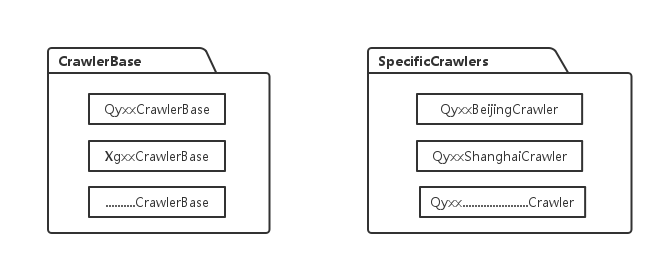
该模块包的基本组成包括验证码识别、令牌获取、搜索列表获取和解析，基本信息获取和解析等。



理想情况下，各个领域垂直爬虫可通过使用该模块库，快速组装出满足业务需要的爬虫系统。

#### 爬虫基类/子类

爬虫基类和子类Package中类的组织结构如下图所示。



爬虫基类Package中包含各个领域垂直爬虫系统的基类，例如企业信息信用公示领域爬虫基类QyxxCrawlerBase、相关信息领域爬虫基类XgxxCrawlerBase等。

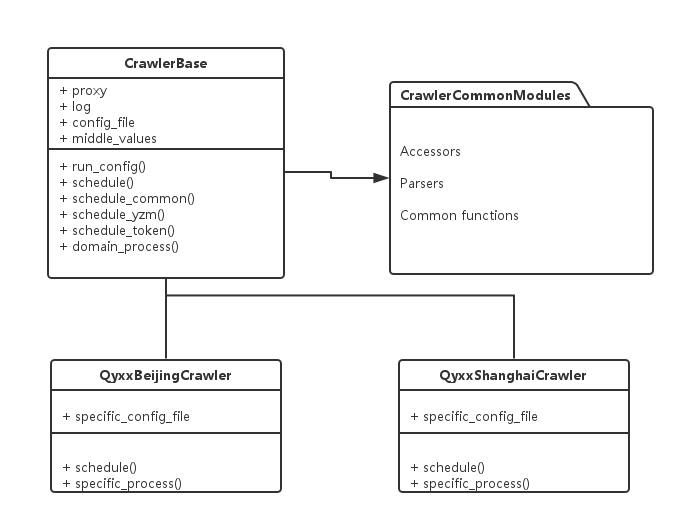
爬虫基类通过组合使用爬虫基础模块库中的不同模块，以及根据领域模型和业务需要，封装通用功能模块和逻辑处理流程，为子类提供有效支持。

爬虫基类提供几种逻辑处理流程的典型实现，例如通过组合HomepageVisitor、SearchListAccessor、JbxxAccessor等通用模块，实现一个可绕过验证码的通用处理流程；亦可通过组合ValidateCodeAccessor、SearchListAccessor、JbxxAccessor等通用模块实现一个无需访问首页，但需要通过验证码验证的处理流程。

爬虫子类按不同垂直领域分不同的包，上图为企业信息垂直领域包的组织情况，每个具体的Crawler对应一个具体网站的爬虫。

### 爬虫实现

爬虫类设计如下图所示。



#### 爬虫基类

包含的属性如下：

proxy、log等提供基础功能支持的功能对象

config\_file 持有运行配置文件

middle\_values 持有各种在运行期产生的中间状态、中间值等

包含的方法：

run\_config 提供通过读取配置文件方式运行特定crawler

schedule 调度逻辑处理流程，实现抓取、解析、存储等

schedule\_common 提供通用的逻辑流程处理实现

schedule\_yzm 提供需要处理验证码的流程裁剪

schedule\_token 提供需要令牌的流程裁剪

domain\_process 代表该领域内通用方法的一组实现

#### 爬虫子类

属性：

specific\_config\_file 持有该子类运行需要的配置文件

方法：

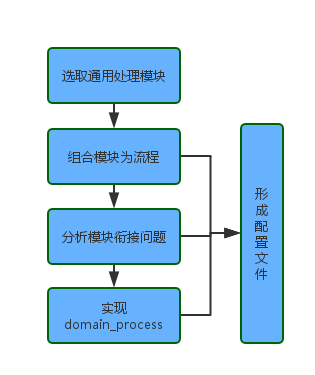
schedule 该子类的具体调度策略

specific\_process 代表该站点特有处理方法的一组实现

### 典型的爬虫开发流程

#### 开发CrawlerBase

具体流程如下图所示



1. 从CrawlerCommonModules模块库中选取适合本领域的模块

2. 根据业务需要和网站实际访问路径和要求组合处理流程

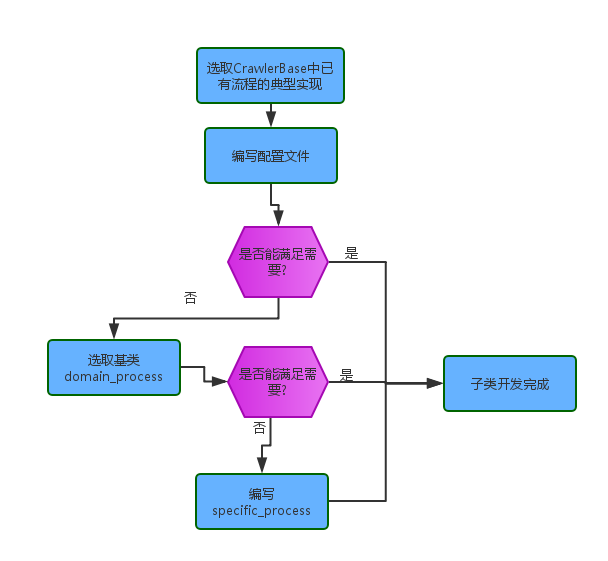
3. 对某些模块间不能衔接或达不到要求的部分进行分析

4. 实现该领域内通用方法domain\_process

5. 在组合功能模块的过程中形成通用的运行配置文件

#### 开发Crawler子类

子类开发流程如下图所示



1. 查看CrawlerBase中是否已有适合的流程实现，例如通用处理流程schedule\_common, 可处理验证码的schedule\_yzm等

2. 参照CrawlerBase中的通用运行配置文件形成该站点的配置文件

3. 若经过测试已能满足要求，则仅需提供配置文件，无需编写任何逻辑代码。

4. 若无法满足要求，则需要查看父类中提供的一组领域通用方法domain\_process，检查其中是否有可以支持的功能，将其纳入模块执行流程中

5. 若4依然无法满足要求，则需要实现特定站点下的specific\_process方法，以满足个性需求

#### 运行方式

运行方式可同时支持两种：

1. 复写父类空方法schedule实现具体调度

2. 通过父类或子类对象，传入配置文件进行执行

方法1可在部署和上线时充分利用面向对象的多态特性。而方法2则可以轻量级实现爬取，无需实现特定站点子类，而仅通过传入配置文件给父类即可实现。

#### 维护

从整个开发过程来看，所有通用方法尽可能向上抽象，各个子类的负担大大减轻，子类必需实现的仅有配置文件，从而可以快速提供领域内不同站点的爬虫，同时由于核心处理逻辑的质量在通用层进行保障，各站点爬虫出现逻辑问题的概率也大大降低。

在使用过程中发现的通用模块相关问题，一处修改即可，而且经过长期使用和打磨，会使通用模块不断趋于可靠和稳定