# 网络爬虫服务化方案建议书

#### 一 引言

网络爬虫作为一种强有力的数据获取手段，迄今为止被运用到各个方面。本团队需要相关行业数据作为支撑，广泛收集互联网数据势在必行。目前团队中爬虫应用主要存在以下问题：

1. 数据收集所涉及的目标网站未统一管理，且各个爬虫应用分散，不能集中管理目标url。
2. 爬虫应用大都是各自为营，相关定时任务等都是在应用中自行设置，任务无法有效统筹和统一管理。
3. 爬虫应用大都是按照需要在不同平台上部署和运行，应用的运行情况无法监控，服务器资源浪费，日志零散，运行状态难以查询。
4. 爬虫数据管理被放置到关系型数据库按照一定的规则命名表，数据零散，维护成本高，没有统一管理和检索，数据检索难。
5. 爬虫应用在不同平台部署，各个应用的配置都是各自修改，当应用量上升，难以大规模维护。
6. 单一服务无法统筹治理，生产和消费服务模糊。

爬虫服务化架构能够有效解决以上问题，各个组件的相互调用，内部功能实现解耦，能够更加全面有效的对数据进行收集，存储和检索。

#### 二 网络爬虫分层架构

按照分层模型，主要基于各个层次提供的功能和服务划分：



* 最底层是硬件设备：相关软件和网络带宽作为支撑，这部分作为网络爬虫的基本设施，主要提供强有力的硬件，资源支撑。提供足够的内存，硬盘，cpu，带宽和IP资源。
* 组件和服务层 ：主要提供爬虫相关的中间件：IP动态资源池，url资源池，任务调度，定时任务配置管理，应用监控，配置中心，服务注册中心，消息队列等基础组件服务。

（1）配置中心主要承担分布式上层应用中的全局配置，基础属性配置。

（2）服务注册中心系统组件服务注册，有效管理服务可用性。

（3）应用监控中心主要管理上层应用的运行情况，是否发生相关异常，资源使用情况是否达到系统极限等。

（4）消息队列可以像不同服务下发订阅相关实时消息，如服务之间的部分数据传输。

（5）任务调度管理主要涉及分布式环境下个服务器资源有效利用，根据服务器状况合理分配任务，相关定时任务可以在任务调度中进行管理。

* 爬虫应用解析和网关层： 该层提供爬虫解析的具体逻辑，相关url和属性可以在服务层中调用获取。网关主要提供各种服务管理可视化界面展示，url管理，配置管理。
* 存储层：结构化，半结构化数据在分布式数据库HBase中存储，非结构化数据如(咨询，图片)使用HDFS文件系统存储和ES等。部分热点数据添加文本索引，利于快速检索。有关存储规则，表设计，索引设计等如何应对大批量写入和读取等策略设计。



爬虫服务化架构图

#### 三 爬虫组件服务层设计

服务层提供基础组件和相关服务，主要涉及网络爬虫中的初始化，任务，资源，配置和任务管理。

##### 硬件，网络配置

1. 硬件资源：

网络爬虫需要部署在稳定的服务器之上运行，涉及到启动爬虫进程占用一定的内存和CPU资源，按照爬虫分布式部署和存储的原则，拟规划4台虚拟服务器部署爬虫服务和存储检索服务。相关配置如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 机器编号 配置项 | 内存 | CPU | 硬盘 | 用途 |
| D1 | 64GB | 8核 | 1T\*2 | mongodb数据存储 |
| D2 | 64GB | 8核 | 1T\*2 | mongodb数据存储 |
| D3 | 64GB | 8核 | 1T\*2 | Mongodb数据存储 |
| C1 | 64GB | 8核 | 1T | 大数据清洗计算节点，爬虫应用 |
| C2 | 64GB | 8核 | 1T | 大数据清洗计算节点，爬虫应用 |
| C3 | 64GB | 8核 | 1T | 大数据清洗计算节点，爬虫应用 |
| C4 | 64GB | 8核 | 1T | 大数据清洗计算主节点,Redis,Mysql |

1. 网络资源：

网络爬虫如果涉及到长时间大规模抓取，涉及到相关政策法规，是否需要专

网或者专门的网络出口。

1. 代理IP

在抓取目标网站数据时，时常会有IP地址或者域名被封的情况，这是一种常见的反爬虫手段,用于处理大量非正常请求。目前网络上主要采用动态切换IP,来伪装防止被封，互联网上有很多厂商提供IP资源服务，优质的IP资源循环使用率高，可作为内部资源循环利用。

##### IP资源池组件

IP资源获取的两种方式：一种是通过网络上抓取免费IP资源。这种方式获取的IP资源质量低，有效率不到10%，IP时效短；另一种是通过相关厂商购买，购买后按照接口方式获取IP。这种方式获取的IP量非常大，可用率高。

1. IP资源的统筹方式：

大量IP获取后需要按照有效的方式管理，作为一种服务，微服务是一个很好的解决方案，IP资源的增删改查可以独立作为一个服务运行。稳定有效的IP资源，快速有效的存取，IP资源池服务作为微服务注册到服务注册中心，由服务注册中心负责IP资源服务的有效管理。

IP池采用Redis队列存储，可动态添加，取出，同时设置备用队列，解决主队列中的IP资源取出后不能复用的问题。

该微服务框架拟采用springboot结构，按照微服务的方式构建，项目提供两方面功能，一方面可以对外抓取国内网免费代理网站的IP资源，另一方面提供相关厂商购买提供的IP资源。

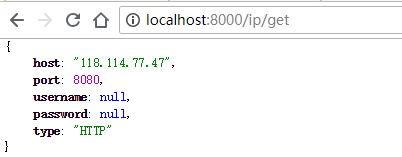
IP资源池架构如下：



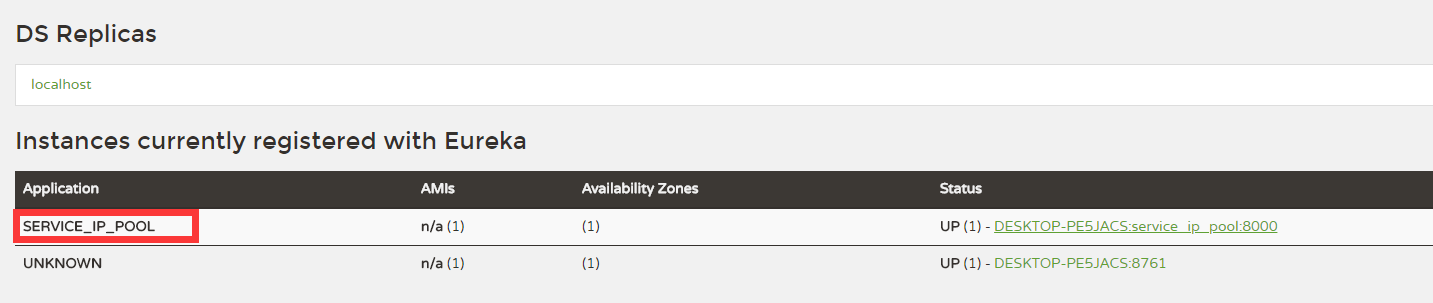
在进入Redis队列之前使用sockets测试代理有效性，有效的会放入资源池。

拟设计Redis队列为去重的key-value（队列）,设计key为ippool,value为序列化的IP对象。

获取该IP可传入必要参数，可以单个获取，也可以批量获取。通过REST方式对外提供。



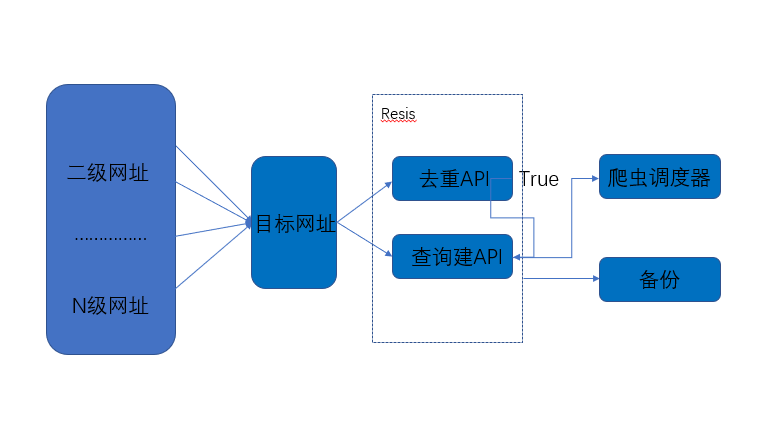
IP池服务注册到服务注册中心，可以由服务注册中心查看服务的状态



##### URL资源池

资源池目标：创建规范化URL去重策略，防止重复抓取。实现URL的可查询性、可追溯性、可量化性。

URL资源池架构图：



1.**去重键**

底层用Redis的HeperLogLog数据类型，储存所有最下级网址（内容页）。对外使用REST方式提供API。可传入一个或多个判断是否重复的Url，返回True或False的列表。Redis HyperLogLog 是用来做基数统计的算法，HyperLogLog 的优点是，在输入元素的数量或者体积非常非常大时，计算基数所需的空间总是固定的、并且是很小的。

在 Redis 里面，每个 HyperLogLog 键只需要花费 12 KB 内存，就可以计算接近 2^64 个不同元素的基 数。这和计算基数时，元素越多耗费内存就越多的集合形成鲜明对比。但是，因为 HyperLogLog 只会根据输入元素来计算基数，而不会储存输入元素本身，所以 HyperLogLog 不能像集合那样，返回输入的各个元素。它的优势是内存消耗小，计算速度快。

2.**查询键**

底层使用Redis，按照不同类别、地区创建多个key:List，用于URL查询。对外提供REST方式的API，传入Redis数据库语句，传入类型为String，并使用BGSAVE主动保存数据到磁盘，作为备份。其读写速度快。

##### 任务管理

任务管理包含任务添加，任务启动，任务执行，任务状态，任务调度，任务删除，任务异常管理。爬虫应用需要定时抓取，对目标站点的定时，执行情况进行统筹管理，需要一个分布式任务调度系统。

项目拟采用唯品会开源的分布式任务调度系统saturn，该开源项目目标是取代传统的Linux Cron/Spring Batch Job/Quartz的方式，做到全域统一配置，统一监控，任务高可用以及分片。

分布式任务调度系统逻辑架构：



###### 执行结点

负责作业的触发（定时），作业执行，结果上报，日志上报，告警上报，监控日志写入等功能。可独立运行在业务服务器，也可与业务代码运行在同一个JVM。 使用java开发，提供jar包和可运行的工程两种方式供业务方使用，是业务作业接入saturn最主要的组件。

###### 控制台

负责作业的统一配置，包括作业添加、删除，作业属性配置，作业状态查看，执行日志查看，执行结点监控等功能。 控制台单独部署，提供WEB应用给全域共用，业务接入方根据申请的权限控制对应的业务作业。

###### 作业分片调度器

Saturn的”大脑“，其基本功能是将作业分片指派到执行结点。通过调整分配算法和分配策略，可以将作业合理地安排到合适的执行结点，从而实现HA，负载均衡，动态扩容，作业隔离，资源隔离等治理功能。 作业分片调度器为后台程序，单独部署；它是公共资源，所有域共用同一套作业分片调度器。接入作业后，会自动接受作业分片调度器的调度。

（1）任务调度管理：

分布式任务调度系统可以分别执行shell作业和java作业，python可以按照相关脚本配置方式，java按照saturn的相关接口即可完成对相关任务的开发。

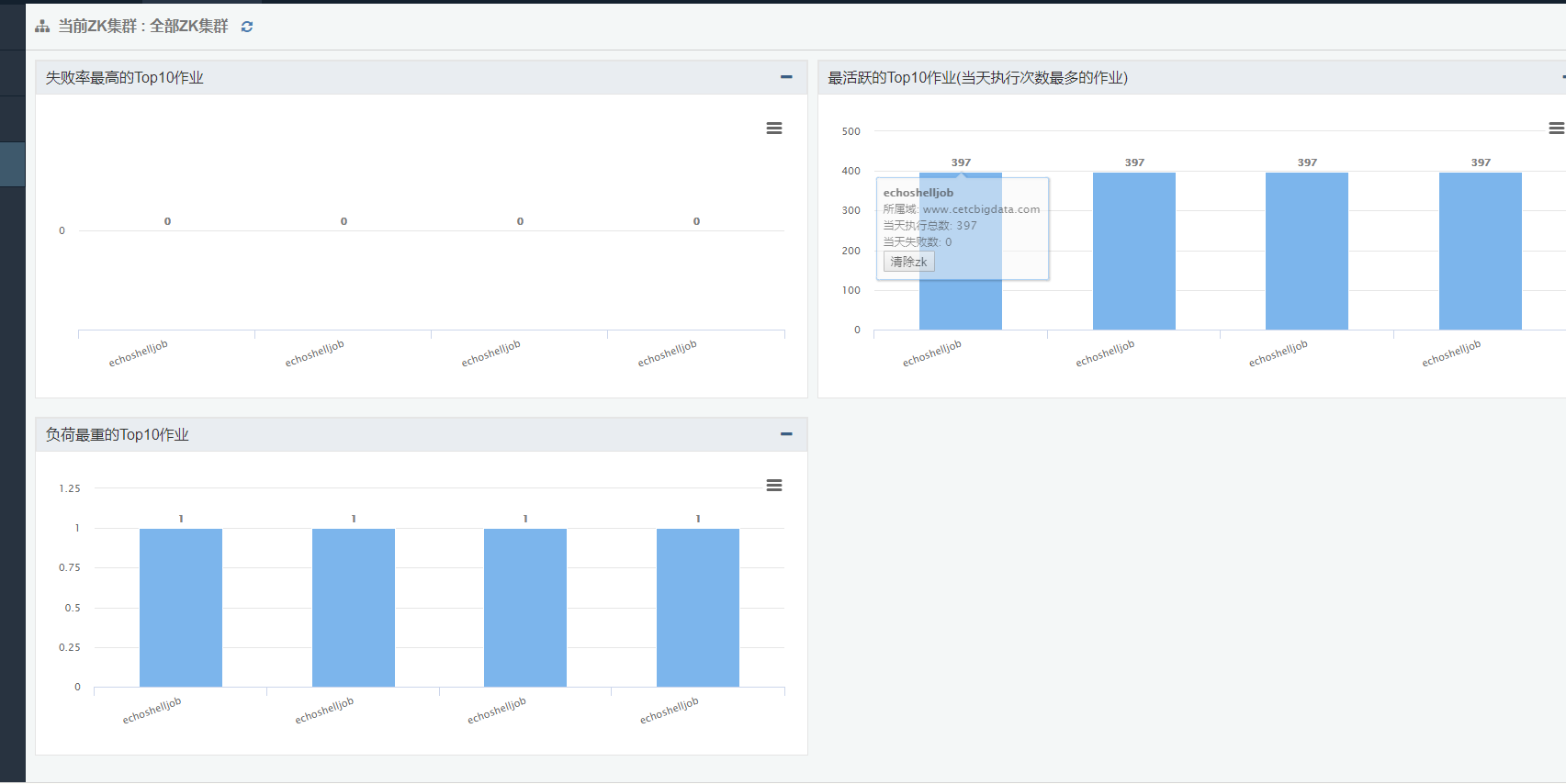


Saturn可以按照cron的方式设置定时任务，触发执行。通过设置任务的优先级可以定义哪个任务先执行。



通过设置任务的执行属性还可以多任务分片执行。

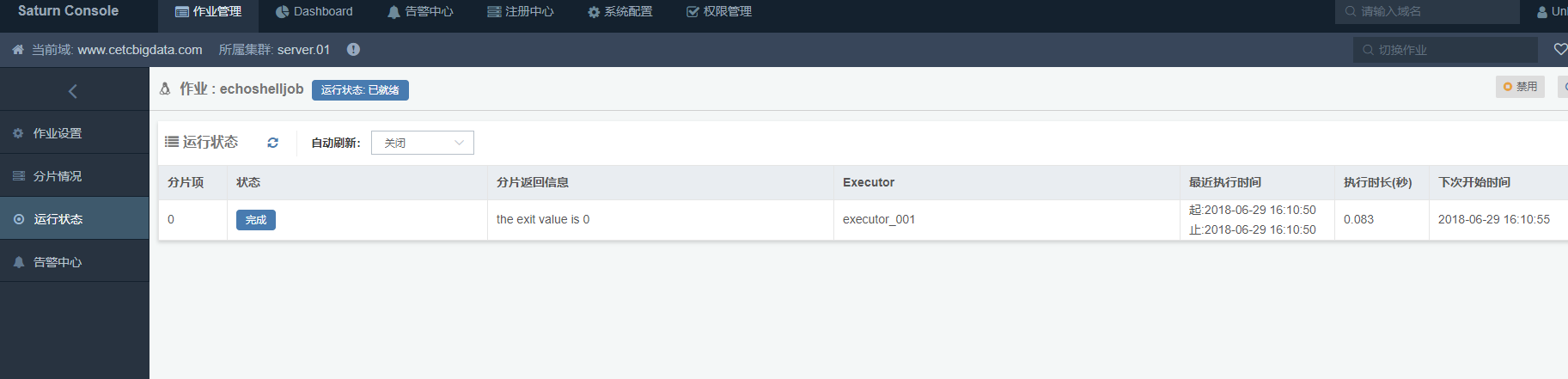
任务的执行总览可以通过报表的形式查看：



任务运行中对异常任务的监控也变得尤为重要，通过监控异常任务可以发现爬虫中的抓取异常，及时处理这些异常保持数据连贯性。



任务的运行状态可以通过相关界面查看



##### 配置中心

项目中配置文件比较繁杂，而且不同环境的不同配置修改相对频繁，每次发布都需要对应修改配置，如果配置出现错误，需要重新打包发布，时间成本较高，因此需要做统一的分布式配置中心，能做到自动更新配置文件信息，解决以上问题。

各个分布式配置中心比较：

| **配置中心** | **配置存储** | **时效性** | **数据模型** | **维护性** | **优点** | **缺点** |
| --- | --- | --- | --- | --- | --- | --- |
| disconf | zookeeper | 实时推送 | 支持传统的配置文件模式，亦支持KV结构数据 | 提供界面操作 | 基于分布式的Zookeeper来实时推送稳定性、实效性、易用性上均优于其他 | 源码较多，阅读和使用起来相对较复杂 |
| zookeeper | zookeeper | 实时推送 | 支持传统的配置文件模式，亦支持KV结构数 | 命令操作 | 实时推送稳定性、实效性 | 开发量大 |
| diamond | mysql | 每隔15s拉一次全量数据 | 只支持KV结构的数据 | 提供界面操 | 简单、可靠、易用 | 数据模型不支持文件，使用不方便 |
| Spring Cloud Config | git | 人工批量刷新 | 文件模式 | git操作 | 简单、可靠、易用 | 需要依赖GIT，并且更新GIT |

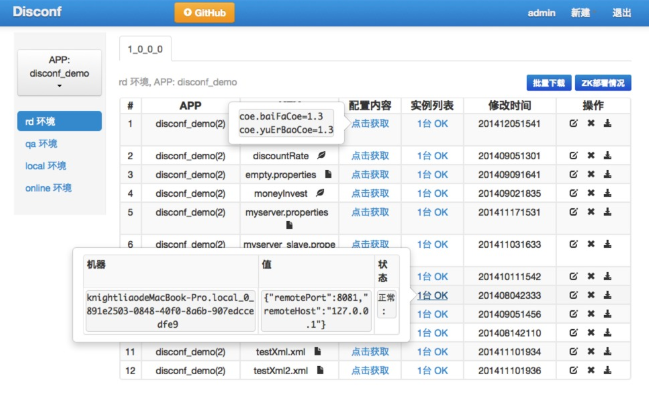
爬虫服务拟采用的分布式配置中心为disconf , disconf是百度开源出来的一款基于Zookeeper的分布式配置管理软件。目前很多公司都在使用，包括滴滴、百度、网易、顺丰等公司。通过简单的界面操作就可以动态修改配置属性。使用disconf省去应用很多配置，而且配置可以自动load，实时生效。disconf优点总结如下：

（1）部署简单：同一个上线包，无须改动配置，即可在多个环境中上线。

（2）部署动态化：更改配置，无需重新打包或重启，即可实时生效。

（3）统一管理：提供web平台，统一管理多个环境多个产品的所有配置。

disconf可以与springboot完美结合，项目中的配置文件管理可在disconf管理界面中进行配置。支持不同部署和开发环境的切换。



分布式爬虫配置中心架构：



分布式配置中心的服务是注册到服务注册中心，由服务注册中心负责监控该服务的运行状况。

##### 监控中心

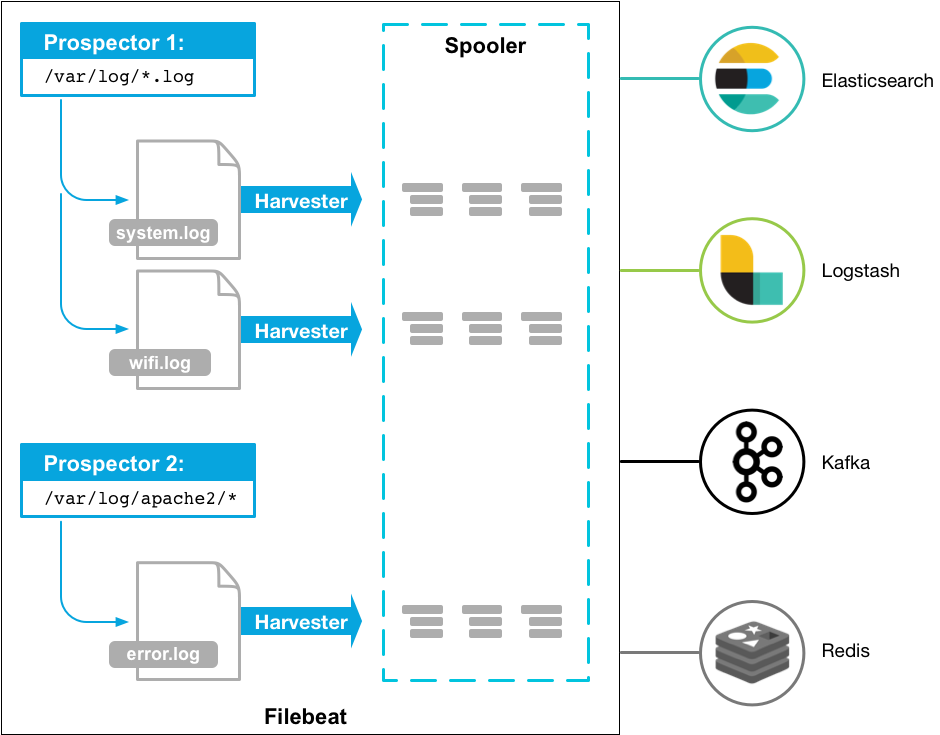
为了保证爬虫服务良好运行，并及时对异常情况作出响应。需要对爬虫服务的日志及其所在服务器系统资源使用情况进行实时监控。建议采用通过轻量级代理Filebeat增量采集爬虫服务日志文件和分布式服务（Kafka、Hbase、Elasticsearch等）日志文件，通过Metricbeat周期性采集监控服务器性能，传入消息队列Kafka相应主题，再由转换工具Logstash负载均衡从Kafka相应主题取出数据解析成Json格式存入Elasticsearch集群相应索引中，最后WEB可视化工具Kibana对Elasticsearch集群相应索引数据进行全文搜索及仪表展示。



###### 组件简介

###### Filebeat

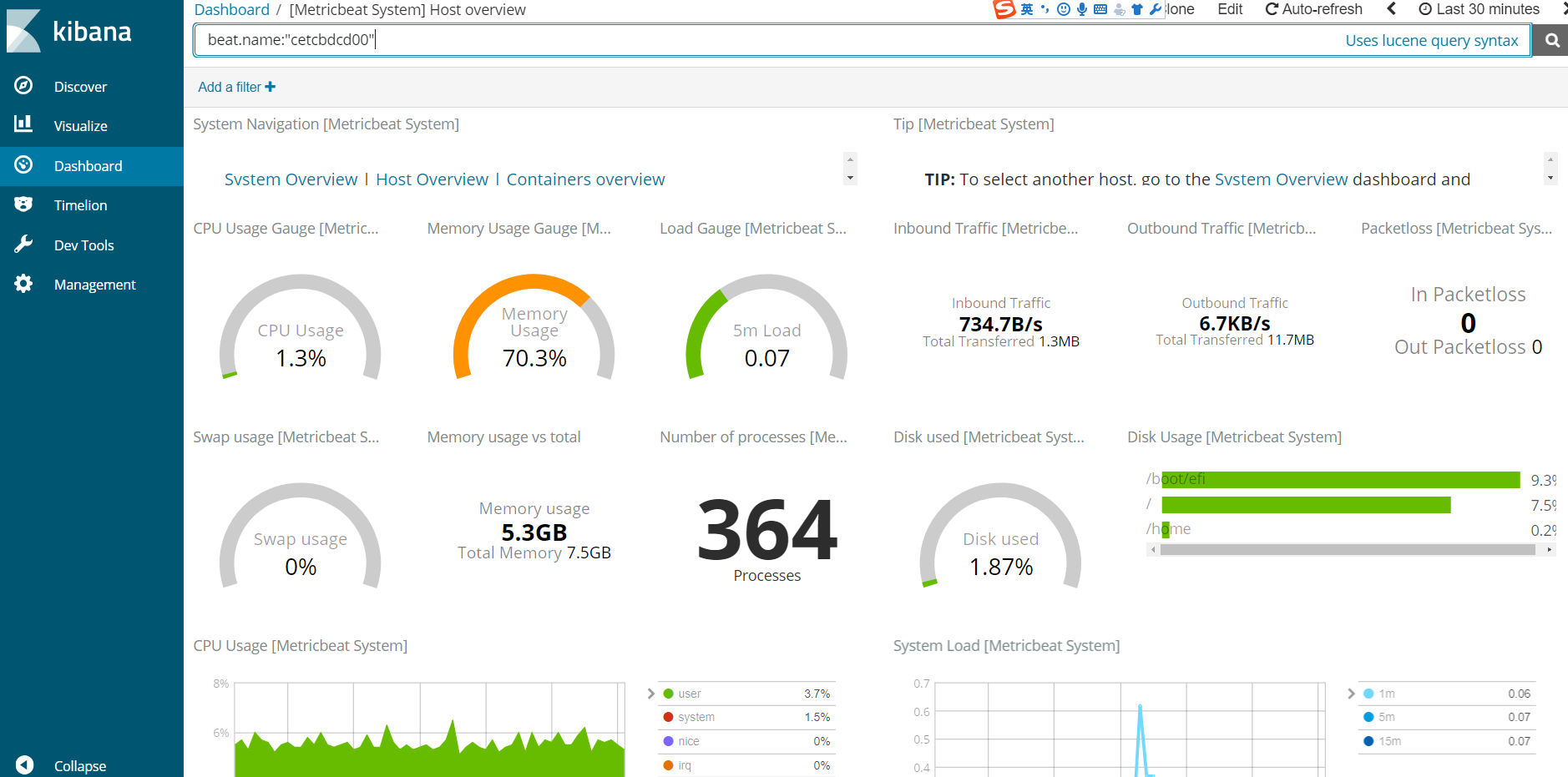
Filebeat是针对本地文本日志文件的轻量级日志采集器，可以监控日志文件文件或目录，默认从本文文件中一行一行的采集，也可以设定正则规则多行采集。记录日志采集偏移量，支持对日志增量采集。支持将采集数据传输到Elasticsearch、Logstash、Kafka或Redis中。

Metricbeat

Metricbeat可以获取系统级的 CPU 使用率、内存、文件系统、磁盘 IO 和网络 IO 统计数据等服务器的性能指标，可以监控运行在服务区之上的应用信息(Apache、MongoDB、MySQL、Ngnix、PostgreSQL、Redis、Zookeeper、 System等)。支持将采集数据传输到Elasticsearch、Logstash、Kafka或Redis中。

###### Kibana

Kibana是一个开源的Web分析和可视化平台，基于Elasticsearch存储数据的搜索及仪表盘展示，并与之交互，可通过图表(柱状图、线状图、饼图、环形图等)、表格和地图等类型仪表分析(可点击，可拖动，可挖掘)和展示数据。可对传入Elasticsearch的日志进行全文搜索及更直观的图表展示，有助于异常报警及故障运维。



###### 消息队列

由于单条日志报文数据及爬虫抓取单个html数据都在10kb以内，综合性能及可靠性，建议采用大数据平台的Kafka作为消息队列。Kafka是一个分布式流处理平台，可作为一个集群在一个或多个服务器上运行，以主题(topic)的方式对存储数据流进行分类，每一条记录由一个键(key)，一个值(value)和一个时间戳(timestamp)组成。具有以下3个关键能力：

* 发布与订阅：流数据像一个消息队列或消息系统（消息中间件）
* 存储：持久且容错保存数据流（数据可存储在分布式集群的多块磁盘中）
* 处理：实时处理流数据

两大类应用：

* 构建可靠地在系统或应用程序之间获取数据的实时的流数据通道
* 构建变换或响应数据流的实时流应用程序

四大核心API：

[Producer API](http://kafka.apache.org/documentation.html) ：允许应用将流记录发布到一个或多个Kafka 主题(topic)

[Consumer API](http://kafka.apache.org/documentation.html)：允许应用订阅一个或多个主题，并处理为其生成的记录流

[Streams API](http://kafka.apache.org/documentation/streams) ：允许应用充当流处理器(processor)，消费来自一个或多个主题的输入流并产生到一个或多个输出主题的输出流，有效地将输入流转换为输出流

[Connector API](http://kafka.apache.org/documentation.html)：允许构建和运行可重用的生产者(producer)或消费者(consumer)，将Kafka主题连接到现有应用程序或数据系统。 例如，关系数据库的连接器可捕获对表的每个更改。



##### 注册中心

常用服务注册中心解决方案包括了consul、zookeeper、eureka，特性对比如下：

| **Feature** | **Consul** | **zookeeper** | **euerka** |
| --- | --- | --- | --- |
| 服务健康检查 | 服务状态，内存，硬盘等 | (弱)长连接，keepalive | 可配支持 |
| 多数据中心 | 支持 | — | — |
| kv存储服务 | 支持 | 支持 | — |
| 一致性 | raft | paxos | — |
| cap | ca | cp | ap |
| 使用接口(多语言能力) | 支持http和dns | 客户端 | http（sidecar） |
| watch支持 | 全量/支持long polling | 支持 | 支持 long polling/大部分增量 |
| 自身监控 | metrics | — | metrics |
| 安全 | acl /https | acl | — |
| spring cloud集成 | 已支持 | 已支持 | 已支持 |

考虑采用consul作为服务注册中心。在spring cloud中使用简单，集成方便。

1. consul集群：

每个提供服务的节点上都要部署和运行consul的agent，所有运行consul agent节点的集合构成consul cluster。consul agent有两种运行模式：server和client。这里的server和client只是consul集群层面的区分，与搭建在cluster之上的应用服务无关。

以server模式运行的consul agent节点用于维护consul集群的状态。

官方建议每个Consul Cluster至少有3个或以上的运行在Server mode的Agent，Client节点不限。

2.服务注册

Consul支持两种服务注册的方式，一种是通过Consul的服务注册HTTP API，由服务自身在启动后调用API注册自己，另外一种则是通过在配置文件中定义服务的方式进行注册。Consul文档中建议使用后面一种方式来做服务 配置和服务注册。

具体使用方式：

在pom文件添加两个依赖：spring-boot-actuator和spring-cloud-starter-consul-discovery。在APP类里面使用@EnableDiscoveryClient注解，在application.properties 文件中配置相应注册信息，包括serviceName和instanceId

3.服务发现

具体使用方式：

在pom文件中添加spring-cloud-starter-consul-discovery依赖，在APP类里面使用@EnableDiscoveryClient注解，在application.properties 配置spring.cloud.consul.discovery.register=false，在业务类里面通过使用LoadBalancerClient和DiscoveryClient来找到服务并调用。

4.服务的维护：

提供了GUI的界面，同时也提供了consul的api。

##### 分布式爬虫应用

按照爬虫服务化的方案，爬虫应用应该是处于各个组件服务的包围中，围绕配置中心，任务管理，url资源池等组件来编写有关具体的爬虫应用。

爬虫抓取按照高内聚低耦合的设计原则，对于政务类网站，使用爬虫master应用将要抓取的url写入到url资源池，各个salver应用通过url资源池中获取的地址资源，有效抓取页面源代码，这一步并不进行对页面所有元素的解析，而是直接放入相应的存储引擎。带到需要使用该资源时，从存储引擎中检索该页面，并编写相应的爬虫解析规则对页面所需数据进行解析。

分布式爬虫应用框架技术选型：java采用webmagic框架编写针对简单页面的爬虫应用，python采用Scrapy框架，相关配置信息从配置中心和任务调度系统中获取，任务的启动和管理都在分布式任务调度系统中进行管理。

对于复杂页面采用模拟浏览器编写相对应的抓取规则。

Java分布式爬虫应用架构：



爬虫应用实现webmagic接口，从url资源池中拿到对应页面的访问地址，获取页面源代码，并将页面源代码通过封装必要的元数据信息使用json格式存入存储引擎中。

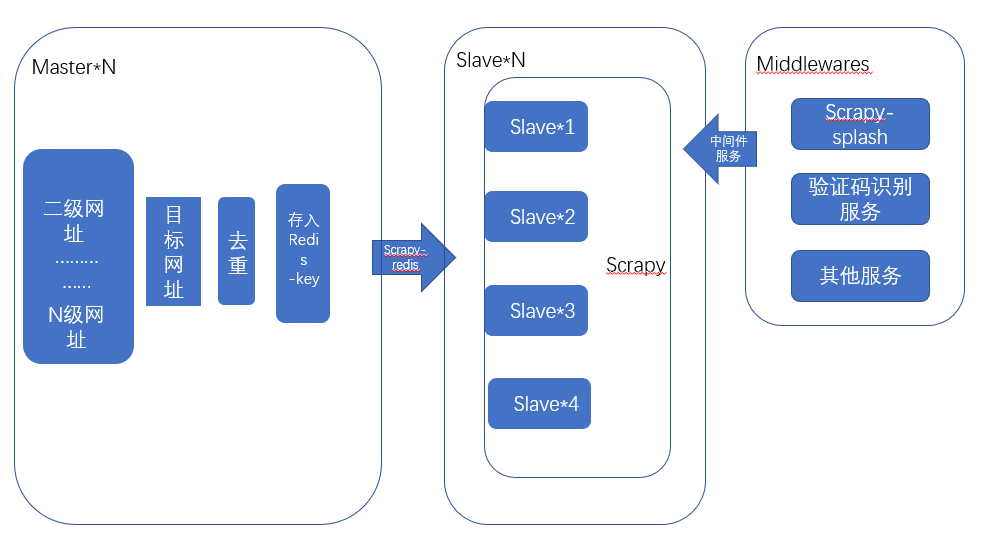
元素解析应用是在需要获取相应页面数据时，从存储引擎中检索到该页面元素，并解析相关数据再次封装必要的元数据信息通过json格式通过kafka消息发送给存储引擎。

url资源写入负责将目标页面的一级二级url全部解析出来并写入到url资源池，以便于爬虫应用从url资源池中获取对应的url并进行抓取。

应用所需线程数等配置信息可以统一通过读取配置中心的数据；爬虫应用集成saturn任务管理接口，可以实现定时任务管理和任务运行状态监控和调度。爬虫应用采用分布式部署，在服务注册中心注册，作为url资源的消费方，会记录url的抓取情况。分布式运行保证每个节点的任务不重复，处理异常的url会重新返回到url资源池中。

Python爬虫策略

一、总体框架



基础框架主要采用Scrapy，Redis。

1.Scrapy

Scrapy是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。

2.Redis

略

采用Scrapy-Redis实现分布式爬虫。

3.Scrapy-Redis

Master使用Python-Redis把内容页存入Redis-key。Slave启动Service监听Redis-key，监听规则：

Slave1 依次取出url（LBPOP）进行抓取，直到线程跑满；

Slave2 启动，同Slave1；

…

4.Scrapy-Splash

一些动态网页，内容是通过JS加载，或者通过AJAX异步获取数据，Scrapy无法处理此类网页，必须通过Scrapy-splash调用Splash处理。Splash是一个轻量级使用Docker容器的页面渲染服务，使用Lua语言执行JS脚本，处理AJAX等问题。Scrapy-splash支持多线程，性能上优于PhantomJS。

二、流程控制

Master：

1.获取目标网页（内容页）

Python爬虫包有很多，主要分为三类——基础静态请求型、模拟浏览器、框架。获取内容页的方法即使上述三种方式，其优劣方面如下表所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 速度 | 开发成本 | JS | AJAX | 灵活性 | 分布式 | 主要包 |
| 基础静态请求型 | 中 | 快 | 无法处理 | 无法处理 | 低 | 难度高 | Urllib2、beautifulsoup、lxml等 |
| 模拟浏览器 | 慢 | 中 | 慢 | 慢 | 高 | 不支持 | Selenium+phantomjs、  Chrome driver |
| 框架 | 快 | 慢 | 快 | 快 | 中 | 已有 | Scrapy |

开发人员可根据具体情况选取三种类型的爬虫。

2.去重

使用Redis的Heperloglog类型进行去重，详见上文。

3.存入Redis

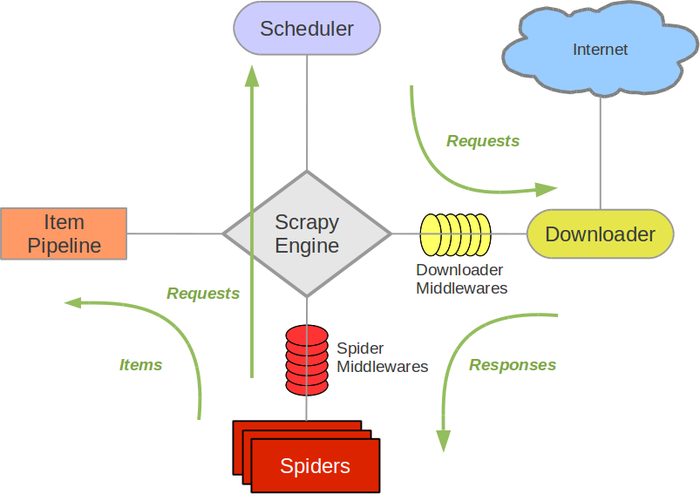
使用Python-redis调用存url如Redis，详见上文。

4.Scrapy-redis

Slave使用Scrapy-redis监听Redis中某个key的词，并取出进行抓取，详见上文。

5.Scrapy

Slave不同于Master，全部采用Scrapy框架编写。



Scrapy主要包括Scheduler、Downloader、Item、Pipeline、Spider、Middlewares等组建。

6.Item字段

Item定义了存储的字段。字段包括：

|  |  |
| --- | --- |
| title | <title>中的内容 |
| value | <body>中的内容 |
| lenth | Value的长度 |
| url | 抓取的目标页面 |
| parse\_time | 抓取时间 |
| region | 数据所属地区（人工定义方便查询） |
| industry | 所属行业（人工定义方便查询） |
| web\_name | 网站名称（人工定义方便查询） |
| Accessory\_path | 附件文件路径（无附件则为空） |

7.Scrapy-splash

调用splash进行JS渲染，以及处理AJAX等异步加载网页，详见上文。

8.验证码识别

采用卷积神经网络（CNN）进行验证码识别，步骤如下：

1）抓取训练集图片

2）人工标注训练集结果

3）TensorFlow建立CNN模型

4）训练

5）将验证码识别模型加载到Downloader中间件或scrapy-splash中

6）打开目标页面——下载验证码图片——预测结果——Post结果——抓取内容

9.中间件

Scrapy middlewares用于下载器、管道、调度器等位置的扩展，在后续开发中可能会用到。

##### 存储方案

爬虫爬取的数据将会作为公司永久的数据资源，为方便开发人员使用这部分数据，必须将爬虫爬取的数据作持久化存储，互联网数据具有以下几个特点：数据量大、数据结构复杂、数据种类繁多，需要结合数据的特点合理的选择存储方案，以方便后期利用这些数据做开发工作。目前公司大数据平台已经集成hdfs、hbase、hive、mysql、elasticsearch等多种数据存储及检索系统，选择合理的存储系统可满足对数据的处理需求。

经过调研，公司拟爬取的数据主要分为图片和文本类型，文本类型又分为结构化、半结构化、非结构化类型。经过讨论拟采用以下方案：

* Pdf、图片等文件类型：hdfs文件系统；
* 结构化文本数据：hive
* 半结构化文本数据：hbase
* 非结构化文本数据：elasticsearch



图片等文件数据直接存入hdfs文件系统，并做好hdfs路径规划，达到数据分类分级存储的目的，方便后期文件的查找。结构化数据可存储到hive中，hive是类似于关系型数据库的大数据仓库，提供了海量数据的存储和sql查询功能，后期可借助hive或spark的处理引擎，对数据做查询与统计分析。半结构化数据由于字段不统一适合存储到hbase中，hbase分布式列式数据库，提供海量数据的存储和查询，具有较高的读写性能，非常适合半结构化数据存储。非结构化数据例如资讯类文本数据，此类数据数据量较大，关键信息都在文本内部，为了方便后期查找，拟采用elasticsearch加入到搜索引擎。

为了降低存储层与各服务之间的耦合性，同时避免直接暴露存储集群服务器地址信息给外部服务，数据存储层将使用独立的接口服务方式对外提供数据读写服务。

###### 数据读写服务

根据不同数据类型、不同的数据结构提供统一的数据读写服务，以方便其他服务直接调用。

数据读写服务将通过kafka消息方式并根据不同的读写内容提供对应的实现接口。接口访问的URL的定义将按照：数据库类型/具体操作的规范命名，例如：向hive中写入一条数据的访问接口定义为：/hive/insert，向hive中一次写入多条数据的访问接口定义为：/hive/inserts。接口访问的请求参数将通过标准的XML格式对每个参数名进行解释定义，返回值也将以XML的格式返回给接口调用者。

###### 全文检索ES

针对从网页爬取的原始网页信息，由于涉及到的量较大，且内容本身是json格式的文本数据，在后期的时候过程需要对该类数据进行全文检索，故考虑将爬取的网页原始数据存储于ElasticSearch中。

ElasticSearch是一个基于Lucene的搜索服务器。它提供了一个分布式多用户能力的全文搜索引擎并且基于RESTful web接口，使用简单方便。ES的shards分片与replicas副本机制在保证读写性能的同事也提高了系统的容错性，recovery数据重分布使得集群在横向伸缩时有了很好的灵活性。

ES的数据存储将按照每天或者每月一个index建立索引，具体需根据索引的数据量而定。每个索引需至少创建两个数据副本，提高数据的容错性。设置每个索引分片的最大容量为30G左右，并根据一定时间段内的数据总量计算index的分片数量。

###### 分布式文件系统HDFS

Hadoop分布式文件系统(HDFS)被设计成适合运行在通用硬件(commodity hardware)上的分布式文件系统。它和现有的分布式文件系统有很多共同点。但同时，它和其他的分布式文件系统的区别也是很明显的。HDFS是一个高度容错性的系统，适合部署在廉价的机器上。HDFS能提供高吞吐量的数据访问，非常适合大规模数据集上的应用。HDFS放宽了一部分POSIX约束，来实现流式读取文件系统数据的目的。

针对爬虫获取到的文件数据，如PDF、word、图片等文件，可将该类文件直接以文件的格式存储到HDFS上，并对每个文件编写好描述信息，包含且不限于HDFS存储路径、文件标题、文件类型、文件来源等元信息。文件元信息将使用统一的数据存储结构，故元信息为结构化数据，该数据可直接存储于hive数据仓库中。在需要对文件进行检索时，可直接通过检索条件通过SQL在hive中获得需要文件的元信息，并根据信息描述到HDFS对应目录下获取源文件，达到快速查询到文件的目的。

###### 数据仓库HIVE

根据具体需求，对于爬取到的数据经过解析后的结构化数据，在hive中创建对应的数据库表并将数据写入其中。在创建hive数据库时，需根据数据种类、数据来源、数据用途等多维度创建数据库，将数据对应分类存储便于后续高效利用。在每个数据库中创建数据表时，需对每张数据表至少按照时间维度创建两级分区。

Hive数据仓库与hadoop以及spark的高度融合，使得存储于hive中的数据可高效的为后续的数据分析计算提供数据支撑。同时可为hive提供hiveserver和impala查询服务，hiveserver可满足稳定的大数据量SQL统计分析，impala可提供快速的查询相应，使得检索服务能够满足各种数据检索需求。

Hive是建立在 Hadoop 上的数据仓库基础构架。它提供了一系列的工具，可以用来进行数据提取转化加载（ETL），这是一种可以存储、查询和分析存储在 Hadoop 中的大规模数据的机制。Hive 定义了简单的类 SQL 查询语言，称为 HQL，它允许熟悉 SQL 的用户查询数据。同时，这个语言也允许熟悉 MapReduce 开发者的开发自定义的 mapper 和 reducer 来处理内建的 mapper 和 reducer 无法完成的复杂的分析工作。

Impala是Cloudera公司主导开发的新型查询系统，它提供SQL语义，能查询存储在Hadoop的HDFS和HBase中的PB级大数据。已有的Hive系统虽然也提供了SQL语义，但由于Hive底层执行使用的是MapReduce引擎，仍然是一个批处理过程，难以满足查询的交互性。相比之下，Impala的最大特点就是它的快速。可以使用Impala对hive中的数据进行操作达到快速分析的目的。

###### 分布式列数据库HBASE

HBase – Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。HBase不同于一般的关系数据库，它是一个适合于非结构化数据存储的数据库。另一个不同的是HBase基于列的而不是基于行的模式。

对于爬取解析后的半结构化或者非结构化数据，可存储于HBASE中，每类数据也可根据时间段创建数据表。创建数据表时必须为每张表创建预分区，并根据预分区数量设计rowkey的散列值。在设计rowkey时，需遵循三个基本的原则，第一：长度原则，rowkey是一个二进制码流，可以是任意字符串，最大长度 64kb，建议越短越好，不要超过16个字节；第二：散列原则，如果rowkey按照时间戳的方式递增，不要将时间放在二进制码的前面，建议将rowkey的高位作为散列字段，由程序随机生成，低位放时间字段，这样将提高数据均衡分布在每个RegionServer，以实现负载均衡的几率。如果没有散列字段，首字段直接是时间信息，所有的数据都会集中在一个RegionServer上，这样在数据检索的时候负载会集中在个别的RegionServer上，造成热点问题，会降低查询效率；第三：唯一原则，必须在设计上保证其唯一性，rowkey是按照字典顺序排序存储的，因此，设计rowkey的时候，要充分利用这个排序的特点，将经常读取的数据存储到一块，将最近可能会被访问的数据放到一块。后期需要根据业务情况和数据量合理设计rowkey的值。

#### 总结

本篇介绍了从网络硬件设施，到分布式爬虫服务组件的设计概要，从数据采集到存储检索做了流程设计。任务管理，配置中心等组件负责为分布式爬虫应用提供必要的组件和服务，围绕在爬虫应用周边的组件也可复用到其他相关分布式系统，形成微服务，对后期项目支持提供有效服务。