Elise架构设计

# 1.架构设计



**任务配置中心：**主要负责配置建立站点，建立页面模型，创建采集任务

**Mysql：**存储配置任务

**Redis：**缓存站点，页面模型，任务等配置信息，同时配置采集器新抽取下来URL，并做去重的操作。

**爬虫采集器：**根据URL提取数据核心程序。

**测试采集器：**提供给界面用的测试采集接口。

# 2.配置中心

## 2.1.业务逻辑



### 2.1.1.站点管理

界面上提供基本的站点管理功能，大概需要提供的接口如下：

#### 2.1.1.1.查询接口

需要根据站点名称，进行模糊查询

#### 2.1.1.2.添加接口

填写站点名称和站点域名,进行提交

填写的内容需要进行验证

1. 非空验证
2. 站点重复验证
3. 域名正则表达式进行验证。

#### 2.1.1.3.更新接口

基于上述已经获取到站点信息进行更改。

填写的内容需要进行验证

1. 非空验证
2. 站点重复验证
3. 域名正则表达式进行验证。

#### 2.1.1.4.删除接口

根据站点的ID进行删除某一个站点

### 2.1.2.页面模型

界面上提供基本的页面模型管理功能，需要提供的接口如下：

#### 2.1.2.1.查询接口

可以根据站点和页面模型名称组合进行查询。

#### 2.1.2.2.添加接口

可以添加页面模型基本信息，1.页面模型基本属性，2.浏览器采集设置，3、翻页设置，4，登录设置，5、页面属性等等

针对必要的界面上填写需要进行验证

填写的内容需要进行验证

1. 非空验证
2. 页面模型重复验证
3. 其他进行验证。

#### 2.1.2.3.签出接口

必须只能一个人进行签出

如果已经有人签出了，不能再被其他人签出。

#### 2.1.2.4.签入接口（更新）

页面模型签入，就是执行更新的操作

**特别提醒**，如果更新的人，不是签入的人，不允许签入。

可以更新页面模型基本信息，1.页面模型基本属性，2.浏览器采集设置，3、翻页设置，4，登录设置，5、页面属性等等

针对必要的界面上填写需要进行验证

填写的内容需要进行验证

1. 非空验证
2. 页面模型重复验证
3. 其他数据验证

#### 2.1.2.5.删除接口

根据页面模型ID进行删除数据

### 2.1.3.任务管理

界面上提供基本的任务管理功能，需要提供的接口如下：

#### 2.1.3.1.查询接口

可以根据站点和页面模型名称组合进行查询。

#### 2.1.3.2.添加接口

填写任务运行的基本信息，针对页面模型进行匹配

#### 2.1.3.3.签出接口

必须只能一个人进行签出

如果已经有人签出了，不能再被其他人签出。

#### 2.1.3.4.签入接口（更新）

填写任务运行的基本信息，针对页面模型进行匹配

#### 2.1.3.5.删除接口

可以根据任务ID进行删除某一个任务。

## 2.2.技术选型

### 2.2.1. Springboot框架介绍



SpringBoot是伴随着Spring4.0诞生的；

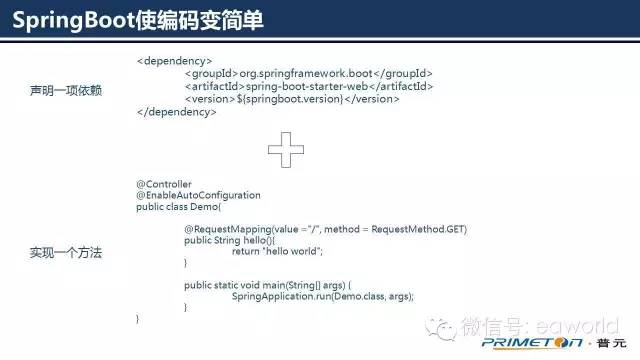
从字面理解，Boot是引导的意思，因此SpringBoot帮助开发者快速搭建Spring框架；

SpringBoot帮助开发者快速启动一个Web容器；

SpringBoot继承了原有Spring框架的优秀基因；

SpringBoot简化了使用Spring的过程。

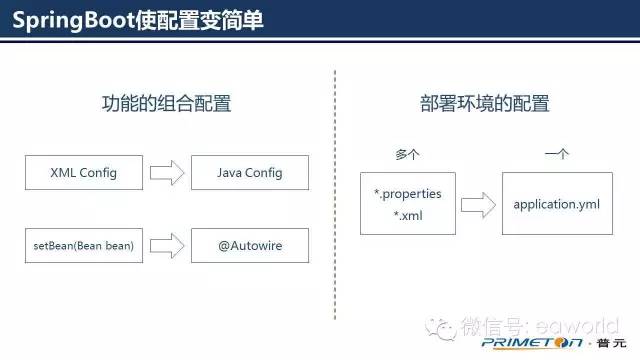
### 2.2.2.Spring Boot使编码变简单



Spring由于其繁琐的配置，一度被人认为**“配置地狱”**，各种XML、Annotation配置，让人眼花缭乱，而且如果出错了也很难找出原因。

Spring Boot更多的是采用Java Config的方式，对Spring进行配置。

### 2.2.3.Spring Boot使配置变简单



### 2.2.4.Spring Boot使部署变简单



## 2.3.前端逻辑层次架构



1. **web页面：**采用vue框架编写的界面。目前采用的是前后端分离的开发模式
2. **拦截器：**1.拦截一切非法请求。2.请求验证
3. **Controller层：**业务接口层

**数据请求预处理**

**数据合法性验证：**

**业务请求验证：**比如，删除某一个站点，需要判断这个站点是否被页面中或者其它任务调用。

**数据封装：**根据请求的参数，封装成实体对象

**数据返回：**返回JSON结果

1. **Service层：**业务逻辑层
2. 需要判断是直接调用数据库还是读取redis
3. 读取数据库同时需要写入redis
4. **DAO层：**数据库访问层

通过mybatis框架，通过配置文件实现ORM轻量级数据库访问调用。

## 2.4.事务处理

事务处理的目的：就是要保证业务逻辑的原子性。

事务有2个 方面的事务，1.数据库的事务，2.Redis事务

数据库处理的位置：写入数据库，同时写入Redis的时候需要执行



## 2.5.Restfull请求接口规范约束

业务请求接口需要符合restfull风格

Get /site/get 请求所有数据

Get /site/get/12 请求单个数据

POST /site/post 添加数据

PUT /site/put/12 修改数据

DELETE /site/delete/12 删除数据

## 2.6.数据同步版本更新机制

### 2.6.1.问题描述



前端操作某个数据的时候，把数据更新到redis之后，他不会直接去更新采集器的内存。而采集器这边也不是频繁的去访问redis，为此页面上更新某个数据之后，如何马上更新采集器的配置，需要设计一套新的方案.

### 2.6.2.RPC同步方案



由采集器提供PRC服务，前端调用RPC的方式进行数据同步，

**优点**：采集器的内存和配置文件会马上更新，

**缺点**：

1.建立RPC连接很慢，同步很慢，

2.前端需要知道采集器地址和接口，否则无法同步。

### 2.6.3.redis发布订阅方案



由采集器从redis订阅数据

前端往redis发布数据

优点：前端有数据更新，采集端马上获取到数据。

缺点：采集器需要与redis维持一个长连接，一旦这个长连接发送了中断，某个消息肯定获取不到，即便是重启，重新建立了连接。也无法获取到已经发生的订阅请求。

### 2.6.4.数据版本更新方案

前端更新数据之后，同时去更新配置表，将配置表中的版本进行更新。

采集器，有一个独立线程，去扫描redis版本配置表，判断版本配置表中的版本信息大于采集器本地版本，则需要进行数据更新，否则不变。

优点：不会丢失订阅消息。

缺点：采集器需要独立开启线程去扫描redis集合，前端修改之后，采集器不会立马更新，如果采集器扫描周期（比如1分钟扫描一次）1分钟之后才会更新过来。

### 2.6.5.三个方案比较

|  |  |  |  |
| --- | --- | --- | --- |
|  | RPC | Redis发布/订阅 | 版本配置集合方案 |
| 前端响应效率 | 慢 | 快 | 快 |
| 采集器更新及时性 | 快 | 快 | 不会立刻生效 |
| 出现网络异常 | 无法同步 | 丢失消息 | 不会丢失消息 |
| 前端配置难度 | 需要知道采集器IP和端口 | 不需要配置 | 不需要配置 |
| 问题出现频度 | 经常 | 可能经常 | 少 |

综上所述，采集版本配置集合的方案应该是合理的

# 3.爬虫采集器和测试采集器

## 3.1.逻辑结构



爬虫采集器包含了4个主要组件（jar）

测试采集器包含了2个主要组件（jar）

这里涉及到2个程序和1个组件(jar)，并且可以下载的时候，可以扩展很多登录服务

2个程序

**爬虫采集器**:核心爬虫采集程序

**测试采集器**:提供前端测试接口程序

6个组件，4个核心组件，2个辅助组件

**调度组件：**负责任务调度，主要获取Redis任务并生成任务。

**下载组件：**主要负责数据下载，下载过程中，需要判断是否采集浏览器下载，是否需要登录下载等，需要是否代理等。

**登录服务：**根据采集的需要可以调用自己的登录服务，也可以调用第三方提供验证码登录服务。

**抽取组件：**专门提供数据抽取，抽取方式可以多种XPATH，正则表达式，jsonXpath等。可以横向扩展采集不同的工具或者组件进行数据的抽取。

**htmParser :** 封装dom4j抽取组件

**持久化组件：**负责数据的持久化，可以保存文件，可以保存hbase,可以推送到kafka，方面横向扩展，可以将数据持久化不同的方式。

## 3.2.调度组件

Scheduler主要管理待爬取的任务、新任务去重后存入Redis待抓取任务队列。



### 3.2.1.Redis任务防丢失备份机制

任务备份工作流程



采用双队列的备份机制

1. **采集任务队列：**采集list数据结构
2. **备份队列：**采用set数据结构

主要工作流程

1. 从采集任务队列，取任务并删除
2. 往备份队列中，备份任务
3. Scheduler任务执行完成，采把备份任务从备份队列中移除掉。

### 3.2.2.备份队列任务扫描和监控



1. 扫描备份队列，根据入队列的时间与当前时间的比较，如果长时间未被处理（时间长度可设定）
2. 获取长时间未删除的任务
3. 将未完成的任务添加到采集任务的队列中

## 3.3.下载组件

Downloader主要负责将种子任务下载下来然后将下载下来的页面交给Extractor进行抽取，并输出日志。



## 3.4.登录服务



## 3.5.抽取组件

### 3.5.1.正常抽取流程

Extractor主要根据下载下来的Page（网页内容），进行解析(html、json、jsonp)，得到新的url任务和页面属性抽取结果：



### 3.5.2.抽取结果，数据结构设计

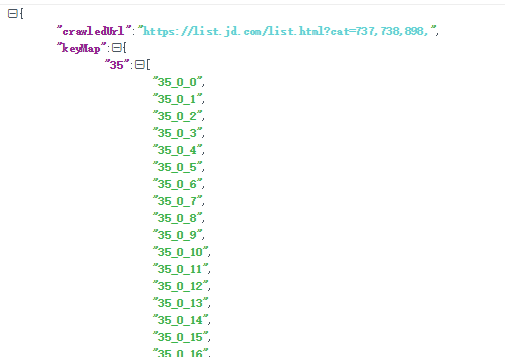
页面模型设计的时候虽然是树形的结构，

但是我们的结果JSON数据，我们没有采用树形结构进行存储。为了以后的数据的解析，我们设计出更加便捷，逻辑并不太复杂的JSON结构。

样本数据如下







设计该结构的特点：

1. 没有更页面模型一样，呈现树形结构。而是采用行式结构。在valueMap每条记录都是一行一行来进行存储。
2. 正因为是行式存储，我们的数据解析更加方便。
3. 我们根据keyMap来确定归属到不同的表。
4. 我们不存储页面属性具体的信息。为了减少冗余，显示的时候可以去查页面模型。

### 3.5.3.数据抽取，递归调用

**void** extractRecursion(String url, Object object, Map<Integer, PageModelProperty> propertyMap, **int** parentId, String rowkey, List<PageModelAttribute> pageModelAttributeList) {

*//1.抽取属性循环*  
 **for** (Map.Entry<Integer, PageModelProperty> entry : propertyMap.entrySet()) {  
 PageModelProperty pageModelProperty = entry.getValue();  
*//排除非当前层级的数据*  
 **if** (pageModelProperty.getParentId() != parentId) {  
 **continue**;  
 }  
*//排除属性组*  
 **if** (pageModelProperty.getGroupType() == Const.***PAGEMODLEPROPERTY\_TYPE\_GROUP***) {  
 **continue**;  
 }  
*//抽取属性* String propertyValue = *pageTypeExtractor*.extractProperty(object, pageModelProperty.getParseType(),  
 pageModelProperty.getMatchExpression(),  
 pageModelProperty.getUrlPropertyExtract(),  
 url);  
 pageModelAttributeList.add(*setpageModelAttribute*(pageModelProperty, propertyValue, rowkey));  
 }  
 *//2.抽取属性组* **for** (Map.Entry<Integer, PageModelProperty> entry : propertyMap.entrySet()) {  
 PageModelProperty pageModelProperty = entry.getValue();  
//*排除非当前层级的数据*  
 **if** (pageModelProperty.getParentId() != parentId) {  
 **continue**;  
 }  
*//排除非属性组*  
 **if** (pageModelProperty.getGroupType() != Const.***PAGEMODLEPROPERTY\_TYPE\_GROUP***) {  
 **continue**;  
 }  
*//抽取属性组的数据，获取到的信息，是一个数组* List<Object> str1 = *pageTypeExtractor*.extractPropertyList(object,  
 pageModelProperty.getParseType(),  
 pageModelProperty.getMatchExpression(),  
 pageModelProperty.getUrlPropertyExtract(),  
 url);  
 **int** nIndex = 0;  
 **for** (Object objNode : str1) {  
*//递归抽取下层内容* extractRecursion(url, objNode, propertyMap, pageModelProperty.getPropertyId(), rowkey + "\_" + nIndex, pageModelAttributeList);  
 nIndex++;  
 }  
 }  
}

## 3.6.封装dom4j抽取组件

dom4j是一个简单的开源库，用于处理XML、 XPath和XSLT，它基于Java平台，使用Java的集合框架，全面集成了DOM，SAX和JAXP。

使用htmlcelaner清洗网页成dom4j的dom树，方便xpath进行抽取。

## 3.7.持久化组件

对抓取数据进行持久化存储。



## 3.8.elise-app程序入口

Elise程序入口，包含全局配置信息(缓存页面规则)，并调用四大组件，完成采集流程。

其中，规则更新流程：



## 3.9.elise-testapp程序入口

采用springboot接口支持，传递采集页面和页面模型和页面模型属性，完成采集，并返回结果：



1. web测试发送测试请求
2. 测试采集器接收到请求，根据URL，调用下载组件
3. 采集完成之后，返回Page对象
4. 根据Page对象以及根据页面提供页面属性，开始抽取相关的字段
5. 返回抽取结果
6. 最后由ELISE返回JSON数据

# 4.日志系统

## 4.1.日志收集系统技术选型

### 4.1.1.ELK介绍

[ELK](http://467754239.blog.51cto.com/4878013/1700828)由Elasticsearch、Logstash和Kibana三部分组件组成；

Elasticsearch是个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。

Logstash是一个完全开源的工具，它可以对你的日志进行收集、分析，并将其存储供以后使用

kibana 是一个开源和免费的工具，它可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助您汇总、分析和搜索重要数据日志。

### 4.1.2.四大组件

Logstash: logstash server端用来搜集日志；

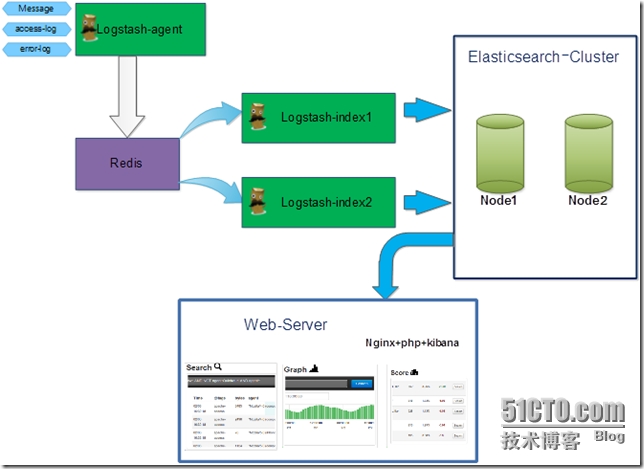
Elasticsearch: 存储各类日志；

Kibana: web化接口用作查寻和可视化日志；

Logstash Forwarder: logstash client端用来通过lumberjack 网络协议发送日志到logstash server；

### 4.1.3.ELK工作流程

在需要收集日志的所有服务上部署logstash，作为logstash agent（logstash shipper）用于监控并过滤收集日志，将过滤后的内容发送到Redis，然后logstash indexer将日志收集在一起交给全文搜索服务ElasticSearch，可以用ElasticSearch进行自定义搜索通过Kibana 来结合自定义搜索进行页面展示。

[](http://s3.51cto.com/wyfs02/M01/74/25/wKioL1YWGBnxpAwyAAGFWRpT6X8070.jpg)

### 4.1.4.ELK的帮助手册

ELK官网：https://www.elastic.co/

ELK官网文档：https://www.elastic.co/guide/index.html

ELK中文手册：http://kibana.logstash.es/content/elasticsearch/monitor/logging.html

## 4.2.日志的关键指标

### 4.2.1.程序中断异常

根据中断之后，收集程序中断的原因，输出其日志信息。

1. 断电中断，采集服务器需要自动重启功能。
2. 运行异常中断，需要有第三方程序守护采集程序的运行。

### 4.2.2.页面请求异常

1）URL链接无效或超时

2）登录失败

3）登录验证失败

### 4.2.3.抽取异常

1）无法抽取数据

2）表达式有问题。

## 4.3.日志的本地化存储方案

