天津市大学软件学院

2016级卓越实验班毕业设计论文



姓 名: 魏 凯

学 号： 20135074

合作培养企业: 中 科 瑞 通

专 业 方 向: 大 数 据

全文搜索引擎的设计与实现

摘要

自互联网诞生以来，人们就越来越多的渴望获取到更多的信息，随着信息技术的发展，如何从互联网海量的信息中选取自己所需要的信息就如大海捞针一直困扰着互联网的建设者。而搜索引擎的提出和实现则将互联网发展推到了一个新高度，搜索引擎可以通过使用者的简单操作从庞大的互联网信息网中筛选出符合使用者需求的信息，从而大大降低了信息获取的时间成本，同时提高了信息获取的效能，而其作为用户进入互联网的入口同时还提供了信息参考指导的作用，而全文搜索引擎作为其中一种类型，深受大量使用者的青睐。

本文针对一个功能精简的全文搜索引擎的实现展开说明，分别通过背景的介绍、可行性分析、领域及知识层面的阐述对该系统实现的背景和基础做一表述，然后通过需求分析、系统设计对该搜索引擎进行设计和实现的记录，最后通过测试及总结对整个系统的运行和后续问题进行总结性的讨论。

整个系统的搭建实现，使用了负载均衡的ES搜索服务，同时数据的存储使用到分布式NoSQL数据库HBase，本文通过详细介绍和说明几种互联网数据处理技术，对搜索引擎的实现过程加深理解。

关键词：搜索引擎 负载均衡 分布式 数据处理

The Design And Implementation Of Full Text Search Engine

**ABSTRACT**

Since the birth of the Internet, people are more and more eager to get more information. With the development of information technology, how to select the information that they need from the vast amounts of information in the Internet like looking for a needle in the ocean that has plagued the Internet builders. The search engine is put forward and realized the development of the Internet be pushed to a new height, the search engine can be selected to meet the needs of users of information from the Internet information network in the large through a simple operation, thereby greatly reducing the time cost of obtaining information, and improve the efficiency of access to information, but as the entrance the user to enter the Internet also provides reference information and guidance, and full text search engine as one type of search engines have a large number of users.

To achieve full-text search engine according to the function of a streamlined expansion, through the introduction of the background, feasibility analysis, and describes the field of knowledge to do a description of the background and the foundation of implementing the system, and then through the requirement analysis, system design and implementation of the record of the search engine, discussion summary finally through the test and summary of the operation of the whole system and subsequent problems.

An implementation of the whole system, the use of load balancing ES search service, and data storage using distributed NoSQL database HBase, this paper introduces and explains several Internet data processing technologies, to understanding and implementation process of search engine.

**Key Words**: Search Engine Load Balance Distributed Data Processing

# 目录

# 第一章 绪论

## 1.1 编写目的

为了规范化本系统的编写，提高系统开发过程中的能见度，保证系统开发的质量、需求的完整与可追溯性，方便系统开发过程中的控制与管理，同时提出全文搜索引擎系统的开发过程而编写该文档。通过此文档，以保证业务需求的提出者与需求分析人员、开发人员、测试人员及其也相关人员对需求达成共识，并作为开发成果验收的原始依据。

## 1.2 术语定义

本文档使用到的相关专业术语的定义和说明如下表1.1所示：

表1.1 术语定义以说明

Table 1.1 Definition of terms

|  |  |  |
| --- | --- | --- |
| 英文/简称 | 中文名称/全称 | 说明 |
| PIM | 平台无关的模型 | 软件模型或业务系统，它独立于实现它的特定技术平台 |
| BCE模式 | Boundary-Control-Entity Patterns | BCE模式在概念上跟著名的MVC模式（Model-View-Control Pattern）相似。简单来说，在BCE模式中，将对象分为三类：边界类（boundary class）、控制类（control class）和实体类（entity class） |

## 1.3 本文结构

文档按章节顺序书写，开头摘要论述了该系统开发的提出及相关概要，本章绪论则主要说明了文档书写的目的及相关术语说明。从第二章开始步入正题，第二章介绍并分析了该系统开发的可行性，包括现行系统的不足及本系统所带来的优势等；第三章对系统开发所使用到的技术环境等进行简要的介绍；第四章对该系统所对应的组织机构进行了系统的领域分析，从而得出相关业务信息；第五章进行了系统的需求分析，确定需求结构；第六章则展开系统的详细设计，包括对象模型、业务逻辑等的具体分析设定以及用到的数据库的设计；然后，在第七章，开始对系统进行编码和一些需求的测试，确保系统符合最初设计分析的用户需求；接下来就使用到的相关文献进行罗列和对系统实现所得到的帮助表示感谢。

# 第二章 系统可行性分析

## 2.1 背景

自互联网提出以来，消息的传递就随着技术的发展而不断变革，由最初的人与人间的口头传递到网络传播，信息量和传播速度都步入了一个崭新的时代。步入21世纪互联网时代，通过覆盖全球的互联网，信息几乎能在发布瞬间就传播到全球各个地方，而且几乎没有任何成本，但如何从这庞大的信息网中取得自己需要的信息就显得越来越困难，起初信息量少的时候，人们通过人工筛选获取符合自己要求的信息，但互联网带动信息爆炸性发展，在信息的海洋里想要找到自己感兴趣的部分堪比徒步登天。因此，如何高效的在海量数据中筛选出符合要求的数据就显得尤为重要，而搜索引擎正是为此诞生的。如今，搜索引擎的使用已经深入人心，每一个互联网使用者都将其作为互联网的入口和指导，同时搜索引擎也大大降低了使用互联网的门槛。随着大数据时代的来临，数据以高于指数级的增长涌现出来，而此时的搜索引擎就显得更加重要了，它可以从大量的数据中获取到有用的部分，对企业乃至国家都是战略分析工具，也是大数据时代将搜索引擎推上了另一个发展的高潮。

## 2.2 现行系统概述

### 2.2.1 组织目标和战略

**1.目标**

整合全球信息，供大众使用，人人受益。

**2.战略**

（1）专注于开发社交网络应用，互联网和数字化世界“在爆发性增长”，社交媒体只是其中一部分；

（2）实现万能搜索模式，不局限于文本；

（3）组织全世界的信息，使之可在全世界内广泛存取和使用。

### 2.2.2 业务概况

现行搜索系统的业务概况如下表2.1所示：

表 2.1 现行组织系统业务概况

Table 2.1 Current organizational system business overview

|  |  |
| --- | --- |
| 组织部门 | 业务 |
| 搜索服务 | 信息检索 |
| 数据获取 | 信息爬取，数据清洗，数据分析处理 |
| 运营部 | 竞价排名，广告推广 |
| 用户 | 信息检索 |

其中，搜索服务及数据获取作为该系统的功能被抽象为组织模块，主要围绕数据的获取和处理展开业务，而运营部门则主要进行市场运营方面的业务，例如竞价排名、广告推广等，而用户作为该系统的重要涉众之一抽象为一个组织，参与信息检索业务，且作为业务发起方。

## 2.3 拟建立的系统

### 2.3.1 摘要说明

本系统主要作为一项服务提供接口，然后通过网页客户端调用进行相关业务流程，较传统的网页应用系统更加注重服务的质量和性能。服务主要分为三个模块，分别是爬虫模块、数据处理模块以及搜索服务模块，数据存储使用分布式的HBase数据集群以提升性能和可用性，搜索服务则由ElasticSearch提供，服务端实现以下基本要求：

1. 便捷的Web接口，对外业务完全透明；
2. 接口提供用户对信息的全文检索及数据快照功能；
3. 使用分布式架构，集群化管理，尽可能提升可用性和性能；

### 2.3.2 初步建设计划

本系统计划于2017年3月5日开工建设，开发期2个月。

## 2.4 经济可行性

### 2.4.1 系统开发费用

1.人员费用

本系统开发期2个月，开发人员1人，但由于是毕业设计，并不存在人员费用。

2.硬件设备费用

本系统开发和使用均为毕业设计需求，硬件设备为开发人员自备，并不存在相关费用。

3.软件费用

系统所需软件费用为0元，其中包括：

1. Apache HBase（免费）
2. IntelliJ IDEA 17 Community（免费）
3. VMware Workstation 12（30天试用）
4. Apache Zookeeper（免费）
5. Apache ElastSearch（免费）
6. Berkeley Database（免费）
7. Apache Spark（免费）

4.耗材费用

由于文档等需求，各种耗材费用预计约50元。

5.其他费用

系统开发中不存在差旅调研等，但需要文档查重，故费用预估500元。

6.不可预见费用

由于开发过程中遇到的事件不可知，拟不可预见费用约200元。

7.总开发费用

综上分析，该系统总的开发费用为750元。

### 2.4.2 系统运行费用

由于是毕业设计，该系统并不需要运行维护，故该项费用为0元。

### 2.4.3 收益

通过本次系统开发，提升个人实际开发水平，积累系统开发经验，发现开发过程中出现的问题和解决办法，同时作为毕业设计，其间接经济收益预计30万元。

## 2.5 技术可行性

该系统服务端实现语言为Java，数据存储分别采用BDB及HBase均为自学，且已经有叫好的掌握，而其他数据处理技术均为大数据应用课程学习技术，加之开发者专业知识，具有技术可行性。

## 2.6 社会可行性

目前各大互联网公司已经开发了此类系统，且该系统与国家政策或法律不存在任何冲突。系统的开发有利于信息化时代的快速建设，方便人们获取信息，且操作简便，适用性广，具有可行性。

## 2.7 可行性分析结论

通过对系统的经济、技术、社会等可行性分析，可以确定该系统具有开发的完全必要性和可行性，予以立项开发。

# 第三章 系统开发环境及相关技术简介

## 3.1 系统开发环境简介

### 3.1.1 JVM简介

JVM是Java Virtual Machine（Java虚拟机）的缩写，JVM是一种用于计算设备的规范，它是一个虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。Java虚拟机包括一套字节码指令集、一组寄存器、一个栈、一个垃圾回收堆和一个存储方法域。 JVM屏蔽了与具体操作系统平台相关的信息，使Java程序只需生成在Java虚拟机上运行的目标代码（字节码）,就可以在多种平台上不加修改地运行。JVM在执行字节码时，实际上最终还是把字节码解释成具体平台上的机器指令执行。

Java语言的一个非常重要的特点就是与平台的无关性。而使用Java虚拟机是实现这一特点的关键。一般的高级语言如果要在不同的平台上运行，至少需要编译成不同的目标代码。而引入Java语言虚拟机后，Java语言在不同平台上运行时不需要重新编译。Java语言使用Java虚拟机屏蔽了与具体平台相关的信息，使得Java语言编译程序只需生成在Java虚拟机上运行的目标代码（字节码），就可以在多种平台上不加修改地运行。Java虚拟机在执行字节码时，把字节码解释成具体平台上的机器指令执行。这就是Java的能够“一次编译，到处运行”的原因。

### 3.1.2 Nginx简介

Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，并在一个BSD-like 协议下发行。由俄罗斯的程序设计师Igor Sysoev所开发，供俄国大型的入口网站及搜索引擎Rambler（俄文：Рамблер）使用。其特点是占有内存少，并发能力强，事实上nginx的并发能力确实在同类型的网页服务器中表现较好，中国大陆使用nginx网站用户有：百度、京东、新浪、网易、腾讯、淘宝等。

Nginx可以在大多数 Unix/Linux OS 上编译运行，并有 Windows 移植版。Nginx 是一个很强大的高性能Web和反向代理服务器，它具有很多非常优越的特性：在连接高并发的情况下，Nginx是Apache服务器不错的替代品：Nginx在美国是做虚拟主机生意的老板们经常选择的软件平台之一。能够支持高达 50,000 个并发连接数的响应。

### 3.1.3 Linux简介

Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

Linux操作系统诞生于1991 年10 月5 日（这是第一次正式向外公布时间）。Linux存在着许多不同的Linux版本，但它们都使用了Linux内核。Linux可安装在各种计算机硬件设备中，比如手机、平板电脑、路由器、视频游戏控制台、台式计算机、大型机和超级计算机。

## 3.2 ElasticSearch简介

Elasticsearch是一个基于Apache Lucene(TM)的开源搜索引擎。无论在开源还是专有领域，Lucene可以被认为是迄今为止最先进、性能最好的、功能最全的搜索引擎库。

但是，Lucene只是一个库。想要使用它，必须使用Java来作为开发语言并将其直接集成到你的应用中，更糟糕的是，Lucene非常复杂，需要深入了解检索的相关知识来理解它是如何工作的。

Elasticsearch也使用Java开发并使用Lucene作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的RESTful API来隐藏Lucene的复杂性，从而让全文搜索变得简单。

不过，Elasticsearch不仅仅是Lucene和全文搜索，还能这样去描述它：

1.分布式的实时文件存储，每个字段都被索引并可被搜索；

2.分布式的实时分析搜索引擎；

3.可以扩展到上百台服务器，处理PB级结构化或非结构化数据。

而且，所有的这些功能被集成到一个服务里面，你的应用可以通过简单的RESTful API、各种语言的客户端甚至命令行与之交互。

上手Elasticsearch非常容易。它提供了许多合理的缺省值，并对初学者隐藏了复杂的搜索引擎理论。它开箱即用（安装即可使用），只需很少的学习既可在生产环境中使用。

Elasticsearch在Apache 2 license下许可使用，可以免费下载、使用和修改

## 3.3 HBase简介

HBase是一个分布式的、面向列的开源数据库，该技术来源于 Fay Chang 所撰写的Google论文“Bigtable：一个结构化数据的分布式存储系统”。就像Bigtable利用了Google文件系统（File System）所提供的分布式数据存储一样，HBase在Hadoop之上提供了类似于Bigtable的能力。HBase是Apache的Hadoop项目的子项目。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。

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。

与FUJITSU Cliq等商用大数据产品不同，HBase是Google Bigtable的开源实现，类似Google Bigtable利用GFS作为其文件存储系统，HBase利用Hadoop HDFS作为其文件存储系统；Google运行MapReduce来处理Bigtable中的海量数据，HBase同样利用Hadoop MapReduce来处理HBase中的海量数据；Google Bigtable利用 Chubby作为协同服务，HBase利用Zookeeper作为对应。 HBase位于结构化存储层，Hadoop HDFS为HBase提供了高可靠性的底层存储支持，Hadoop MapReduce为HBase提供了高性能的计算能力，Zookeeper为HBase提供了稳定服务和failover机制。

此外，Pig和Hive还为HBase提供了高层语言支持，使得在HBase上进行数据统计处理变的非常简单。 Sqoop则为HBase提供了方便的RDBMS数据导入功能，使得传统数据库数据向HBase中迁移变的非常方便。

## 3.4 Berkeley DB简介

Berkeley DB是一个开源的文件数据库，介于关系数据库与内存数据库之间，使用方式与内存数据库类似，它提供的是一系列直接访问数据库的函数，而不是像关系数据库那样需要网络通讯、SQL解析等步骤。

Berkeley DB是历史悠久的嵌入式数据库系统，主要应用在UNIX/LINUX操作系统上，其设计思想是简单、小巧、可靠、高性能。

Berkeley DB (BDB)是一个高性能的，嵌入数据库编程库，和C语言，C++，Java，Perl，Python，PHP，Tcl以及其他很多语言都有绑定。Berkeley DB可以保存任意类型的键/值对，而且可以为一个键保存多个数据。Berkeley DB可以支持数千的并发线程同时操作数据库，支持最大256TB的数据，广泛

用于各种操作系统包括大多数Unix类操作系统和Windows操作系统以及实时操作系统。

2.0版本或以上的Berkeley DB由Sleepycat Software公司开发，并使用基于自由软件许可协议/私有许可协议的双重授权方式提供[1]，附有源代码。开发者如果想把Berkeley DB嵌入在私有软件内需要得到Sleepycat公司的许可，若将软件同样遵循GPL发布，则不需许可即可使用。而2.0版本以下的则使用BSD授权，可自由作商业用途。

Berkeley DB最初开发的目的是以新的HASH访问算法来代替旧的hsearch函数和大量的dbm实现（如AT&T的dbm，Berkeley的 ndbm，GNU项目的gdbm），Berkeley DB的第一个发行版在1991年出现，当时还包含了B+树数据访问算法。在这以后，Berkeley DB得到了广泛的应用，成为一款独树一帜的嵌入式数据库系统。2006年Sleepycat公司被Oracle 公司收购，Berkeley DB成为Oracle数据库家族的一员，Sleepycat原有开发者继续在Oracle开发Berkeley DB，Oracle继续原来的授权方式并且加大了对Berkeley DB的开发力度，继续提升了Berkeley DB在软件行业的声誉。Berkeley DB的当前最新发行版本是6.4.9。

值得注意的是DB是嵌入式数据库系统，而不是常见的关系/对象型数据库，对SQL语言不支持（目前已经支持SQL），也不提供数据库常见的高级功能，如存储过程，触发器等。

# 第四章 系统领域分析

## 4.1 组织目标

### 4.1.1 使命和方向

集全世界的各种信息，服务人人，人人都可从中受益。

### 4.1.2 组织总目标

整合全球的信息，建立一个完善的搜索引擎，使其为每个人所用，让所有人受益，为所有信息搜索者提供更准确的服务。

## 4.2 组织业务

### 4.2.1 机构分析

搜索服务的组织机构本身较为简单，一般自成一个机构部门，最大细分可分为研发部门、运营部门及市场等平行一级，其结构大致如下图4.1所示：

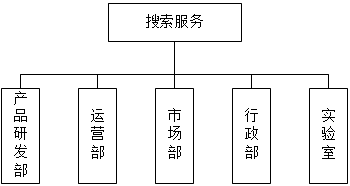


图 4.1 组织结构

Graph 4.1 Organization structure

### 4.2.2 职能/岗位

**1. 岗位分析**

该组织结构的岗位设置如下图4.2所示，其中“\*”表示可重复设置的岗位，岗位分析师组织结构优化调整的基础，为后续的业务分析奠定基石。

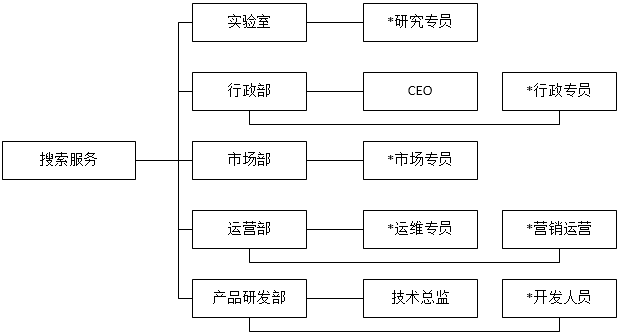


图 4.2 岗位设置

Graph 4.2 Post setting

由上岗位设置图可知，该组织岗位结构简单，业务划分明确。

**2. 职能域分析**

关于本系统的开发业务，如下图4.3为组织职能域结构，其中略去了与本系统无关的业务职能。

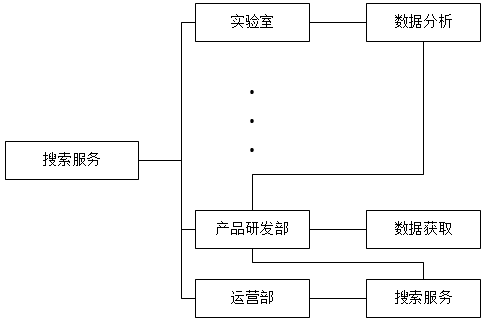


图 4.3 职能域结构

Graph 4.3 Function area structure

通过对组织职能的划分，得出组织关于本系统的业务明细，如下表4.1所示：

表 4.1 业务明细

Table 4.1 Business detail

|  |  |  |
| --- | --- | --- |
| 职能域 | 业务名称 | 业务描述 |
| 数据获取 | 页面抓取 | 最初始的数据获取 |
| 文档清洗 | 对数据进行过滤提取 |
| 数据存储 | 存储到分布式数据库 |
| 数据分析 | 权值计算 | 对数据建图分析 |
| 索引 | 创建索引 |
| 搜索服务 | 全文检索 | 关键字检索服务 |
| 快照 | 查看原始数据页面 |

### 4.2.3 业务/机构

**1. 涉众分析**

涉众是本系统业务所涉及到的实体对象，根据业务明细总结出如下表4.2所示的涉众分析报告。

表 4.2 涉众分析报告

Table 4.2 Constituents Report

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 名称 | 描述 | 期望 |
| SH0001 | 用户 | 搜索服务的使用者 | 1. 根据关键字检索信息 2. 在网络状况差的情况下查看原始快照 |
| SH0002 | 搜索服务器 | 数据检索的主观提供者 | 对索引的数据进行查找 |
| SH0003 | 爬虫程序 | 数据的生产者 | 1. 对网页文档进行爬取 2. 抽取文档信息并存储 |
| SH0004 | Web浏览器 | 用户和搜索服务器间通信的表达者 | 与服务器通信，完成数据请求和结果展示 |
| SH0005 | 服务优化程序 | 数据结果的分析者 | 对爬取的数据进行计算分析，以获取最优的检索服务 |

**2. 业务机构关系**

据以上的分析，总结得出业务机构关系表，如表4.3所示，描述了本系统业务和机构间的关系，为组织结构业务的调整的依据。

图 4.3 业务机构关系表

Table 4.3 Business and organization relation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 部门  业务名称 | 实验室 | 产品研发部 | 运营部 | 用户 |
| 页面抓取 |  | ★ | ✔ |  |
| 文档清洗 |  | ★ | ✔ |  |
| 数据存储 |  | ★ | ✔ |  |
| 权值计算 | ★ | ✔ |  |  |
| 索引 | ✔ | ★ |  |  |
| 全文检索 |  | ★ | ✔ | ✔ |
| 快照 |  | ★ | ✔ | ✔ |

说明：★表示该部门的主要业务；✔表示与该部门有关系的业务。

# 第五章 需求分析

## 5.1 需求结构

如图5.1所示，本系统的模块主要有三个，具体每个模块下又有其实现功能的组件，各模块即独立运行又互相存在依赖，其中爬虫模块是一个独立的爬虫程序，用以数据的获取和存储，而数据分析模块则又是一个Spark工程，用以对爬取的网页建立关系图并计算PageRank，而搜索服务模块则以一个独立Web应用的形式对用户提供搜索和快照服务。

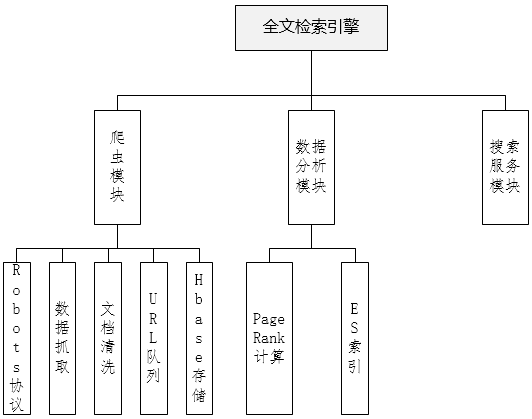


图 5.1 系统模块

Graph 5.1 System modules

爬虫程序属该系统中组件较多的一个模块，其中包括了国际上对数据爬取得文明规范Robots协议的支持组件、URL队列组件用以控制爬取的连接网络，而存储则使用到了分布式的Nosql数据库HBase，以便实现高扩展，也方便数据分析和读取。

## 5.2 用例分析

系统用例由组织业务分析而来，经过三大模块分流，其用例表示如下图5.2所示。

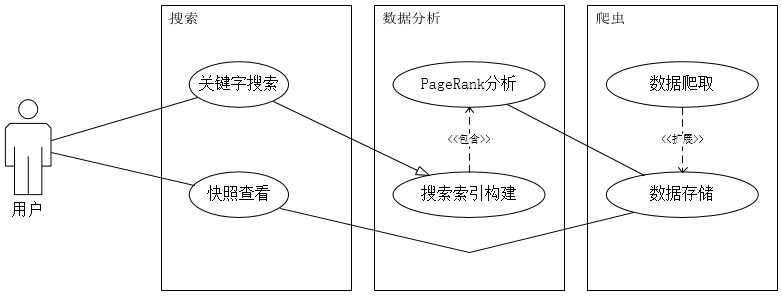


图 5.2 系统用例图

Graph 5.2 System use case

用户在搜索服务模块的用例直接表现为关键字搜索和快照查看，而其关键字搜索又构建在搜索索引之上，快照则依赖于爬虫的数据存储，数据分析PageRank包括索引的创建且来自爬虫爬取得数据。

以下是对每个用例的分析。

### 5.2.1 关键字搜索

1. 用例说明

关键字搜索用例的明细说明如下表5.1所示：

表 5.1 关键字搜索用例明细

Table 5.1 Search case detail

|  |  |
| --- | --- |
| 用例名 | 关键字搜索 |
| 参与者 | 用户 |
| 前置条件 | 用户在浏览器页面输入关键字并发起搜索 |
| 事件流程 | 1. 用户填写搜索关键字并发起请求，用例开始 2. 浏览器发送检索参数到服务器 3. 服务器解析参数 4. 服务器发起检索，请求ES集群 5. 服务器获取结果并处理包装 6. 服务器返回结果，浏览器接收，用例结束 |

2. 用例流程

关键字搜索用例处理流程如下图5.3所示：

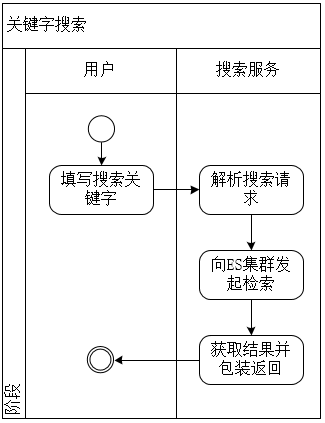


图 5.3 关键字搜索用例处理流程

Graph 5.3 Search case handle process

3. 人机交互

关键字搜索用例人机交互图如下图5.4所示：

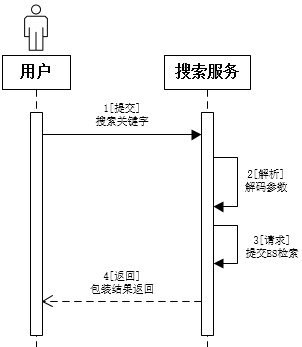


图 5.4 关键字搜索人机交互

Graph 5.4 Search case HC interaction

### 5.2.2 快照查看

1. 用例说明

快照查看用例的明细说明如下表5.2所示：

表 5.2 快照查看用例明细

Table 5.2 Snapshot case detail

|  |  |
| --- | --- |
| 用例名 | 快照查看 |
| 参与者 | 用户 |
| 前置条件 | 用户在浏览器页面得到检索结果 |
| 事件流程 | 1. 用户在搜索结果中点击快照连接，用例开始 2. 服务器接受请求并解码参数 3. 服务器从分布式存储获取源数据 4. 服务器进行页面预处理 5. 服务器返回页面，浏览器展示，用例结束 |

2. 用例流程

快照查看用例处理流程如下图5.5所示：

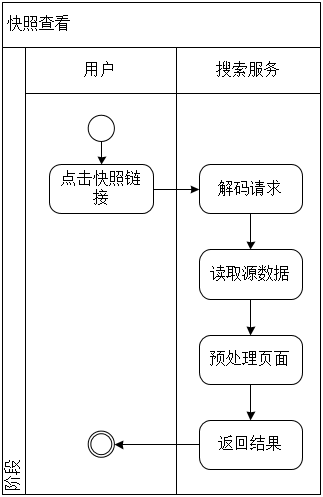


图 5.5 快照查看用例处理流程

Graph 5.5 Snapshot case handle process

3. 人机交互

快照查看用例人机交互图如下图5.6所示：

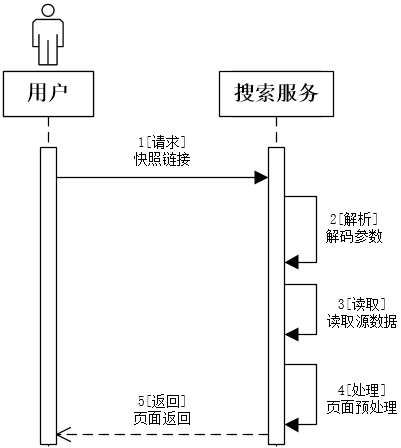


图 5.6 快照查看人机交互

Graph 5.6 Snapshot case HC interaction

### 5.2.3 PageRank分析

1. 用例说明

PageRank分析用例的明细说明如下表5.3所示：

表 5.3 PageRank分析用例明细

Table 5.3 PageRank calc case detail

|  |  |
| --- | --- |
| 用例名 | PageRank分析 |
| 参与者 |  |
| 前置条件 | 源数据已被存储 |
| 事件流程 | 1. 数据分析工程启动，用例开始 2. 向数据集群获取数据 3. 构建网络关系图 4. 迭代计算PR值 5. 将结果写入数据集群并创建索引，用例结束 |

2. 用例流程

PageRank分析用例处理流程如下图5.7所示：

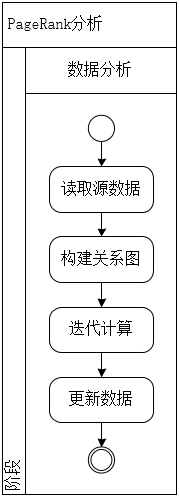


图 5.7 PageRank分析用例处理流程

Graph 5.7 PageRank calc case handle process

3. 人机交互

PageRank分析用例人机交互图如下图5.8所示：

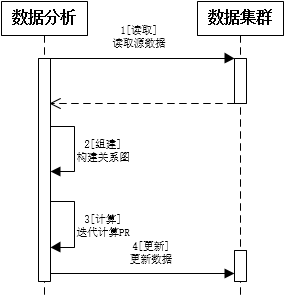


图 5.8 PageRank分析人机交互

Graph 5.8 PageRank calc case HC interaction

### 5.2.4 搜索索引构建

1. 用例说明

搜索索引构建用例的明细说明如下表5.4所示：

表 5.4 搜索索引构建用例明细

Table 5.4 Search index build case detail

|  |  |
| --- | --- |
| 用例名 | 搜索索引构建 |
| 参与者 |  |
| 前置条件 | 数据分析工程对数据的迭代计算已完成 |
| 事件流程 | 1. 数据分析工程结束计算，用例开始 2. 从网络关系图组建数据集合 3. 连接ES集群 4. 添加数据映射 5. 循环创建数据索引，直至所有数据已被添加，用例结束 |

2. 用例流程

搜索索引构建用例处理流程如下图5.9所示：

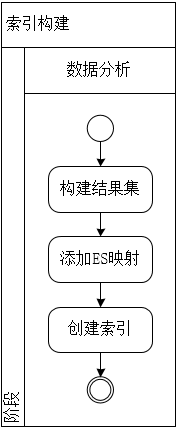


图 5.9 搜索索引构建用例处理流程

Graph 5.9 Search index build case handle process

3. 人机交互

搜索索引构建用例人机交互图如下图5.10所示：

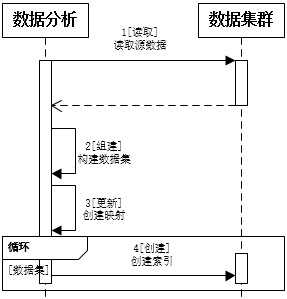


图 5.10 搜索索引构建人机交互

Graph 5.10 Search index build case HC interaction

### 5.2.5 数据爬取

1. 用例说明

数据爬取用例的明细说明如下表5.5所示：

表 5.5 数据爬取用例明细

Table 5.5 Data crawl case detail

|  |  |
| --- | --- |
| 用例名 | 数据爬取 |
| 参与者 |  |
| 前置条件 |  |
| 事件流程 | 1. 数据爬虫程序启动，用例开始 2. 从连接队列取待爬取链接 3. 解析主机robots协议 4. 抓取链接页面 5. 清洗页面数据，抽取链接并加入队列，重复以上步骤 6. 达到指定要求，程序退出，用例结束 |

2. 用例流程

数据爬取用例处理流程如下图5.11所示：

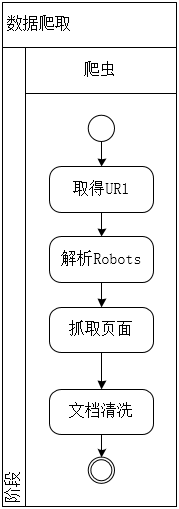


图 5.11 数据爬取用例处理流程

Graph 5.11 Data crawl case handle process

3. 人机交互

数据爬取用例人机交互图如下图5.12所示：

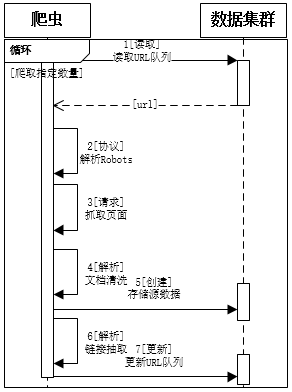


图 5.12 数据爬取人机交互

Graph 5.12 Data crawl case HC interaction

### 5.2.6 数据存储

1. 用例说明

数据存储用例的明细说明如下表5.6所示：

表 5.6 数据存储用例明细

Table 5.6 Data store case detail

|  |  |
| --- | --- |
| 用例名 | 数据存储 |
| 参与者 |  |
| 前置条件 | 爬虫成功爬取到页面数据 |
| 事件流程 | 1. 爬虫取得数据，用例开始 2. 解析文档内容，提取所有链接存入BDB队列 3. 解析文档内容，提取正文 4. 封装页面数据并按字段序列化 5. 连接HBase集群，写入页面数据 6. 重复以上，直到程序结束，用例结束 |

2. 用例流程

数据存储用例处理流程如下图5.13所示：

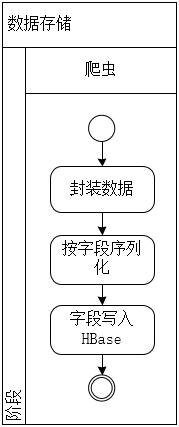


图 5.13 数据存储用例处理流程

Graph 5.13 Data store case handle process

3. 人机交互

数据存储用例人机交互图如下图5.14所示：

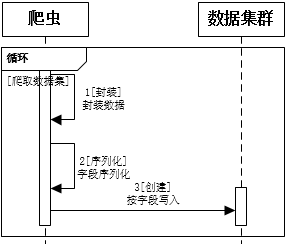


图 5.14 数据存储人机交互

Graph 5.14 Data store case HC interaction

# 第六章 系统详细设计

## 6.1 对象模型设计

### 6.1.1 业务对象

提取业务领域中与系统相关的事物，且加之本文档领域分析结果，提取出业务实体对象有：网页实体、爬虫实体、搜索服务、数据分析，这些实体有来自数据结构提炼，也有来自业务处理抽象，将在其上展开逻辑设计。

### 6.1.2 对象属性

预分析的业务抽取对象，其对象属性如下图6.1所示：

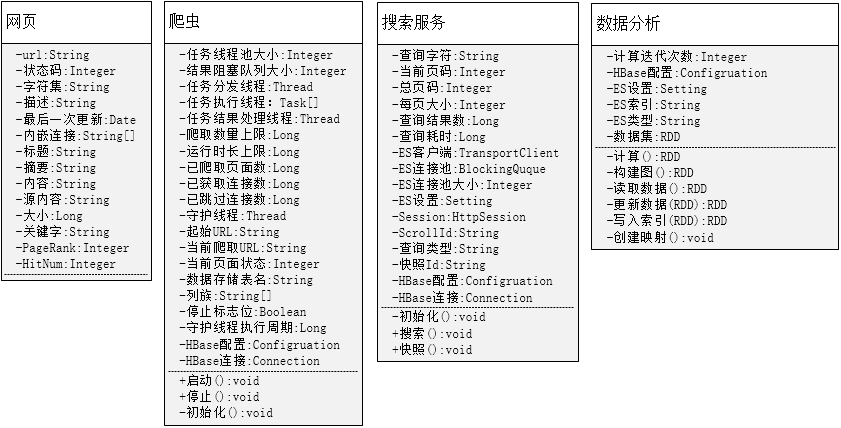


图 6.1 实体对象属性

Graph 6.1 Entity properties

### 6.1.3 对象优化调整

根据业务调整，发现对于各种独立的模块存在粘连，且对于一些不需要的对象属性进行提炼（即便会有冗余，但在本业务流程下是建议的），同时将配置信息和上下文信息进行封装以简化实体属性，达到单一职责的高内聚。以下调整并非最终状态，后根据性能优化需要灵活调整。

1. 爬虫模块调整

爬虫实体包含了配置属性和一些需要共享的对象属性，使得其他操作均需要依赖爬虫实体，但爬虫实体并不特有该类属性，因此分别提取出配置实体和上下文实体，而为了后续配置方便调整，再更新到各个配置实体（系统配置、结果管道配置、线程池配置、HBase配置），而至于爬虫队列直接使用网页实体会存在大量属性和空间浪费，为此对各个功能实现抽出URL实体和封装属性实体，而其数据统计内容单独新增一个全局统计实体，调整后的属性如下图6.2所示：

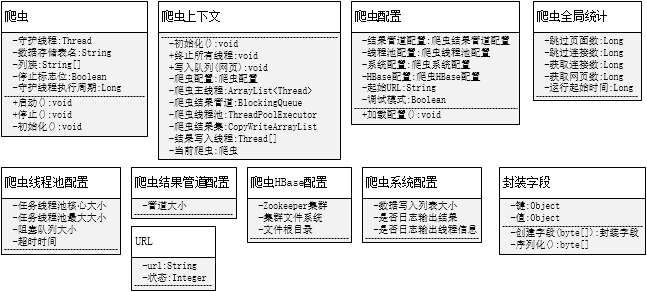


图 6.2 爬虫模块优化实体

Graph 6.2 Spider entities property optimized

1. 搜索服务模块调整

搜索模块同理抽出上下文和配置实体，同时对结果的更好处理，提炼出结果包装实体，包括页码实体（主要是包装每个页面独一无二的url，经过编码处理）和检索结果实体。优化后的实体如下图6.3所示：

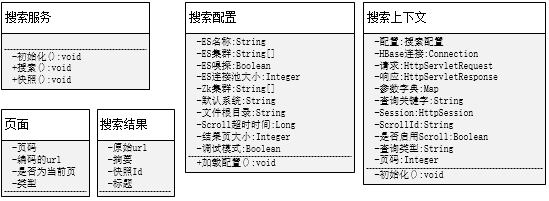


图 6.3 搜索服务模块优化实体

Graph 6.3 Search entities property optimized

### 6.1.4 对象关系

根据以上优化分析和业务流程，对各个实体对象的关系总结得出的对象关系PIM类模型如下图6.4所示，其中网页实体作为数据完整的实体连接了爬虫与数据分析模块，而数据封装作为数据存储的实体，关联了三大模块，由此可看见三大模块彼此独立又相互关联（即数据层面的关联）。

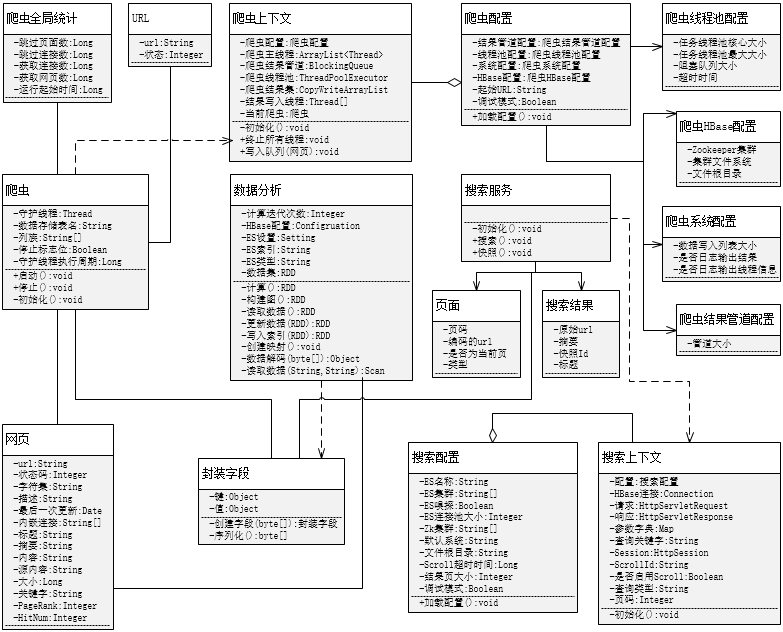


图 6.4 实体关系图

Graph 6.4 Entities relation

## 6.2 业务逻辑设计

### 6.2.1 提取基本类

1. 实体类

系统涉及的实体类有：爬虫、爬虫配置、爬虫系统配置、爬虫线程池配置、爬虫结果管道配置、爬虫HBase配置、网页、URL、封装字段、爬虫全局统计、爬虫上下文、数据分析、搜索服务、页面、搜索结果、搜索配置、搜索上下文。

2. 控制类

按照用例，设置控制类有：检索控制类、快照控制类、数据分析控制类、数据存储控制类、数据爬取控制类、索引创建控制类。

3. 边界类

各个模块边界类为搜索服务接口，爬虫线程接口，其中并不存在数据分析模块的边界类，该模块由Spark管理调度，可抽象出一个Spark任务接口。

### 6.2.2 逻辑类

1. 关键字搜索

搜索模块的关键字搜索用例逻辑类图如下图6.5所示：

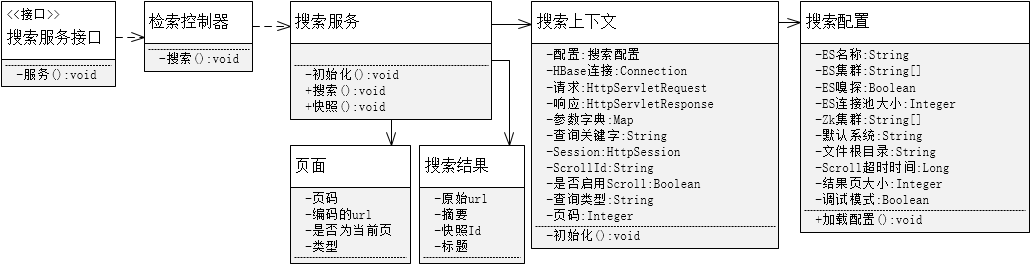


图 6.5 关键字搜索逻辑类图

Graph 6.5 Search case logic class

2. 快照查看

搜索模块的快照查看用例逻辑类图如下图6.6所示：

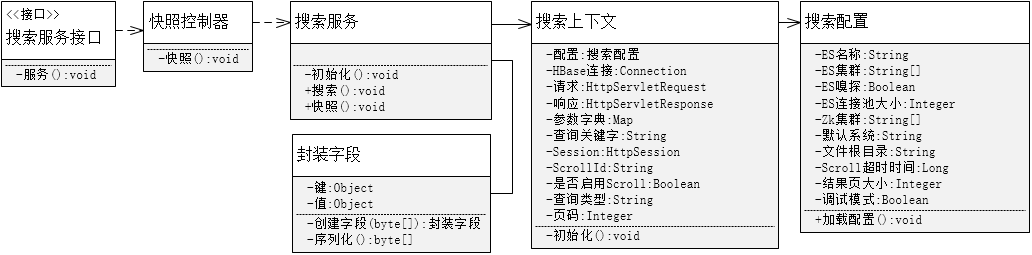


图 6.6 快照查看逻辑类图

Graph 6.6 Snapshot case logic class

3. PageRank分析

数据分析模块的PageRank分析用例逻辑类图如下图6.7所示：

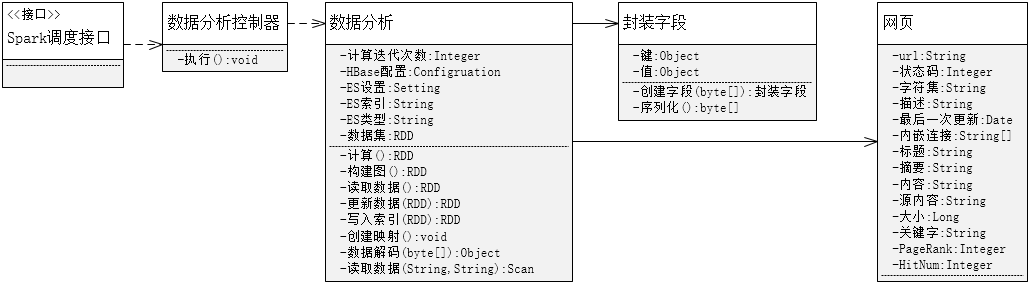


图 6.7 PageRank分析逻辑类图

Graph 6.7 PageRank calc case logic class

4. 搜索索引创建

数据分析模块的搜索索引创建用例逻辑类图如下图6.8所示：

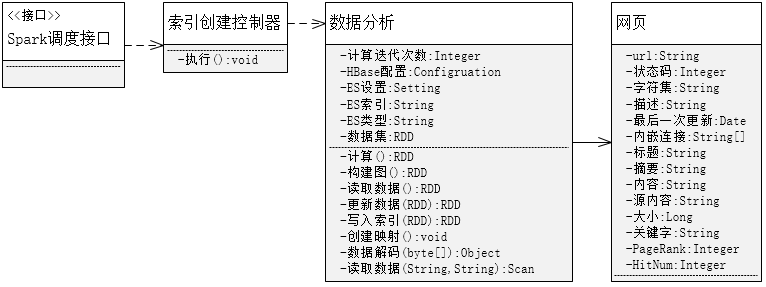


图 6.8 搜索索引创建逻辑类图

Graph 6.8 Search index case logic class

5. 数据爬取

爬虫模块的数据爬取用例逻辑类图如下图6.9所示：

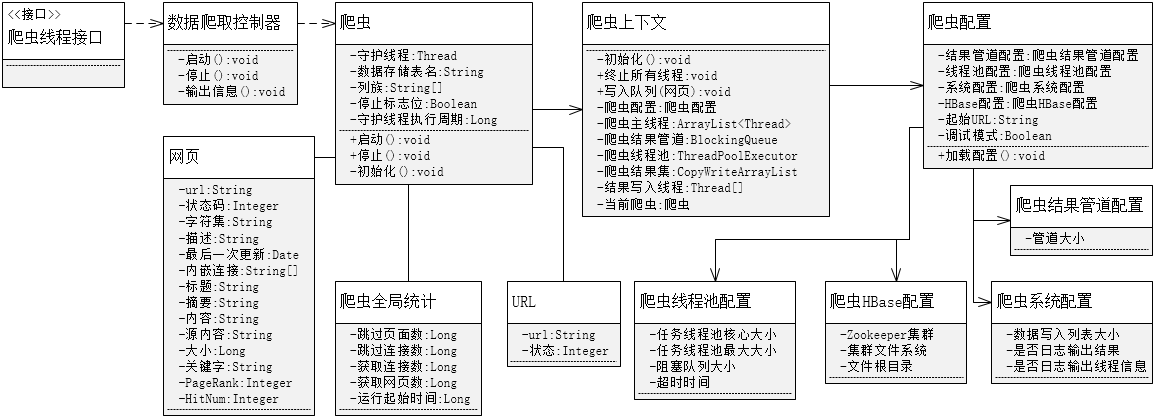


图 6.9 数据爬取逻辑类图

Graph 6.9 Data crawl case logic class

6. 数据存储

爬虫模块的数据存储用例逻辑类图如下图6.10所示：

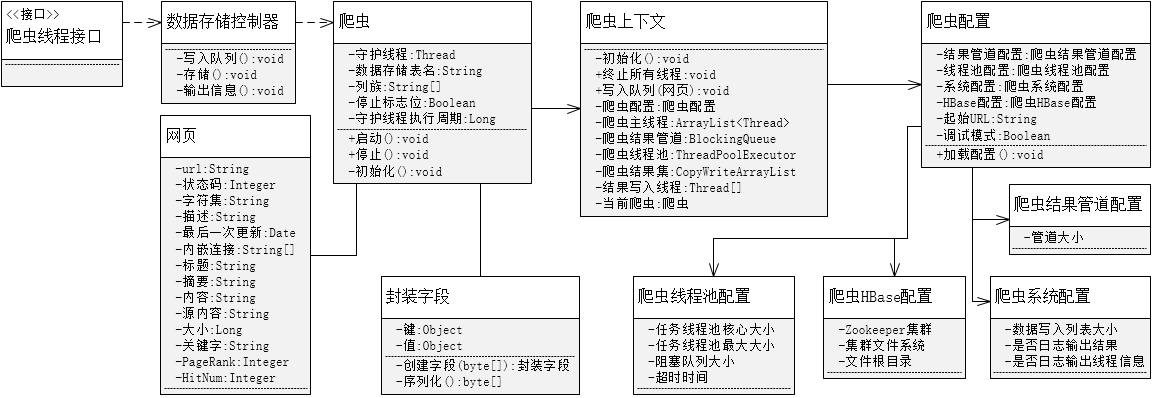


图 6.10 数据存储逻辑类图

Graph 6.10 Data store case logic class

### 6.2.3 功能交互

实现一个用例涉及到边界类、控制类和实体类三种逻辑要素，这三种类通过消息交互完成用例功能。以下为通过BCE模式描述的用例功能逻辑设计，包含功能逻辑模型和说明。

1. 关键字搜索

用户的搜索请求通过搜索服务接口传递到控制器，由搜索服务进行处理，其逻辑模型如下图6.11所示：

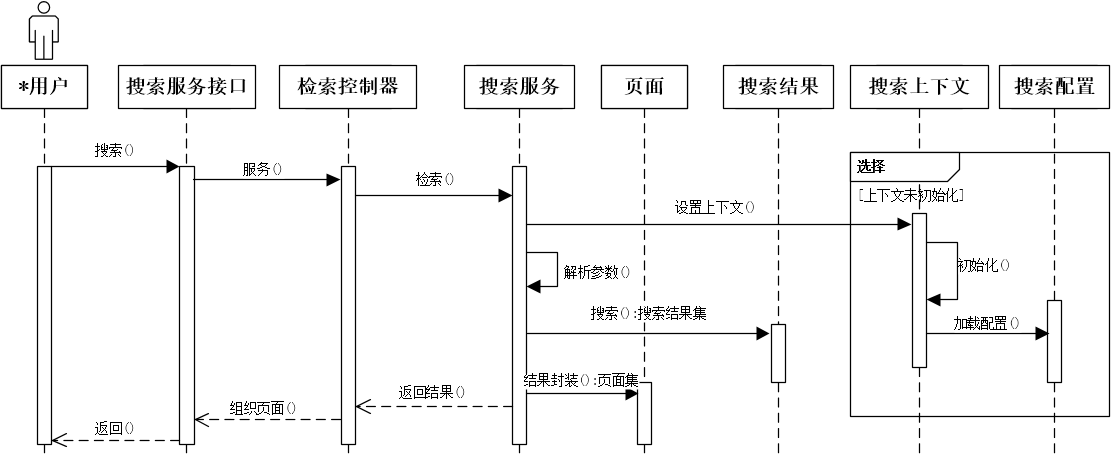


图 6.11 关键字搜索功能逻辑模型

Graph 6.11 Search case logic model

对功能逻辑的说明如下表6.1所示：

表 6.1 关键字搜索功能逻辑模型说明

Table 6.1 Search case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| \*用户 | 参与者 | \*表示任何匿名发起者 | - |
| 搜索服务接口 | 边界类 | 搜索服务对外Servlet | WebSearchServlet.java |
| 检索控制器 | 控制类 | 搜索服务调用控制 | WebSearchHandler.java |
| 搜索服务 | 实体类 | 搜索服务业务实现 | SearchService.java |
| 页面 | 实体类 | 分页数据封装实体 | PageHolder.java |
| 搜索结果 | 实体类 | 搜索结果解析存储的实体 | SearchResult.java |
| 搜索上下文 | 实体类 | 搜索执行过程上下文 | DefaultContext.java |
| 搜索配置 | 实体类 | 搜索服务控制配置 | DefaultContext$Config.class |

2. 快照查看

用户点击快照链接，通过搜索服务接口将请求交给快照控制器，其逻辑模型如下图6.12所示：

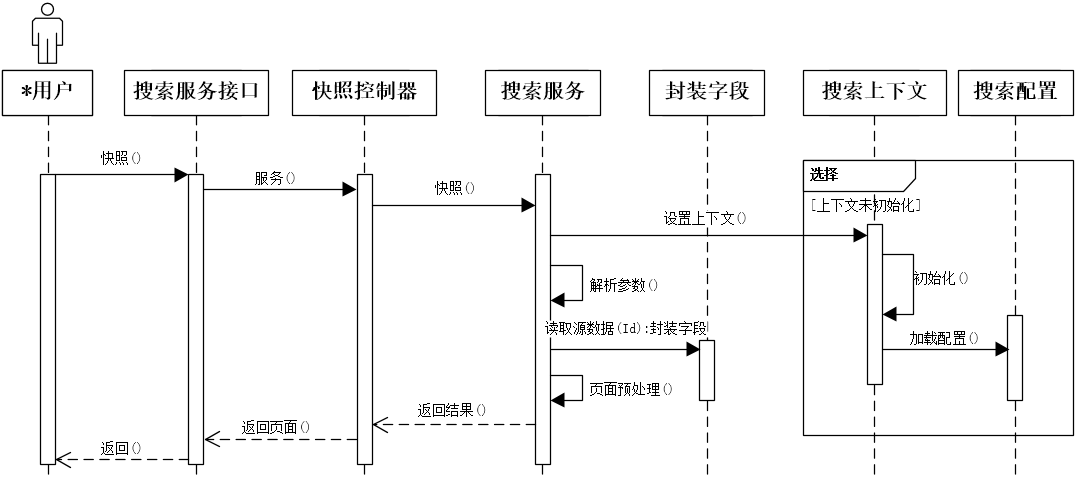


图 6.12 快照查看逻辑模型

Graph 6.12 Snapshot case logic model

对功能逻辑的说明如下表6.2所示：

表 6.2 快照查看功能逻辑模型说明

Table 6.2 Snapshot case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| \*用户 | 参与者 | \*表示任何匿名发起者 | - |
| 搜索服务接口 | 边界类 | 搜索服务对外Servlet | WebSearchServlet.java |
| 快照控制器 | 控制类 | 快照服务调用控制 | WebSearchHandler.java |
| 搜索服务 | 实体类 | 快照服务业务实现 | SnapshotService.java |
| 封装字段 | 实体类 | 源数据的存储结构实体 | PackagedValue.java |
| 搜索上下文 | 实体类 | 搜索执行过程上下文 | DefaultContext.java |
| 搜索配置 | 实体类 | 搜索服务控制配置 | DefaultContext$Config.class |

3. PageRank分析

Spark任务提交，Spark调度器执行任务，数据分析控制器处理PageRank计算流程，其逻辑模型如下图6.13所示：

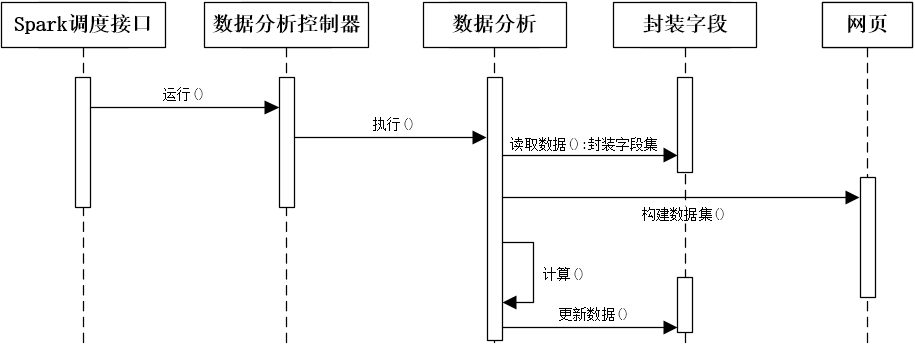


图 6.13 PageRank分析逻辑模型

Graph 6.13 PageRank calc case logic model

对功能逻辑的说明如下表6.3所示：

表 6.3 PageRank分析功能逻辑模型说明

Table 6.3 PageRank calc case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| Spark调度接口 | 边界类 | Spark任务调度执行入口 | - |
| 数据分析控制器 | 控制类 | 数据分析业务流程控制 | Analyser.scala |
| 数据分析 | 实体类 | 数据分析服务业务实现 | Analyser.scala |
| 封装字段 | 实体类 | 源数据存储的结构实体 | PackagedValue.java |
| 网页 | 实体类 | 源数据存储的逻辑结构实体 | BaseURL.java |

4. 搜索索引创建

Spark任务执行中对计算的结果进行索引创建，其逻辑模型如下图6.14所示：

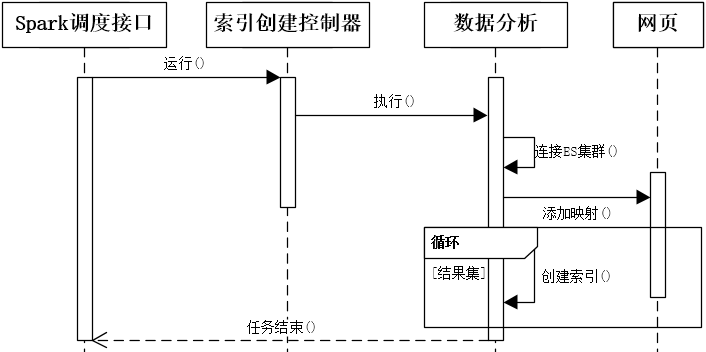


图 6.14 搜索索引创建逻辑模型

Graph 6.14 Search index case logic model

对功能逻辑的说明如下表6.4所示：

表 6.4 搜索索引创建功能逻辑模型说明

Table 6.4 Search index case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| Spark调度接口 | 边界类 | Spark任务调度执行入口 | - |
| 索引创建控制器 | 控制类 | 数据分析业务流程控制 | Analyser.scala |
| 数据分析 | 实体类 | 数据分析服务业务实现 | Analyser.scala |
| 网页 | 实体类 | 源数据存储的逻辑结构实体 | BaseURL.java |

5. 数据爬取

爬虫模块启动程序后创建爬虫线程，通过对线程接口的操作，控制爬虫线程的处理，数据爬取控制器则控制了线程按预期设定执行，其逻辑模型如下图6.15所示：

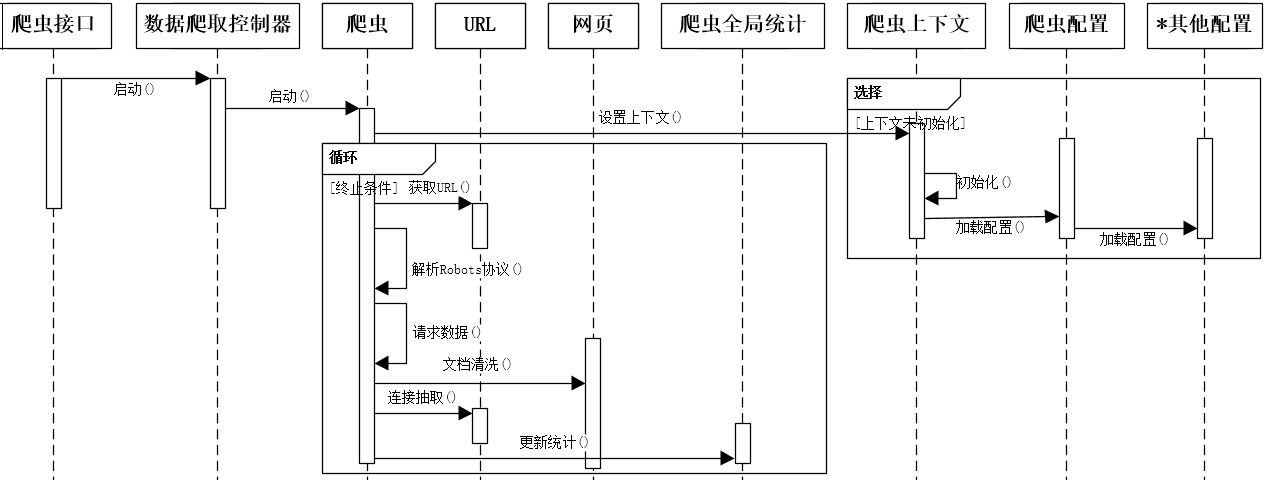


图 6.15 数据爬取逻辑模型

Graph 6.15 Data crawl case logic model

对功能逻辑的说明如下表6.5所示：

表 6.5 数据爬取功能逻辑模型说明

Table 6.5 Data crawl case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| 爬虫接口 | 边界类 | 爬虫业务程序入口 | SpiderThread.java |
| 数据爬取控制器 | 控制类 | 数据爬取流程控制 | SpiderProductor.java |
| 爬虫 | 实体类 | 爬虫业务管理实体 | Crawl.java |
| URL | 实体类 | 爬虫URL资源实体 | UrlInfo.java |
| 网页 | 实体类 | 数据解析存储的逻辑结构实体 | BaseURL.java |
| 爬虫全局统计 | 实体类 | 统计全局数据的实体对象 | Global.java |
| 爬虫上下文 | 实体类 | 爬虫运行期上下文实体 | CurrentContext.java |
| 爬虫配置 | 实体类 | 爬虫控制配置实体 | Config.java |
| \*其他配置 | 实体类 | \*表示省略其他同类配置实体 | PipleLineConfig… |

6. 数据存储

爬虫模块程序将结果通过爬虫线程的回调经由数据存储控制器统一写入数据库，其逻辑模型如下图6.16所示：

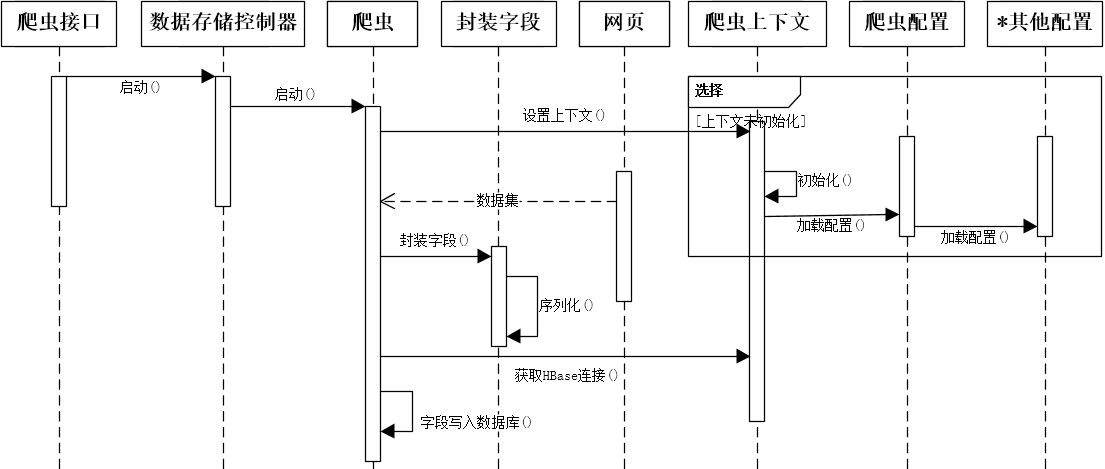


图 6.16 数据存储逻辑模型

Graph 6.16 Data store case logic model

对功能逻辑的说明如下表6.6所示：

表 6.6 数据存储功能逻辑模型说明

Table 6.6 Data store case logic model detail

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 说明 | 资源 |
| 爬虫接口 | 边界类 | 爬虫业务程序入口 | SpiderThread.java |
| 数据存储控制器 | 控制类 | 数据存储流程控制 | SpiderConsumer.java |
| 爬虫 | 实体类 | 爬虫业务管理实体 | Crawl.java |
| 封装字段 | 实体类 | 源数据存储的结构实体 | PackagedValue.java |
| 网页 | 实体类 | 数据解析存储的逻辑结构实体 | BaseURL.java |
| 爬虫上下文 | 实体类 | 爬虫运行期上下文实体 | CurrentContext.java |
| 爬虫配置 | 实体类 | 爬虫控制配置实体 | Config.java |
| \*其他配置 | 实体类 | \*表示省略其他同类配置实体 | PipleLineConfig… |

## 6.3 数据库设计

由前面对象模型设计结果PIM类模型转换为数据库关系模式，范式无要求，由于各模块通过数据关联，需要持久化的实体类有URL、网页L，其中URL为爬虫URL队列的数据实体，为提升爬虫运行速度已牺牲存储空间为代价，使用冗余的数据存储，构建两张表分别记录待爬取URL和已爬取URL以免去数据判断更新流程，而网页实体作为源数据需要后续的分析和搜索模块中使用，需要明确的存储结构，但直接进行序列化存储则会使字段查找变得难以实现，因此需要按字段存储，为此引入封装字段存储实体，多个字段实体集合为一条记录。

而本系统使用的数据库均为NoSql（即非关系型）数据库，无需数据关系，更简单的数据结构和更快的存储性能。

爬虫URL队列以BerkeleyDB为数据库存储，其简单的键值对存储结构通过构建B+数能快速对键进行检索，对于本爬虫的时效性起到较大的性能作用，其数据库设计如下表6.7所示：

表 6.7 爬虫URL队列数据库设计

Table 6.7 Crawl url queue database design

|  |  |  |
| --- | --- | --- |
|  | 类型 | 说明 |
| 键 | String | 存储键，数据检索的依据，为连接URL |
| 值 | Integer | 存储值，为对应连接的状态码，用以过滤无效结果 |

网页实体存储则使用了分布式的非关系型数据库HBase，其存储以列族为物理结构，可以灵活增加字段存储，而且能够存储的数据结构也根据简单灵活，而行键的设计采用了反转的URL字符串，这样其主机名会在前面，在存储排序时根据反转的URL行键可以将同一主机域名的数据存储在一起，从而使数据排列和扫描时的顺序更加方便相关数据的批量操作，其数据库的设计如下表6.8所示：

表 6.8 网页实体存储数据库设计

Table 6.8 Website entity database design

|  |  |  |
| --- | --- | --- |
| 行键 | 类型 | 说明 |
| Reversed URL | String | 行键位反转的URL字符串，以便数据排列规整 |
| 列族 | 列 |  |
| info | url | 页面真实连接地址 |
| status | 页面状态 |
| pageRank | PageRank值 |
| hitNum | HitNum值 |
| refs | 页面引用连接集合 |
| content | raw | 原始页面数据 |
| abs | 页面摘要 |
| desc | 文档描述 |
| keywords | 网页关键词 |
| text | 网页抽取内容 |
| title | 页面标题 |
| uptime | 最后一次更新时间 |
| charset | 字符集 |
| size | 页面大小 |

至此，系统建设的分析阶段告落，接下来将根据设计开始业务功能实现与编码测试。

# 第七章 编码和测试运行

## 7.1 项目结构

所有项目的构建均使用Maven构建，以便项目管理和依赖管理，以下是各个模块的项目结构。

1. 数据分析模块

数据分析模块的项目结果如下图7.1所示，由两种语言实现，其中Java只是使用了与其他模块的存储关联实体，主要业务则是由scala语言实现。

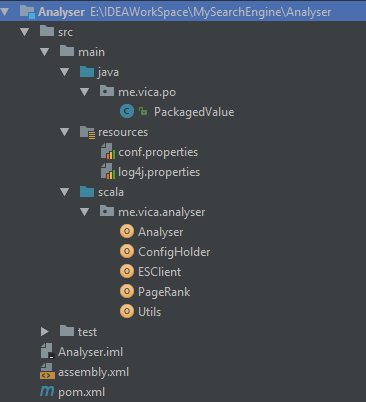


图 7.1 数据分析模块项目结构

Graph 7.1 Data analyser module structure

对项目结构的介绍如下表7.1所示：

表 7.1 数据分析模块项目结构说明

Table 7.1 Data analyser module structure detail

|  |  |  |
| --- | --- | --- |
| 包/命名空间 | 类/接口 | 说明 |
| me.vica.po | PackagedValue | 封装字段类，用以数据存取 |
| me.vica.analyser | Analyser | 数据分析主处理对象，兼控制类和实体类 |
| ConfigHolder | 配置信息掌控对象 |
| ESClient | ES操作处理对象 |
| PageRank | PageRank计算处理对象 |
| Utils | 类型隐式转换及工具操作对象 |

2. 爬虫模块

爬虫模块的项目结构如下图7.2所示，其中引入了各种数据操作模块，也是三大模块中最为复杂的一个模块。

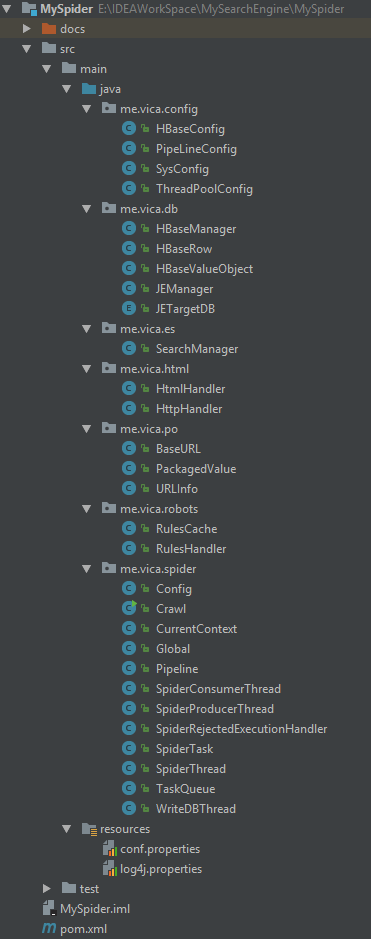


图 7.2 爬虫模块项目结构

Graph 7.2 Crawl module structure

对项目结构的介绍如下表7.2所示：

表 7.2 爬虫模块项目结构说明

Table 7.2 Crawl module structure detail

|  |  |  |
| --- | --- | --- |
| 包/命名空间 | 类/接口 | 说明 |
| me.vica.config | HBaseConfig | 爬虫HBase存储配置 |
| PipleLineConfig | 爬虫结果阻塞管道配置 |
| SysConfig | 爬虫系统配置 |
| ThreadPoolConfig | 线程池配置 |
| me.vica.db | HBaseManager | HBase操作处理实现 |
| HBaseRow | HBase行数据封装 |
| HBaseValueObject | HBase值数据封装 |
| JEManager | BerkeleyDB操作处理实现 |
| JETargetDB | BerkeleyDB目标数据库枚举 |
| me.vica.es | SearchManager | 搜索测试实现 |
| me.vica.html | HtmlHandler | Html文档清洗抽取处理 |
| HttpHandler | Http请求处理 |
| me.vica.po | BaseURL | 网页实体 |
| PackagedValue | 封装字段实体类 |
| URLInfo | URL队列使用数据实体 |
| me.vica.robots | RulesCache | Robots协议缓存 |
| RulesHandler | Robots协议处理实现 |
| me.vica.spider | Config | 全家配置管理 |
| Crawl | 爬虫业务流程实现 |
| CurrentContext | 爬虫上下文 |
| Global | 全局统计 |
| PipeLine | 结果阻塞管道 |
| SpiderConsumerThread | 爬虫结果处理线程 |
| SpiderProducerThread | 爬虫爬取控制线程 |
| SpiderRejectedExecutionHandler | 爬虫线程池注入失败处理 |
| SpiderTask | 爬虫调度任务 |
| SpiderThread | 爬虫线程边界类 |
| TaskQueue | 线程池任务队列 |
| WriteDBThread | 数据写入线程 |

3. 搜索服务模块

搜索服务模块的项目结构如下图7.3所示：

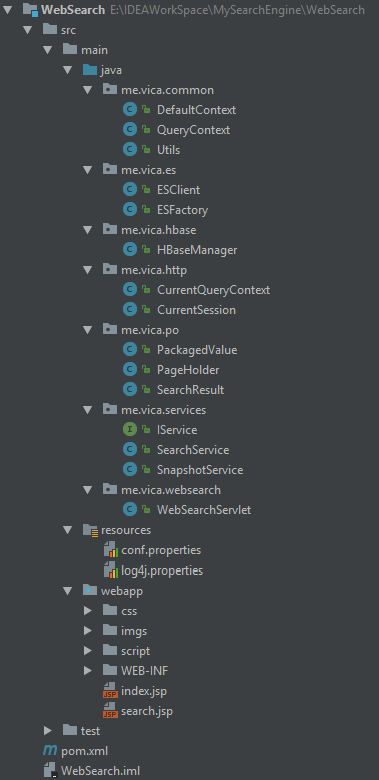


图 7.3 搜索服务模块项目结构

Graph 7.3 Search module structure

对项目结构的介绍如下表7.3所示：

表 7.3 搜索服务模块项目结构说明

Table 7.3 Search module structure detail

|  |  |  |
| --- | --- | --- |
| 包/命名空间 | 类/接口 | 说明 |
| me.vica.common | DefaultContext | 全局的上下文实体 |
| QueryContext | 本次请求上下文实体 |
| Utils | 工具类 |
| me.vica.es | ESClient | ES客户端操作类 |
| ESFactory | ES工厂，维护客户端连接池 |
| me.vica.hbase | HBaseManager | HBase数据操作类 |
| me.vica.http | CurrentQueryContext | 临时上下文对象 |
| CurrentSession | 当前会话封装类 |
| me.vica.po | PackagedValue | 封装字段实体（用于反序列化） |
| PageHolder | 分页对象封装实体 |
| SearchResult | 查询结果解析实体 |
| me..vica.services | IService | 服务接口 |
| SnapshotService | 快照服务业务实现 |
| SearchService | 搜索服务业务实现 |
| me.vica.websearch | WebSearchServlet | 请求边界类，默认Servlet |

说明：以上类/接口可能会随后续性能优化而调整。

## 7.2 编写难点

1. 数据分析ES映射的创建

由于elasticsearch-hadoop接口虽然提供了spark的支持，但也仅限于简单的支持，即索引创建和数据读取更新等，但至于更定制性的操作官方文档只提供了REST接口和Java接口，这就不得不再引入Java依赖包进行单独的映射创建操作，且该操作只能在映射未创建的时候执行一次，将其改造为scala语言交互模式，加上隐式转换，经测试可行。

2. 搜索分页的深度问题

由于ES的搜索分页需要各分片先排序然后合并再排序再取需要的页面数据，因此如果要请求的页面的页码较大则会导致大量无关的数据排序和传输，大幅降低查询性能，而为此官方的建议是使用scroll，但使用scroll意味着只能遍历每一页，且存在结果保留时常的问题，即如果用户在某一页等待到scroll超时再翻页就会重新进行一次scroll查询并遍历之前的每一页直到用户需要的一页然后返回，这个性能损耗未测试过且数据需要在内存中保存，而测试环境虚拟机的配件都较低，因此希望能通过节省内存空间而争取最大的性能，为此就是防止用户跳页，即每个页都有唯一的url，且该url在用户看来毫无价值，所以目前的解决方案是对分页数据进行封装然后编码，在用户点击结果页可用的页码时对参数进行解码，从而避免用户刻意的大幅跳页带来性能急剧下降。

## 7.3 项目测试

为方便调试，特在配置中增加了调试模式的配置（devModel），该模式下取消对数据层的依赖，绝对独立的运行每个模块。但整个系统的测试需要关闭该模式。

### 7.3.1 爬虫模块

在调试模式下开启线程日志输出，运行爬虫程序一段时间，然后日志会每隔一段时间输出当前信息，如下图7.4所示：

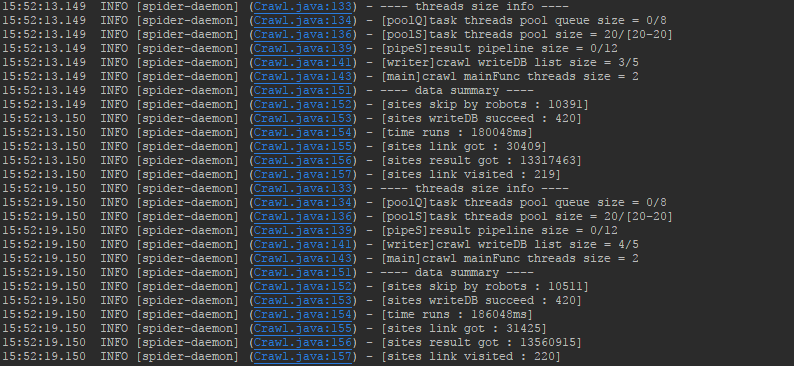


图 7.4 爬虫模块运行输出

Graph 7.4 Crawl running output

从日志可看出，爬虫运行186秒的时间里已经获取到31425个连接，而有10511个连接被Robots协议跳过，成功写入数据库的有420个页面，累计成功获取页面13.5MB。

通过JConsole对爬虫程序进行监控，数据Summary如下图7.5所示，类和线程均稳定，对老年代的垃圾回收为2次，而年轻代则为79次，即说明本程序有较多临时对象的创建，在后期优化的时候可适当增加年轻代的大小和年老代的大小，同时增加数据写入列表的大小以减少数据写入次数并减少对象创建次数。

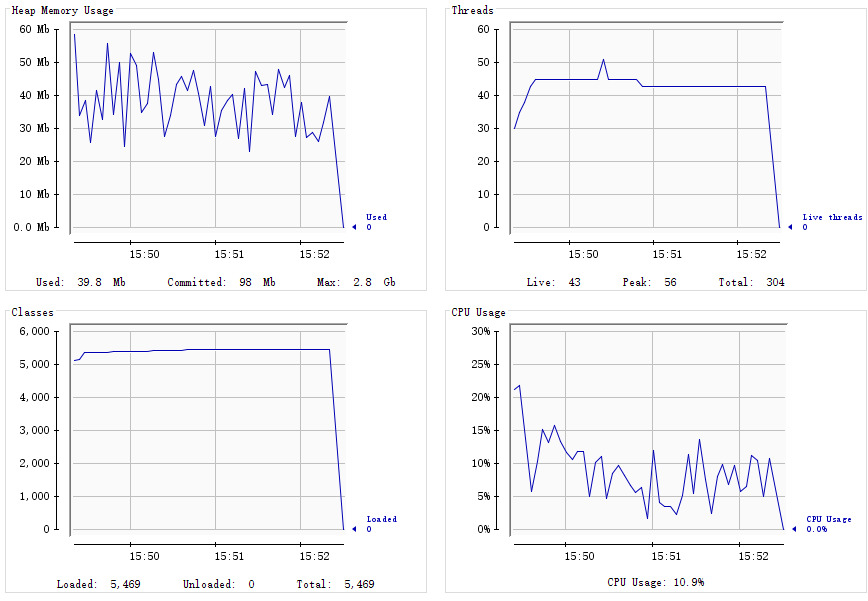


图 7.5 爬虫运行资源监控

Graph 7.5 Crawl running resource usage

再次以非调试模式运行爬虫，经过一段时间后在HBase命令行查看写入数据的信息，如下图7.6所示，扫描了表前10行记录的info:url字段，然后统计了行数。

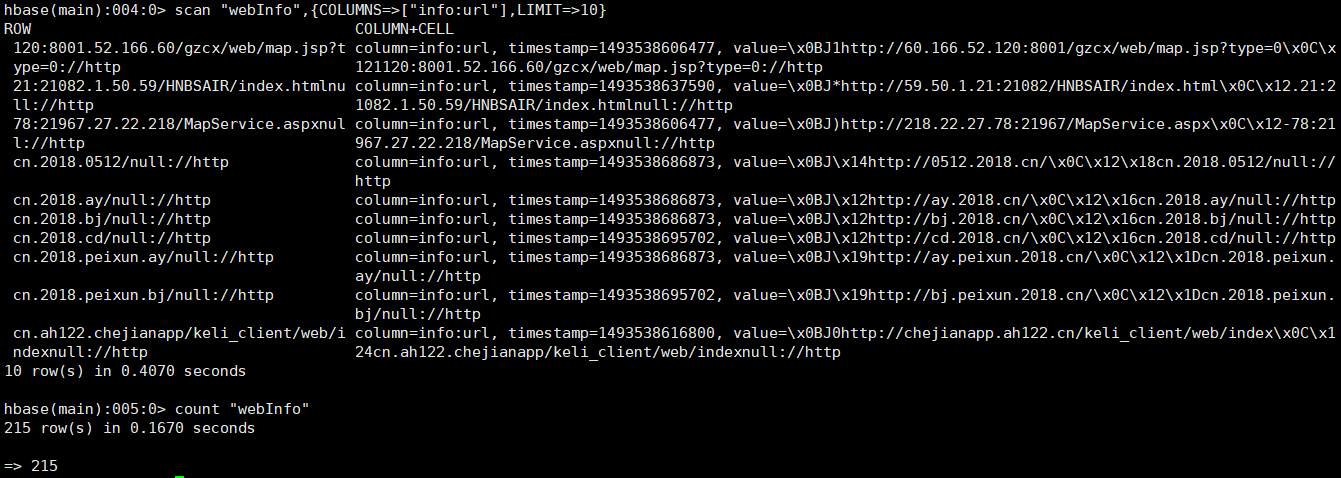


图 7.6 HBase数据查看

Graph 7.6 HBase data scan

### 7.3.2 数据分析模块

提交打包好的任务Jar包并运行，可在SparkUI查看运行状态，如下图7.7所示：

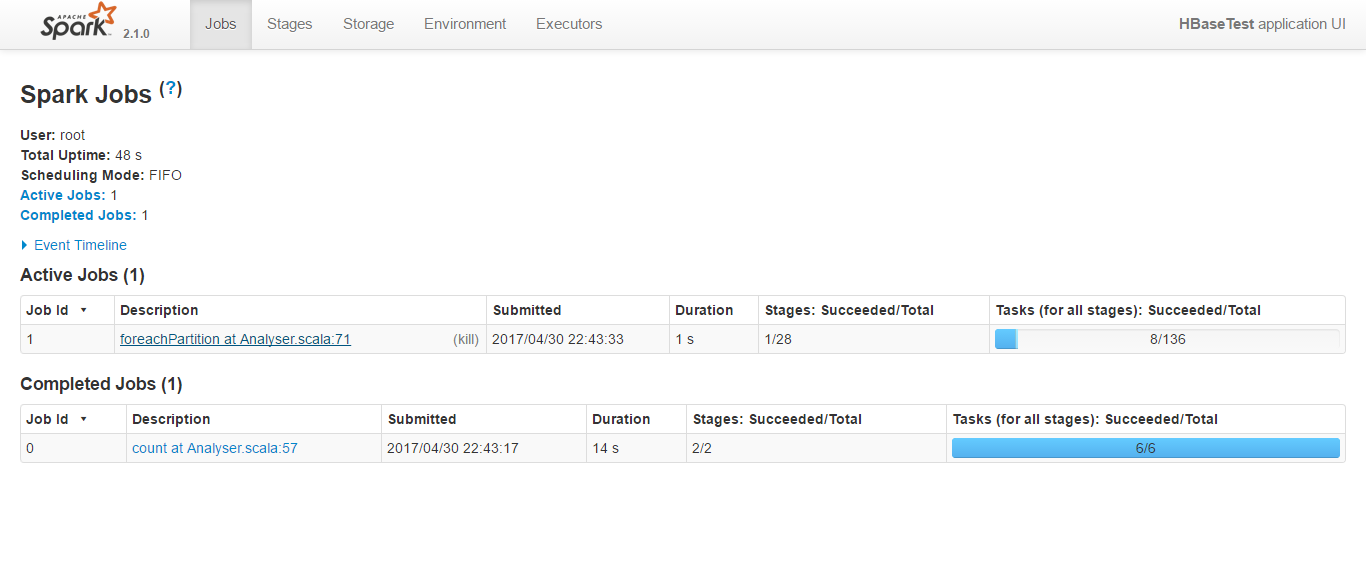


图 7.7 Spark任务状态

Graph 7.7 Spark job status

### 7.3.3 搜索服务模块

启动部署Web应用后，在搜索页输入关键字搜索即可获得相应内容列表如下图7.8所示：



图 7.8 搜索结果

Graph 7.8 Search results

然后通过点击结果连接即可跳转至结果页面，点击“VS快照“连接查看快照如下图7.9所示：



图 7.9 快照查看

Graph 7.9 Snapshot

# 参考文献

[1] 罗刚. 自己动手写网络爬虫. 北京：清华大学出版社, 2016

[2]（美）乔治. HBase权威指南. 北京：人民邮电出版社, 2013.10

[3] 心随梦飞[周路敏]. 搜索引擎选择：Elasticsearch与Solr . 博客园, <http://www.cnblogs.com/chowmin/articles/4629220.html>. 2015.07.08

[4] Elasticsearch 权威指南. <http://www.learnes.net/>. 2015.08.01

[5] Web Crawler. 维基百科,

<https://en.wikipedia.org/wiki/Web_crawler>. 2017.04.27

[6] KeenWon. 为Elasticsearch添加中文分词，对比分词器效果. KeenWon, <http://keenwon.com/1404.html>. 2014.12.12

[7] 使用Apache Spark将数据写入ElasticSearch. IteBlog, <https://www.iteblog.com/archives/1728.html>. 2016.08.10

[8] 走在前方. Elasticsearch+hbase整合. CSDN, <http://blog.csdn.net/shenfuli/article/details/49095147>. 2015.10.13

[9] Apache Spark support | Elasticsearch for Apache Hadoop [master] |Elastic. Elastic, <https://www.elastic.co/guide/en/elasticsearch/hadoop/master/spark.html>. 2017.04.27

[10] Search API | Java API [5.3] | Elastic. Elastic, <https://www.elastic.co/guide/en/elasticsearch/client/java-api/current/java-search.html>. 2017.04.27

[11] Oracle Berkeley DB Java Edition, 12c Release 1. Oracle, <http://docs.oracle.com/cd/E17277_02/html/index.html>. 2017.02.01

[12] jediael\_lu. Berkeley DB基础教程. CSDN, <http://blog.csdn.net/jediael_lu/article/details/27534223>. 2014.05.29

# 致谢

经过了这近三个月的不懈努力，我终于完成了对预期全文搜索引擎系统的设计。从开始选题到设计内容、流程的拟定，一直到系统最后的完成，每一个节点对于我来说都是崭新的尝试和挑战。这次的设计也是我在大学期间独立完成过的涉及技术最复杂、考虑因素最多的一次极具挑战性的设计。在设计该系统的时间里，我不仅仅学到了许多架构及大数据应用的知识，同时也有很多精神上或是心灵上的成长感受。通过这次设计我意识到自己独立的学习和探索能力的增长提高，通过查看各种相关的资料、书籍以及国外相关的论坛介绍，让自己的思维由简单到复杂再到简单，自己的设计也逐步优化和完善起来，每一次改进与调整都使我收益颇丰。

虽然我的设计不是很成熟，也存在诸多不足之处，但这也是我自己努力的果实，我通过这几个月细心研究和设计，其中收获到的不仅仅是年龄的成长，更是综合能力的快速提升，这是我引以自豪的地方，我相信只有经历过的人才会明白其中的酸甜苦辣。

这次的系统设计经历也使我终身受益，一个好设计不单单靠合理的系统架构，还需要各方因素综合考虑，适合的往往才是最好的，而且是要真正用心去做去完成的一套设计，是真正自己学习和研究的过程，这个过程往往比结果更重要。当然也希望这次得经历能让我在以后的工作及学习中不断激励我继续进步。

同时感谢各位老师在学习和设计期间给予我的指导和帮助。

# 附录一 运行环境的搭建

1. Spark任务运行环境的搭建

由于数据分析模块是建立在Spark任务之上的，因此需要一个能运行该任务的Spark集群。

首先，需要构建主机，以虚拟机为例，安装好3台Linux虚拟机后即可开始搭建运行环境。

（1）安装JDK，项目依赖Java8，因此建议使用JDK1.8及以上版本，安装完成后并设置环境变量，这是所有运行环境的基础，以下的其他集群部署也需要该环境。

（2）安装Scala SDK，由于Spark核心是Scala语言编写的，为此需要ScalaSDK才可以使用，而且Scala语言也是非常方便的。以Spark2.0.1为例，需要Scala 2.11.8，使用以下命令下载Scala：

|  |
| --- |
| wget <http://downloads.lightbend.com/scala/2.11.8/scala-2.11.8.tgz> |

然后使用tar命令解压，并在环境变量中进行设置：

|  |  |  |
| --- | --- | --- |
| .bash\_profile   |  |  | | --- | --- | | 1  2 | export SCALA\_HOME=/home/bigdata/services/scala-2.11.8  export PATH=$PATH:$SCALA\_HOME/bin | |

下载Spark后解压，进入conf目录，重命名一template后缀的文件去掉这个后缀，然后编辑：

|  |  |  |
| --- | --- | --- |
| spark-env.sh   |  |  | | --- | --- | | 1  2  3  4  5 | export HADOOP\_CONF\_DIR=/home/bigdata/services/hadoop-2.6.4/etc/hadoop  export JAVA\_HOME=/usr/local/jdk1.8.0\_101  export SCALA\_HOME=/home/bigdata/services/scala-2.11.8  export SPARK\_MASTER\_HOST=hdfs1  export SPARK\_MASTER\_WEBUI\_PORT=8081 | |

上面的配置信息配置了Hadoop目录及JDK、Scala SDK、主节点等信息，而下面则对节点的一些系统参数进行配置。

|  |  |  |
| --- | --- | --- |
| Spark-default.conf   |  |  | | --- | --- | | 1  2  3  4 | spark.master spark://hdfs1:7077  spark.eventLog.enabled true  spark.eventLog.dir hdfs://hdfs1:9000/spark-event  spark.dirver.memory 1g | |

最后再在slaves文件中写入其他子节点的主机名称即可，然后配置环境结束，可使用以下命令即scp命令通过ssh将已经配置好的spark传送到其他节点：

|  |
| --- |
| scp -r ~/services/scala-2.11.8 bigdata@hdfs1:~/services/  scp -r ~/services/spark-2.0.1-bin-hadoop2.6 bigdata@hdfs1:~/services/  scp ~/.bash\_profile bigdata@hdfs1:~/ |

（3）在主节点执行命令./sbin/start-all.sh即可启动spark集群。

2. HBase集群的搭建

三大模块的数据关联来自于对HBase的访问，爬虫的数据存储在HBase，数据分析的数据来源也是HBase，快照查看的源数据也是HBase，因此HBase集群贯穿着三大模块的整个生命周期。

（1）在Spark集群的基础上，安装HBase，下载本项目对应的1.2.2版本，解压后进入conf目录，先配置环境文件：

|  |  |  |
| --- | --- | --- |
| Hbase-env.sh   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10 | # The java implementation to use. Java 1.7+ required.  export JAVA\_HOME=/usr/local/jdk1.8.0\_101  # Extra Java CLASSPATH elements. Optional.  # 加入Hadoop和Zookeeper的配置文件路径  export HBASE\_CLASSPATH=/home/bigdata/services/hadoop-2.6.4/etc/hadoop:/home/bigdata/services/zookeeper-3.4.8/conf  # The directory where pid files are stored. /tmp by default.  export HBASE\_PID\_DIR=/home/bigdata/services/hbase-1.2.2/pids  # 使用Zookeeper管理  export HBASE\_MANAGES\_ZK=false | |

（2）接下来是一些节点信息的配置，包括数据目录：

|  |  |  |
| --- | --- | --- |
| Hbase-site.xml   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | <property>  <name>hbase.cluster.distributed</name>  <value>true</value>  </property>  <property>  <name>hbase.rootdir</name>  <value>hdfs://hdfs1:9000/hbase</value>  </property>  <property>  <name>hbase.zookeeper.quorum</name>  <value>hdfs1,hdfs2,hdfs3</value>  </property>  <property>  <name>hbase.master.info.port</name>  <value>60010</value>  </property>  <property>  <name>hbase.client.write.buffer</name>  <value>4194304</value>  </property> | |

（3）最后在regionservers文件中写入子节点的主机名称即可完成配置。同理使用scp命令传送到其他节点后使用bin/start-hbase.sh命令启动HBase集群。

3. 附加集群

以上两个集群的运行都离不开另外两个集群，Hadoop集群提供分布式文件存储依赖，而Zookeeper集群则提供分布式协调服务，所以在以上两个集群搭建的前提就是这两个集群已经存在。

4. ElasticSearch集群

该集群相对较好搭建，只需下载官方提供的程序包，解压即可，如需要特殊配置可修改conf目录下的elasticsearch.yml文件，而我们的集群不仅仅是本地访问，为此需要更改该配置文件下的network配置项为0.0.0.0，即解除访问限制，然后运行./bin/elasticsearch即可启动，它会自动发现同一网段下的同名节点为一个集群。

# 附录二 模块核心代码

1. 爬虫模块核心代码

爬虫的核心就是任务生产者和任务消费者，生产者线程（SpiderProducerThread）部分代码如下：

|  |
| --- |
| int startUrlIndex = 0; while (!getStopped()) {  Map.Entry<String, URLInfo> entry = null;  if(firstRun){  if(startUrlIndex < startUrl.length) {  entry = new AbstractMap.SimpleEntry<String, URLInfo>(startUrl[startUrlIndex], null);  startUrlIndex ++ ;  }else {  firstRun = false;  continue;  }  }else {  entry = queue.readUrl();  }  if (entry != null && entry.getKey() != null) {  if(visited.search(entry.getKey(), null)!=null){  continue;  }  if(!RulesHandler.*robots*(entry.getKey())){  // Robots协议支持  Global.*mergeSitesSkip*(1);  continue;  }  try {  pipeline.putBlocking(executor.submit(new SpiderTask(entry.getKey())));  emptySzie = 0;  } catch (InterruptedException e) {  *logger*.error(e.getMessage(),e);  }  }else {  if(emptySzie > 10){  *logger*.error("over 30s has empty queue, system exit.");  CurrentContext.*getCrawl*().stop();  break;  }  emptySzie ++;  try {  Thread.*sleep*(3000);  } catch (InterruptedException e) {  *logger*.error(e.getMessage(),e);  }  } } executor.shutdown(); |

而消费者（SpiderConsumerThread）部分代码如下：

while (!getStopped()) {  
 BaseURL url = null;  
 if (pipeline != null) {  
 if(pipeline.size()>0){  
 try {  
 Future<BaseURL> future = pipeline.takeBlocking();  
 url = future.get();  
 if(url == null){  
 continue;  
 }  
 //更新爬取数据大小  
 Global.*mergeBytesReceived*(url.getByteSize());  
 Global.*mergeSitesReceived*(1);  
 Global.*mergeSitesGot*(url.getUrlReferences().length);  
 //写队列  
 queue.write(url.getUrlReferences(), false);  
 visited.write(url.getInfo(), false);  
 *logger*.debug("["+url.getUrlReferences().length+" links get] url="+url.getUrl());  
 //数据写入HBase  
 if(url.getStateCode()!=200)  
 continue;  
 CurrentContext.*writeDb*(url);  
 if(CurrentContext.*getConfigContext*().getSysConfig().isResultLogOut()){  
 *logger*.info("get site result={"+url.getTitle()+":{"+url.getContext()+"},"+url.getLastUpdateTime()+"}");  
 }  
 } catch (Exception e) {  
 *logger*.error(e.getMessage(),e);  
 }  
 }  
 } else {  
 *logger*.debug("pipe was not defined. cancel it.");  
 break;  
 }  
}

2. 数据分析模块

数据分析模块代码则以主类任务的流程有关，其核心代码部分如下：

|  |
| --- |
| //加载配置 *ConfigHolder*("conf.properties")  **def** main(args: Array[String]): Unit = {   // 本地模式运行,便于测试  **val** sparkConf = **new** SparkConf().setMaster("local[2]").setAppName("HBaseTest")  sparkConf.set("es.index.auto.create", "false")  sparkConf.set("es.nodes", ConfigHolder.*getEsServers*.mkString(","))  **val** sc = **new** SparkContext(sparkConf)   */\*\*  \* 创建HBaseRDD  \*  \** ***@param tabName*** *表名称  \** ***@return*** *\*/* **def** createHBase(tabName: String) = {  // 创建hbase configuration  **val** hBaseConf = HBaseConfiguration.*create*()  hBaseConf.set(TableInputFormat.*INPUT\_TABLE*, tabName)  hBaseConf.set("hbase.zookeeper.quorum", ConfigHolder.*getZkQuorum*)  hBaseConf.set("hbase.rootdir", ConfigHolder.*getRootDir*)  hBaseConf.set("fs.defaultFS", ConfigHolder.*getDefaultFS*)   // 从数据源获取数据  **val** hbaseRDD = sc.newAPIHadoopRDD(hBaseConf, *classOf*[TableInputFormat], *classOf*[ImmutableBytesWritable], *classOf*[Result])  hbaseRDD  }   //缓存HBase数据，根据Map的Key即URL进行Hash分5个区  **val** hbaseResult = *createMap*(createHBase("webInfo"))  .partitionBy(**new** DomainNamePartitioner(5))  .persist   **val** count = hbaseResult.count   //获取URL与URL References的Tuple并缓存，用以计算  **val** links = hbaseResult.map { **case** (url, maps) => (url, maps("refs").asInstanceOf[Array[String]]) }  .persist   //计算迭代12次并缓存，一般10次应该就收敛了  **val** ranks = PageRank.*calculate*(links, count, 12)  .persist   //与HBase结果集合并  **val** result = hbaseResult.join(ranks)   //更新HBase数据库中PageRank的记录  result.foreachPartition(r => {  **val** hBaseConf = HBaseConfiguration.*create*()  hBaseConf.set("hbase.zookeeper.quorum", ConfigHolder.*getZkQuorum*)  hBaseConf.set("hbase.rootdir", ConfigHolder.*getRootDir*)  hBaseConf.set("fs.defaultFS", ConfigHolder.*getDefaultFS*)  **val** connection = ConnectionFactory.*createConnection*(hBaseConf)  **val** table = connection.getTable(TableName.*valueOf*("webInfo")).asInstanceOf[HTable]  **val** puts = **new** util.ArrayList[Put]  r.foreach {  **case** (url, (maps, pageRank)) => {  **val** key = Bytes.*toBytes*(maps("key").asInstanceOf[String])  **val** p = **new** Put(key)  p.addColumn(Bytes.*toBytes*("info"), Bytes.*toBytes*("pageRank"), **new** PackagedValue(pageRank, url).getBytes)  puts.add(p)  }  }  table.put(puts)  table.flushCommits()  table.close()  connection.close()  })   //创建ES映射  **val** mapping = Utils.*createEsMapping*("fulltext", x => {  **val** fiels = *Map*(/\*field -> boost\*/  "keywords" -> 3,  "description" -> 2,  "title" -> 5,  "text" -> 1)  fiels.foreach(m => {  x.startObject(m.\_1)  .field("type", "text")  .field("analyzer", "ik\_smart")  .field("search\_analyzer", "ik\_smart")  .field("boost", m.\_2)  .endObject()  })  x.startObject("pagerank")  .field("type", "float")  .field("index", "not\_analyzed")  .endObject()  x.startObject("key")  .field("type", "string")  .field("index", "not\_analyzed")  .endObject()  x.startObject("url")  .field("type", "string")  .field("index", "not\_analyzed")  .endObject()  })   **import** Utils.\_  **val** client = ESClient.*remote* **val** rep = client.putMapping("website", "fulltext", mapping.string)  client.close()  **if** (!rep.isAcknowledged) {  **throw new** RuntimeException("mapping create failed.")  }   //创建ES索引  **import** org.elasticsearch.spark.rdd.Metadata.\_  **val** indices = result.map {  **case** (url, (maps, pk)) =>  (*Map*(*ID* -> url.reversedURL), (maps - "refs") ++ *Map*("pagerank" -> pk, "url" -> url))  }   indices.saveToEsWithMeta("website/fulltext")   //执行以上计算  result.count()  } |

3. 搜索服务模块

搜索服务部分核心代码如下：

|  |
| --- |
| //获取参数 String qstr = request.getQueryString().trim().substring(2); PageHolder holder = null; try {  if (qstr.contains("query=")) {  holder = new PageHolder();  String[] args = qstr.split("&");  for (String arg : args) {  String[] kv = arg.split("=");  if (kv[0].equals("query")) {  holder.setQuery(kv[1]);  } else if (kv[0].equals("type")) {  holder.setType(kv[1]);  } else if(kv[0].equals("page")){  holder.setPageIndex(Integer.*valueOf*(kv[1]));  }  }  } else {  holder = Utils.*decodeRequestQueryString*(qstr);  }  if (holder == null || holder.getQuery() == null || holder.getQuery().trim().length() < 1) {  throw new IllegalArgumentException("query error! holder is empty. holder=" + holder);  }  holder.setQuery(URLDecoder.*decode*(holder.getQuery(),"utf-8")); } catch (Exception e) {  *logger*.error(e.getMessage(), e);  return false; } if(request.getQueryString() == null || request.getQueryString().startsWith("s=")){  //同步请求参数  request.setQueryString(holder.getQuery());  request.setPage(holder.getPageIndex());  request.setQueryType(getType(holder.getType())); } ESClient client = new ESClient(ESFactory.*getClient*()); //搜索 SearchResponse response = client.search(  holder.getQuery(), getType(holder.getType()),  request.isUseScroll(), request.getScrollId(), holder.getPageIndex()); List<SearchResult> result = new ArrayList<SearchResult>(); if (response.getHits().getHits().length > 0) {  //取结果  for (SearchHit hit : response.getHits().getHits()) {  SearchResult sr = new SearchResult();  sr.setTitle(getString(hit, "title", true));  sr.setContent(getString(hit, "text", true));  sr.setKey(getString(hit, "key", false), holder.getQuery());  sr.setUrl(getString(hit, "url", false));  sr.setSize(response.getHits().getHits().length);  result.add(sr);  } } int pageSzie = (int) (response.getHits().getTotalHits() / DefaultContext.*getConfig*().getEsPageSize()); if (response.getHits().getTotalHits() % DefaultContext.*getConfig*().getEsPageSize() != 0) {  pageSzie += 1; } int pageStart = 1; int pageEnd = pageSzie; int mid = *pagesPerPage*/2; if(*pagesPerPage* < pageEnd ){  pageEnd = *pagesPerPage*; } if(mid < holder.getPageIndex() + 1){  int offset = holder.getPageIndex() - mid + 1;  pageEnd += offset;  pageStart += offset; } if(pageEnd > pageSzie){  pageEnd = pageStart; } //页面信息 List<PageHolder> pages = new ArrayList<PageHolder>(); for(int i = pageStart; i <= pageEnd; i++){  PageHolder p = new PageHolder();  p.setType(holder.getType());  p.setQuery(holder.getQuery());  p.setPageIndex(i);  p.setSelected(holder.getPageIndex() == i);  p.build();  pages.add(p); } request.getRequest().setAttribute("result", result); request.getRequest().setAttribute("query", holder.getQuery()); request.getRequest().setAttribute("pages", pages); request.getRequest().setAttribute("totalSize", response.getHits().getTotalHits()); request.getRequest().setAttribute("totalPage", pageSzie); request.getRequest().setAttribute("tokenTime", response.getTookInMillis()); |

快照服务的部分核心代码如下：

|  |
| --- |
| QueryContext.RequestLocal request = QueryContext.*getRequest*(); HBaseManager manager = DefaultContext.*gethBaseManager*(); System.*out*.println("snapshot key="+request.getSnapKey()); Result result = manager.select("webInfo", request.getSnapKey()); StringBuffer html = new StringBuffer(); if (result.isEmpty()) {  //Utils.handNotFound(null, request.getResponse());  return false; } else {  //组装快照页面  String rowhtml = PackagedValue.*create*(  result.getValue(Bytes.*toBytes*("content"),  Bytes.*toBytes*("raw"))).getValue().toString();  String url = PackagedValue.*create*(  result.getValue(Bytes.*toBytes*("info"),  Bytes.*toBytes*("url"))).getValue().toString();  String date = PackagedValue.*create*(  result.getValue(Bytes.*toBytes*("content"),  Bytes.*toBytes*("uptime"))).getValue().toString();  String charset = PackagedValue.*create*(  result.getValue(Bytes.*toBytes*("content"),  Bytes.*toBytes*("charset"))).getValue().toString();  StringBuffer temp = new StringBuffer();  rowhtml = rowhtml.replace("<title>", "<title>V快照 - ");  temp.append(rowhtml.substring(0, rowhtml.indexOf("<body")));  String body = rowhtml.substring(temp.length());  body = body.replace("<body", "<div")  .replace("</body>", "</div></div></body>");  temp.append("<body>\n")  .append("<div style=\"height: 120px; background-color: #909090; color: #121212; padding: 10px 20px; margin-bottom: 20px;\">\n")  .append("<h4 style=\"font-size: 14px;\">\n")  .append("V快照来自关键词\n")  .append("<span style=\"color:red;\">")  .append(request.getQueryString())  .append("</span> &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; 快照时间：\n")  .append("<span style=\"color:darkmagenta;\">")  .append(date)  .append("</span>\n")  .append("</h4>\n")  .append("<h5 style=\"font-size: 12px;\">访问原网页：\n")  .append("<a href=\"")  .append(url)  .append("\">")  .append(url)  .append("</a>\n")  .append("</h5>\n")  .append("<h5 style=\"font-size: 11px; color: #701030; text-align: right;\">\n")  .append(" VSearch和该网页的作者无关，不对其内容负责。V快照谨为网络故障时之索引，不代表被搜索网站的即时页面。\n")  .append("</h5>\n")  .append("</div>\n")  .append("<br/>\n")  .append("<hr/>\n")  .append("<div style=\"position: relative\">");  temp.append(body);  html.append(temp);  request.getResponse().setCharacterEncoding(charset); }  try {  HttpServletResponse response = request.getResponse();  PrintWriter writer = response.getWriter();  writer.write(html.toString());  writer.flush(); } catch (Exception e) {  *logger*.error(e.getMessage(), e);  return false; } return true; |