

目錄

简介	1.1
1. 入门知识部分	1.2
1.1 垂直搜索引擎基础知识	1.2.1
1.2 Solr基础知识及安装	1.2.2
1.3 Solr5.5 集成 Tomcat8	1.2.3
2. 搜集信息部分	1.3
2.1 使用Heritrix抓取数据	1.3.1
2.2 利用jsoup提取网页中有用信息	1.3.2
2.3 基于webmagic爬虫框架的数据抓取	1.3.3
3. 整理信息部分	1.4
3.1 Solr5配置文件参数解析	1.4.1
3.2 Solr整合中文分词器mmseg4j	1.4.2
3.3 空间搜索原理及相关算法	1.4.3
3.4 Solr空间搜索配置及实践	1.4.4
3.5 导入Mysql数据到Solr中	1.4.5
4. 接受查询部分	1.5
4.1 查询方法及参数说明	1.5.1
4.2 搜索服务API	1.5.2
4.3 对搜索结果进行筛选和排序	1.5.3
4.4 基于Ionic的客户端设计	1.5.4
5. SolrCloud部分	1.6
5.1 SolrCloud基础	1.6.1
5.2 借助Docker技术的Solr集群实现	1.6.2

用Solr构建垂直搜索引擎

简介

这是笔者毕业设计的项目『利用Lucene和Solr实现旅游景点位置感知搜索引擎』，通过三个多月的学习和工作，自认为完成的还可以，想把自己的学习内容分享出来。本文档是系列教程，希望系统介绍垂直搜索引擎的开发构建过程，鉴于笔者水平有限，难免有错误和不足，欢迎指正。此外，采用Gitbook的形式发布，希望能对该系列教程持续改进。

项目

Gitbook链接：<https://www.gitbook.com/book/fliaping/create-your-vertical-search-engine-with-solr/details>

作者博客系列文章(备用链接)：

[1]"【总述】用Solr构建垂直搜索引擎", Payne's Blog, 2016. [EB/OL]. Available: <https://blog.fliaping.com/create-your-vertical-search-engine-with-solr/>. [Accessed: 03- Jun- 2016].

项目源码：

[2]"paynexus/trip-search", GitHub, 2016. [EB/OL]. Available: <https://github.com/fliaping/trip-search>. [Accessed: 03- Jun- 2016].

备用链接：<http://git.oschina.net/fliaping/trip-search>

摘要

由于通用搜索引擎数据源的广泛性，对于一些专业性或者特定领域的搜索结果并不能令人满意，所以诞生了垂直搜索引擎。它是通用搜索引擎的细分和延伸，只聚焦于某一特定主题，因此也叫作主题搜索。常见的领域有汽车产业、法律信息、医药信息、学术文献、旅游信息。

本项目实现了具有位置感知功能的旅游景点垂直搜索引擎。以开源搜索框架Solr和Lucene为基础，以另外一些开源项目例如Heritrix、webmagic、Zookeeper、Ionic、gradle、jetty等为工具，并在相关文档和技术博客的帮助下，完成了整个垂直搜索引擎系统的开发。用到的技术主要有网络爬虫、HTML解析、中文分词、文档索引、空间搜索、RESTful Web Service、Ajax、Hybrid App、容器技术Docker、SolrCloud、集群等。本文将进行项目背景的说明，技

术原理上的概述，构架层面的分析以及项目的测试和交付。通过对搜索引擎技术流程分为搜集信息、整理信息、接受查询这三个部分，将每个模块进行阐述，并辅以相关的图表，对搜索引擎技术在宏观层面上进行解释说明。总的来说本项目是一个“可用于生产环境的垂直搜索引擎原型”。另外，比较详细的开发过程以及用到的技术细节请参考文末的相关链接和作者写的博客文章。

ABSTRACT

Because of the comprehensive source of general search engine, its results can't satisfy our needs in specific segment or individual verticals, so vertical search engine was born. It is subdivided and extended from the general web search engine, which focuses on a specific theme. They are also called specialty or topical search engines. Common verticals include shopping, the automotive industry, legal information, medical information, scholarly literature, and travel.

This project is vertical search engine about travel sights with spatial search. It based on open source search framework Solr and Lucene, and other open source such as Heritrix, webmagic, Zookeeper, Ionic, gradle, jetty as tools. With the assistance of related technical documents and blogs, the development of this project was finished finally. The main technologies were used include web spider, HTML extract, Chinese analyzer, document indexing, spatial search, RESTful web service, Ajax, Hybrid App, container technology(Docker), SolrCloud, cluster and so on. This thesis will make introduction about the background of project, summary of technical theory, analysis in architecture, testing and deployment. According to the process of search engine, we divide it into three parts – collect information, index information, accept query. I will illustrate every module with diagram and words, thus, search engine technology will be explained in macrostructure. To sum up, this project is a prototype of vertical search engine that can be used in production environment. Besides, if you want to learn more about the detailed development process and technology implementation, you can reference the auctorial blogs and other articles through the links in the end of the paper.

1. 入门知识部分

本章内容主要介绍垂直搜索引擎的基础知识，以及开源搜索框架Solr的安装。

引言/废话

搜索引擎是大家平时使用最广泛的网络应用之一，它是普通网民接触互联网的入口，也是网络信息的搜集系统，其重要性不言而喻。我认为一个优秀的搜索服务应该实现一些基本的要求：

- 准确回应用户搜索目的
- 提供公正的结果排序

然而，目前最大的中文搜索引擎，在经历了[血友病吧事件](#)、[魏则西事件](#)之后，网民已经失去对其的信任，甚至很多人产生了厌恶之情，且不说是否有人落井下石，或者说是罪有应得，总之，它带给大家的是不好的用户体验。吐槽到此为止，从这里开始我们进入正题。

搜索引擎基本原理

众所周知，我们浏览的网页其实是一个个文本文件（动态生成的也算，总之浏览器接受到的就是文本），并且这些文本来自于不同的站点和服务器，例如新浪网在新浪的服务器上。截止2014年，[全球互联网站点数量超过十亿](#)，大部分站点之间数据没有关联，我们如何能在这十多亿的站点上查到我们需要的信息？总不能一个一个站点询问吧，因此我们需要一个包含这十多亿站点信息的Master来告诉我们结果，这个Master就可以看做搜索引擎，它的基本性质有如下几点：

- 基本包含整个互联网的信息
- 对信息进行了初步的整理
- 能及时更新信息，达到基本和源站点信息同步
- 能在短时间回应查询请求

按照[Wikipedia-搜索引擎](#)的步骤及分类方法，分为：

搜集信息 -> 整理信息 -> 接受查询

其中『整理信息』及『接受查询』的过程，大量应用了[文本信息检索技术](#)，并根据网络超文本的特点，引入了更多的信息。

这里不引用Wikipedia上的原文，我对这些概念进行一些简化和总结。

搜集信息

这个Master如何才能包含整个互联网的信息？它没有权利直接访问站点数据存储系统，那么它就只能通过模拟用户浏览来把网页保存下来，但浏览网页是要通过URL来定位的，它原本没有所有链接的列表，但通过[网络爬虫（Web spider/Web crawler）](#)就可以遍历所有网页，当然这个过程基本都是自动的。因为网页的访问时基于HTTP（Hypertext Transfer Protocol），意为超文本传输协议，这里的超文本指文本页面中包含有指向到其他页面的链接，网页通过超链接彼此联系。所以理想状态下，爬虫利用少量的起始页面，通过解析已访问页面中的超链接，经过N次跳转即可访问到大部分页面。那么简单的说，搜集信息的这个步骤就是利用网络爬虫通过自动化方式下载网页。

整理信息

如果不进行信息整理，那么每次查找就需要遍历所有页面，这显然和去所有站点查询一遍没什么区别，时间代价大的难以接受。因此我们需要对搜集来的信息建立目录，按照一定的规则对其进行编排，使其变得有序。这个过程叫做『创建索引』。

接受查询

用户向搜索引擎发出查询请求，搜索引擎分析请求，在创建的索引库中进行查询，之后返回给用户结果。在当前这个步骤中，我们要考虑的是搜索引擎的性能问题，因为可能有很多人同时进行查询，并且要保证用户的查询能快速回应，这不是一件简单的事情。

搜索引擎分类

搜索引擎按其工作方式主要可分为三种，分别是全文搜索引擎（Full Text Search Engine）、垂直搜索引擎（Vertical Search Engine）和元搜索引擎（Meta Search Engine）。

全文搜索

通常我们用到的例如Google、Bing、Yahoo、Baidu等都属于全文搜索，即通用搜索引擎。它们提取各个站点中的整个页面信息而创建索引，通过用户查询的关键字匹配相关记录，对结果进行排序并返回给用户，一般展示格式为网页标题和摘要信息，其查询的范围是所有网页。

垂直搜索

垂直搜索是针对某一行业专业搜索引擎，是搜索引擎的细分和延伸，是对网页库中的某类专门的信息进行一次集成，定向分字段抽取出需要的数据进行处理后再以某种形式返回给用户。这里的形式可能各不相同，结果也更加精准。解决了通用搜索中信息量大、查询不准确、深度不够等问题，但其信息范围局限于某一特定领域、人群、需求。常见的垂直搜索有学术论文搜索、旅游信息搜索、周边信息搜索等等。

元搜索

元搜索引擎在接受用户查询请求时，同时在其他多个引擎上进行搜索，并将结果返回给用户。按照作者理解，元搜索不是真正的搜索引擎，并且当前元搜索的并没有什么优势，几乎见不到了。

垂直搜索特点

前面简要介绍了搜索引擎的基本知识，这里该进入本文的重点--垂直搜索。

信息采集的特点

从采集方式看，通用搜索以被动方式为主，搜索引擎和被采集的网页没有约定的、标准的格式；站内搜索以主动方式为主，被采集的办公文档、CRM和ERP中的数据等都和企业搜索引擎有着约定好的采集接口和安全接口；垂直搜索则采用被动和主动想结合的方式，通过主动方式，有效采集网页中标引的元数据，整合上下游网页资源或者商业数据库，提供更加准确的搜索服务。通用搜索采用广度为先的策略，所以对采集深度要求不高，而垂直搜索和站内搜索需要挖掘出行业内所有相关的网页信息，所以采用深度为先的策略，同时由于行业内的一些有商业价值的信息采用动态发布的方式，所以垂直搜索对动态网页的采集优先级别较高。实际应用中，垂直搜索和站内搜索都需要集成和采集关系数据库中的结构化信息。

信息整理的特点

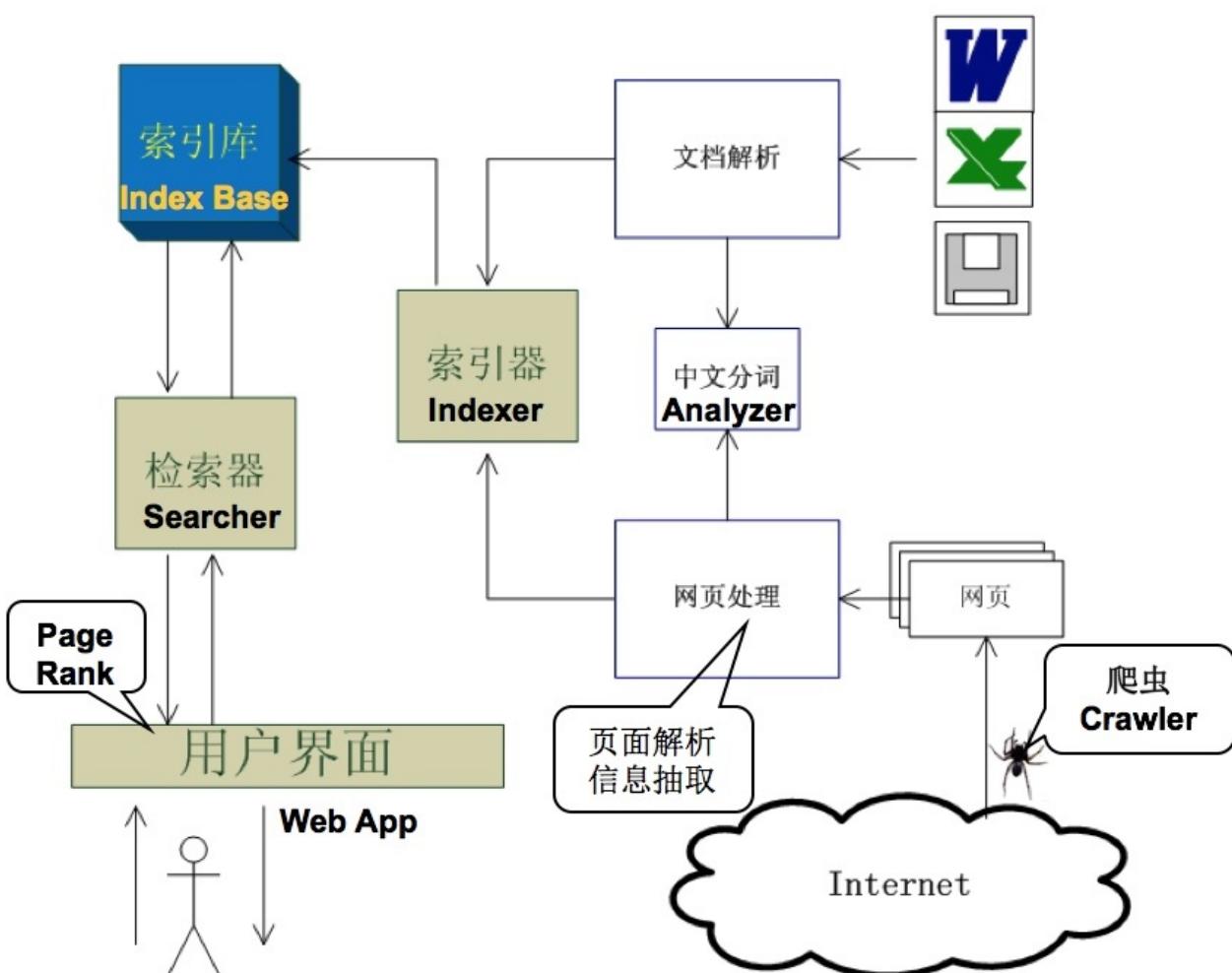
垂直搜索和普通的网页搜索的最大区别是对网页信息进行了结构化信息抽取加工，也就是将网页的非结构化数据抽取成特定的结构化信息数据，好比网页搜索是以网页为最小单位，而垂直搜索是以结构化数据为最小单位。垂直搜索的结构化信息提取和加工主要包括：网页元数据的提取和内容中结构化实体信息的提取。这些数据存储到数据库中，进行进一步的加工处理，如：去重、分类等，最后分词、索引再以搜索的方式满足用户的需求。目前，大部分垂直搜索的结构化信息提取都是依靠手工、半手工的方式来完成的，面对互联网的海量信息，很难保证信息的实时性和有效性，因此智能化成为垂直搜索的发展趋势。

查询结果的特点

从信息检索的结果来看，垂直搜索引擎不但能够对网页信息中的结构化信息进行检索，而且能够提供结构化和非结构化信息相结合的检索方式。从检索结果的排序方式看，垂直搜索的排序需求更加多样化。

如何构建垂直搜索

技术流程



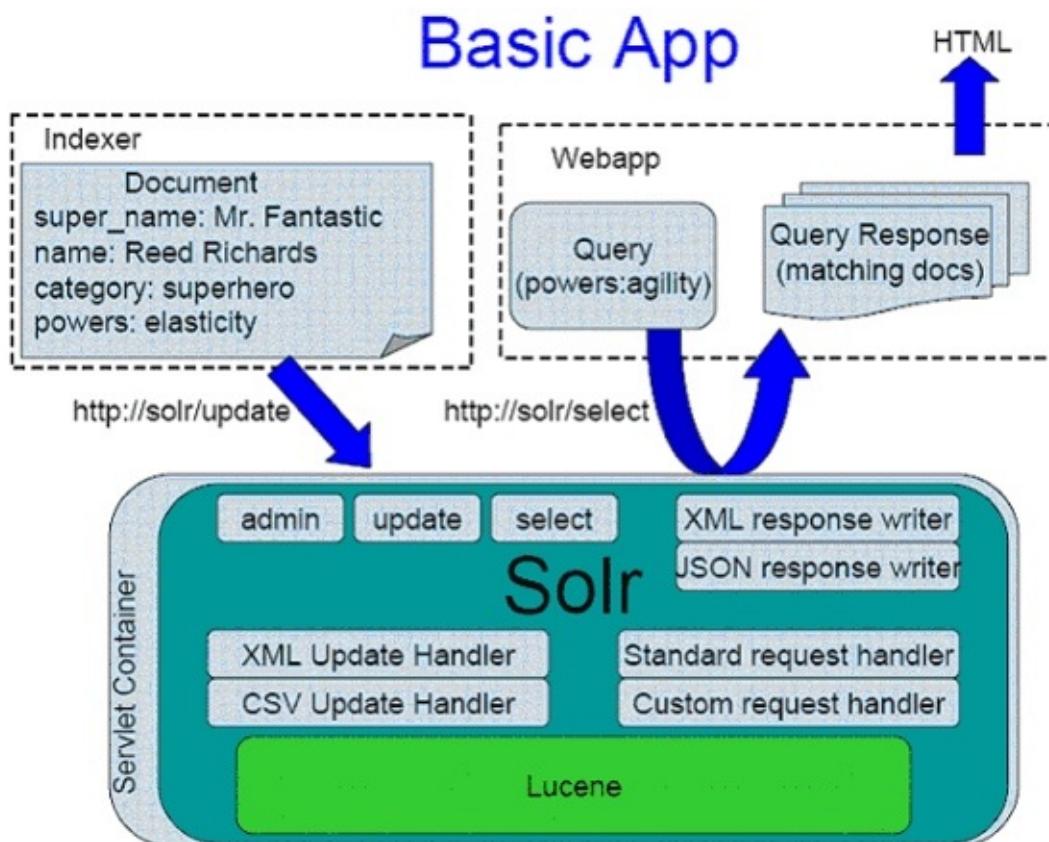
从上图可以看出，搜索引擎的基本技术流程，其中有三大核心组件

- 索引库(DB)是信息存储的地方，这里的信息已经变成倒排结构。
- 索引器(Indexer)是生成索引的模块，我们将处理过后的信息包装成文档交给索引器，索引器会在索引库中建立该文档的索引（也就是倒排结构）。
- 检索器（Searcher）是信息查询的模块

具体实现

搜索引擎的每一个模块都是作为一个单独的研究方向，我们不可能在短时间从无到有实现整个搜索引擎，幸好有开源社区的帮助，我们不用重复造轮子，将造好的轮子组装起来就好了。这里我们用到的开源软件是Apache的 Solr，它是基于Lucene的企业级搜索服务器，关于Solr和Lucene的介绍后面会讲到。

下图是我们利用Solr构建搜索引擎的基本构架，整个Solr可以看做一个完整的服务，Indexer模块来更新Solr的数据；Webapp从Solr获取数据，经过一些处理返回到浏览器，呈现给用户。



用到的开源项目

Lucene

Solr

Heritrix

webMagic

Zookeeper

AngularJS

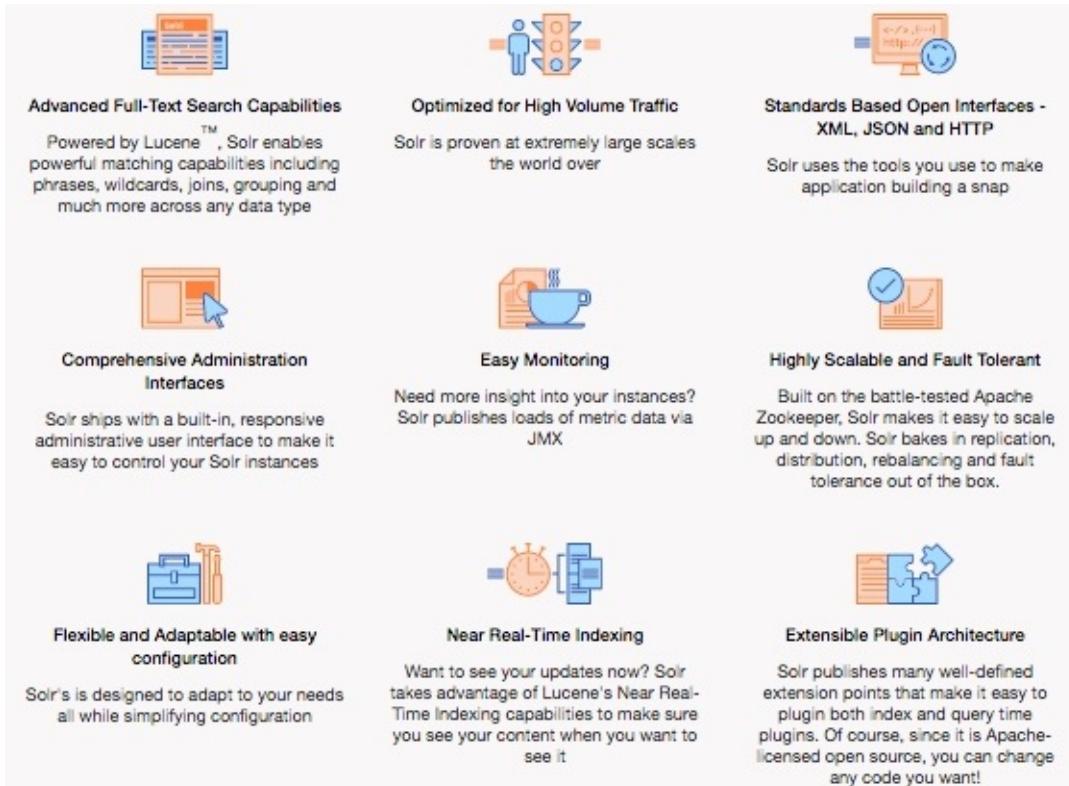
Ionic

Solr的身世

引用Solr官网的slogan,blazing-fast一词可见一斑。

Solr is the popular, blazing-fast, open source enterprise search platform built on Apache Lucene™.

再来看看它的特性



- Advanced Full-Text Search Capabilities (高级全文搜索能力)
- Optimized for High Volume Traffic (大数据性能优化)
- Standards Based Open Interfaces - XML, JSON and HTTP (标准的XML,JSON,HTTP接口)
- Comprehensive Administration Interfaces (综合管理界面)
- Easy Monitoring (易于监控)
- Highly Scalable and Fault Tolerant (高可扩展和容错力)
- Flexible and Adaptable with easy configuration (通过简单配置带来灵活性和适应性)
- Near Real-Time Indexing (近乎实时的索引)
- Extensible Plugin Architecture (可扩展的插件构架)

其实简单的说，Solr是一个基于Apache Lucene项目的开源企业级搜索平台，是用JAVA编写的、运行在Servlet容器中的一个独立的全文搜索服务器（换句话说就是个JAVA-WEB APP），并具有类似REST的HTTP/XML和JSON的API。

主要功能包括全文检索，高亮命中，分面搜索(faceted search)，近实时索引，动态集群，数据库集成，富文本索引，空间搜索；通过提供分布式索引，复制，负载均衡查询，自动故障转移和恢复，集中配置等功能实现高可用，可伸缩和可容错。

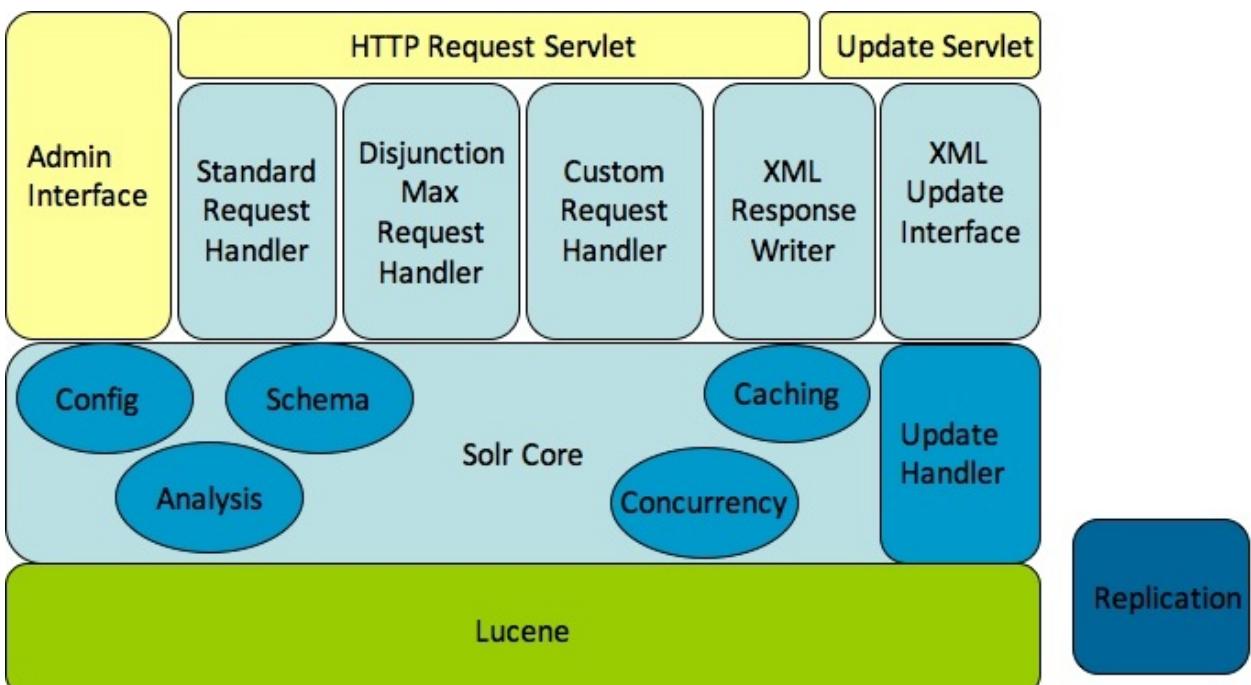
Solr和Lucene的关系

- Solr是Lucene的一个子项目，它在Lucene的基础上进行包装，成为一个企业级搜索服务框架。
- Solr与Lucene的主要区别体现在：
 - Solr更加贴近实际应用，是Lucene在面向企业搜索服务领域的扩展；
 - Solr的缓存等机制使全文检索获得性能上的提升；通过配置文件的开发使得Solr具有良好的扩展性；
 - Solr提供了用户友好的管理界面与查询结果界面。

简单讲：Solr使用Lucene并且扩展了它！

Solr的构架

其构架如下。



安装和配置

参考官方[quickstart](#)文档

下载Solr

去Lucene的[官网](#)下载就可以，会有三个文件可以下载

```
solr-x.x.x-src.tgz  
solr-x.x.x.tgz  
solr-x.x.x.zip
```

有src是源码，如果你不准备看源码、调试，不用管这个，另外两个是已经编译过的，下那个都行，只不过打包方式不同。这里直接下载zip后缀的。

环境要求

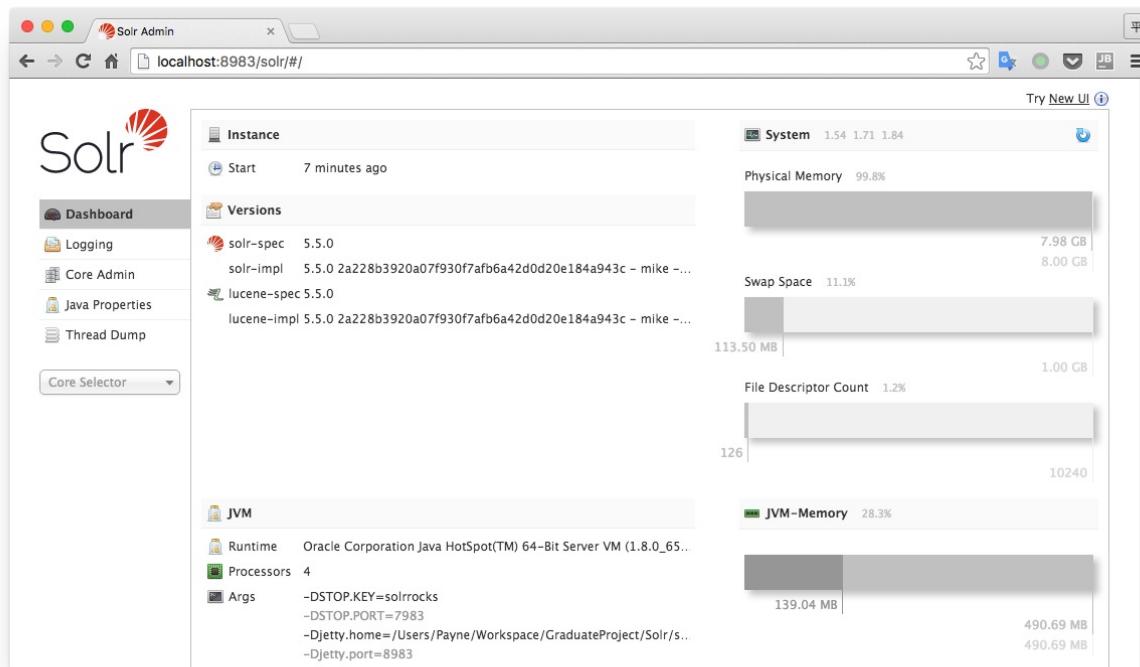
如果版本是5.x.x 或者 6.x.x用Java 8（写这篇文章最新版本是6.0.0）建议操作系统最好是 Unix系列，笔者这里用的是OS X系统，鉴于java环境的配置是开发最基本的技能，这里不再展开。

启动Solr

解压下载的文件，转到目录，输入一下命令即可启动solr服务器

```
bin/solr start
```

通过浏览器访问 <http://localhost:8983>即可看到Solr的管理界面。上面的这条命令是按照单机模式启动，还有cloud模式，顾名思义应该是solr集群了。关于cloud后面会单独来讲。



必要了解

目录结构

以我开发用的solr-5.5.0为例

```
.
├── CHANGES.txt
├── LICENSE.txt
├── LUCENE_CHANGES.txt
├── NOTICE.txt
├── README.txt
├── bin          # 存放的是solr为了方便使用所编写好的脚本
├── contrib      # 存放的大多是一些第三方提供的类库，使用这些类库能够极大扩展solr的功能
├── dist         # 存放的是solr自身所用到的一些核心类
├── docs         # 存放solr的文档，功能介绍，一些api的介绍
├── example      # 存放的是solr的例子
├── licenses     # 许可相关
└── server       # 是solr自带的jetty
```

主要概念

Solr安装目录和**SolrHome**目录

首先要知道solr的启动必须要在类似于tomcat一样的servlet容器中进行的，在servlet容器中启动之后的solr仅仅是一个webapp实例，能够使用url进行admin页面的访问，然而这个页面并没有任何的数据可以进行索引。同时在servlet容器中启动solr时，需要指定一个**solr.home.dir**，这个路径为这个solr实例的根路径。可以用同一个servlet容器的文件启动多个servlet容器（需要端口号不同），同时，启动多个servlet容器的过程中如果使用不同的**solr.home.dir**可以启动多个不同的solr实例。也就是说**solr.home.dir**是solr在servlet容器中启动的时候定义的。之后如果这个solr实例需要创建一个collection，则会在该solr的home.dir下创建一个存放该collection索引文件的文件夹，这个文件夹中需要有**solr.xml**文件夹。如果使用cloud模式启动的话，这个collection的配置文件会被上传到ZooKeeper中，在ZooKeeper的资源路径下会有个该collection名字命名的文件夹，然后相应的配置文件会存放在ZooKeeper的这个文件夹中（这里只是提一下could模式，不了解可以跳过这句）。如果不是使用cloud模式启动的，则在solr的该collection路径下会有**conf**文件夹，这个文件夹里必须有**solrconfig.xml**和**schema.xml**文件。

而**solr.installation.dir**文件路径是你下载好solr并解压缩的路径，前面讲过这个路径下的文件结构，如果感兴趣的话可以阅读**README.txt**文件里的内容。

SolrHome和**SolrCore**

SolrHome是一个目录，它是solr运行的主目录，它包括多个**SolrCore**目录，**SolrCore**目录中就solr实例的运行配置文件和数据文件。

SolrHome中可以包括多个**SolrCore**，每个**SolrCore**互相独立，而且可以单独对外提供搜索和索引服务。

也许你不熟悉Jetty，或者觉得它性能不行，你想在Tomcat上运行Solr，没问题，理论上只要是servlet容器都可以运行Solr。不过问题是自从solr5开始官方不再支持Tomcat的集成，所以可以有些配置问题需要自己来解决。于是我进行了一次尝试，solr4本来是比较容易的，Solr5就出现一些问题，由于对Tomcat了解也不是很深入，除运行的时候Solr管理界面有些小问题外，基本可以正常使用。

环境

OSX 10.11.4 JDK 1.8

安装Tomcat

由于各位的平台不同，请自行搜索安装。

Solr基础知识

[Solr基础知识及安装](#)

集成Solr

1. 指定 \$solr 为压缩包路径，\$TomcatDir 为tomcat目录。
2. 将 \$Solr/server/solr-webapp/ 下webapp文件夹，复制到 \$TomcatDir/webapps/ 目录下，并改成solr(或你喜欢的名字) .
3. 将 \$Solr/server/lib/ext 中的 jar 全部复制到 \$TomcatDir/webapps/solr/WEB-INF/lib 目录中.
4. 将 \$Solr/server/resources/log4j.properties 复制到 \$TomcatDir/webapps/solr/WEB-INF/classes 目录中 (如果没有classes则创建，放在bin中貌似也没影响) .
5. 将 \$Solr/server/solr 目录复制到你的 \$SolrHome 目录下，例如: ~/SolrHome .
6. 打开 \$TomcatDir/webapps/solr/WEB-INF 下的web.xml，找到如下配置内容 (初始状态下该内容是被注释掉的)

```

<env-entry>
    <env-entry-name>solr/home</env-entry-name>
    <env-entry-value>/put/your/solr/home/here</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
</env-entry>

```

将 `<env-entry-value>` 中的内容改成你的 \$SolrHome 路径，~/SolrHome.

7. 打开 `$TomcatDir/webapps/solr/WEB-INF` 下的 `web.xml` 修改项目欢迎页面

```

<welcome-file-list>
    <welcome-file>./index.html</welcome-file>
</welcome-file-list>

```

注意：一定要确保文件中有这一项，否则会出现404错误。

8. 如果要导入数据，还要添加 `$Solr/dist/` 中的 `solr-dataimporthandler-*.jar` 和 `solr-dataimporthandler-extras-*.jar` 到 `$TomcatDir/webapps/solr/WEB-INF/lib` 目录下。

9. 启动tomcat，在浏览器输入<http://localhost:8080/solr/admin.html> 即可出现Solr的管理界面。

添加SolrCore

- 在 `solr_home` 目录下创建 `core_1`（可自定义），在 `core_1` 目录下创建 `data` 目录，并将 `solr_home/configsets/basic_configs/` 目录下的 `conf` 目录复制到 `core_1` 下；通过控制台添加 `core`，并重新启动 Tomcat，就会看到新建的 `core_1` 了。

或者：

- [http://localhost:8080/solr/admin/cores?
action=CREATE&name=universal&instanceDir=universal&config=solrconfig.xml&schemat=a=schema.xml&dataDir=data](http://localhost:8080/solr/admin/cores?action=CREATE&name=universal&instanceDir=universal&config=solrconfig.xml&schemat=a=schema.xml&dataDir=data)

2.搜集信息部分

本章将介绍搜索引擎搜集信息部分，主要是通过爬虫获得原始数据，当然目前主流的爬虫框架都会带有解析功能，因此，有些解析工作也在此阶段完成。

Heritrix 是一个由 java 开发的、开源的网络爬虫，用户可以使用它来从网上抓取想要的资源。其最出色之处在于它良好的可扩展性，方便用户实现自己的抓取逻辑。最重要的是对于一般的抓取，你是不用碰任何代码的，只要写好配置文件就可以了，简直就是某些人的福音。不过对于一个真正的程序员，代码可是他们的整个生命啊。

原理及特点

深度遍历网站的资源，将这些资源抓取到本地，分析网站每一个有效的URL，并提交Http请求，从而获得相应结果，生成本地文件及相应的日志信息等。

特点：Heritrix 是个 "Archival crawler" -- 用来获取完整的、精确的、站点内容的深度复制。包括获取图像以及其他非文本内容。抓取并存储相关的内容。对内容来者不拒，不对页面进行内容上的修改（可以通过配置进行过滤）。重新爬行对相同的URL不针对先前的进行替换。爬虫通过Web用户界面启动、监控、调整，允许弹性的定义要获取的URL。

优点：Heritrix的爬虫定制参数多

缺点：单实例的爬虫，之间不能进行合作。在有限的机器资源的情况下，却要复杂的操作。只有官方支持，仅仅在Linux上进行了测试。每个爬虫是单独进行工作的，没有对更新进行修订。在硬件和系统失败时，恢复能力很差。很少的时间用来优化性能。

安装及使用

下载及文档: <https://webarchive.jira.com/wiki/display/Heritrix/>

写这篇文章时最新版本为3.2.0，安装方法非常简单，我参考的是官方的文档[Heritrix 3.0 and 3.1 User Guide](#)，因为是基于JAVA的，只要有正确的java环境就很容易运行起来。

系统要求：

- 系统：官方只支持Linux，别的系统理论上也是可以的，我用的是OSX也是OK的；
- **JDK**：官方要求JDK1.6，但我在Linux上用过Openjdk1.7也是OK的，甚至MAC上JDK1.8也没问题，不过推荐还是按照官方要求。怎么安装多个版本的JDK？这不用再说了吧。
- 内存什么的要1G吧，Java Heap默认256M，对于爬虫应该不够，可能会出现out-of-memory异常，可以用JAVA_OPTS来配置jvm的内存。

配置

解压就不用说了，完了之后直接配置使用就行，当然前提JDK环境已经有了。

1. 配置环境变量JAVA_HOME,配过的请忽略

```
export JAVA_HOME=/usr/local/java/jre
```

2. 配置环境变量HERITRIX_HOME，就是设置Heritrix的主目录

```
export HERITRIX_HOME=/PATH/TO/HERITRIX(替换成解压出来的路径)
```

3. 为启动文件设置执行权限

```
chmod u+x $HERITRIX_HOME/bin/heritrix
```

4. 设置JVM的内存占用,这里是1G，也可以更高

```
export JAVA_OPTS=-Xmx1024M
```

运行

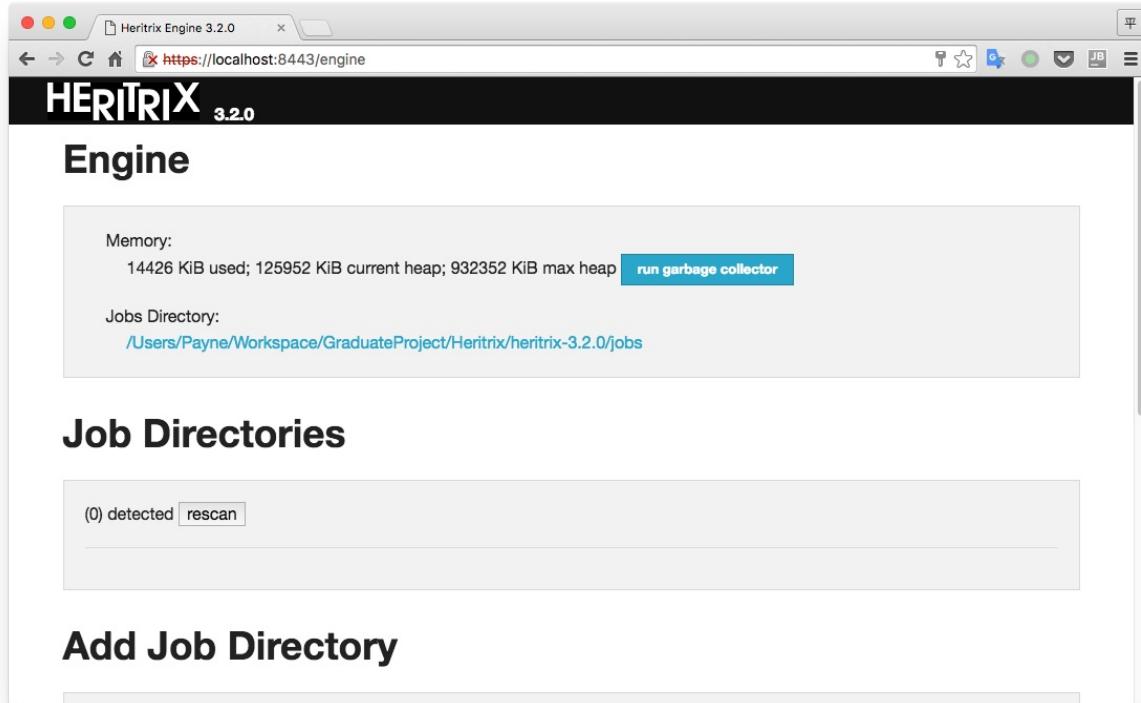
```
$HERITRIX_HOME/bin/heritrix -a admin:admin
```

上面的这条命令就启动了Heritrix，并设置登陆web管理页面的用户名及密码都为admin。当然很有别的参数可以用，查看帮助就好了。

```
$HERITRIX_HOME/bin/heritrix --help
```

登陆管理界面：<https://localhost:8443>

一定注意是**https**链接，但因为没有证书，所以会出现安全警告，忽略继续访问就可以的。



创建抓取任务

在『Add Job Directory』这一项中可以创建新任务或者添加已存在的任务。如下图所示我添加了一个叫做tripsearch的任务。

The screenshot shows the Heritrix Engine 3.2.0 interface. At the top, the title bar says "Heritrix Engine 3.2.0". The address bar shows the URL "https://localhost:8443/engine". The main content area has a heading "Job Directories". Below it, a message says "(1) detected rescan". Underneath, there is a section for "tripsearch" with "0 launches" and the path "/Users/Payne/Workspace/GraduateProject/Heritrix/heritrix-3.2.0/jobs/tripsearch/crawler-beans.cxml". Below this is a form titled "Add Job Directory" with instructions to "Create new job directory with recommended starting configuration". It includes fields for "Path:" (containing "/Users/Payne/Workspace/GraduateProject/Heritrix/heritrix-3.2.0/jobs/") and a "create" button. There is also a field for "Specify a path to a pre-existing job directory" with a "Path:" input field and an "add" button. A note at the bottom explains how to compose or copy an existing job directory.

点击这个tripsearch就可以进入这个任务的详情页面，如下图所示：

The screenshot shows the Heritrix interface for the "tripsearch" job. The top navigation bar includes tabs for "Engine", "Job Dir", "Configuration", "Copy Job", "Scripting Console", and "Browse Beans (disabled)". The main content area shows the job name "Job tripsearch" with "(0 launches)" status. Below the job name is a toolbar with buttons: "build" (highlighted in blue), "launch", "pause", "unpause", "checkpoint", "terminate", and "teardown". Underneath the toolbar is a section titled "Job Log" with a "more" link. The status is listed as "Job is Unbuilt". Below this is a section titled "Configuration-referenced Paths" with the instruction "build the job to discover referenced paths".

看到上面那些菜单项，最重要的是Configuration，通过一个名为crawler-beans.cxml的配置文件控制整个抓取过程。另外还有些控制按钮，这里也来说下。

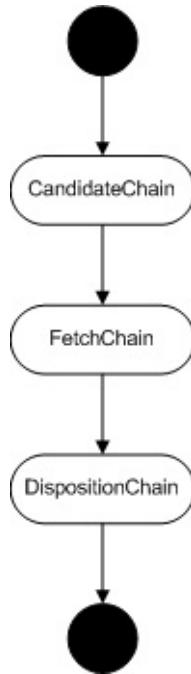
- build：构建任务实例
- launch：启动抓取

- pause : 暂停抓取
- unpause : 从暂停中恢复
- checkpoint : 建立检查点 (这个时候会将之前抓到的数据封包备份, 如同数据库的checkpoint)
- terminate : 终止抓取任务
- teardown : 删除任务实例

正常的启动流程是：改好配置文件 -> build -> launch。

Heritrix处理流程

Heritrix内部按模块划分成一个个处理器，一个个处理器有序排列，构成处理器链，而处理器链也有序排列。大致可划分为3个处理器链，每个处理器链中又有多个处理器模块。每个URL任务从头走到尾巴，经过各个处理器的处理。



`Candidate Chain` 主要负责筛选要抓取的URL，然后把它放入抓取队列中。`Fetch Chain` 主要负责抓取网页内容，提取和解析出内容中的URL。`Disposition Chain` 主要负责将抓取的内容存储下来，把新的解析出的URL再发送回 `Candidate Chain`。

参数解释及配置

可以说 `crawler-beans.xml` 可以主导整个Heritrix的抓取，采用spring来管理.里面的配置都是一个个bean,通过修改配置文件 `crawler-beans.xml` 即可完成几乎所有的需求。

我是要抓取携程各个景点页面，对不多的几个地方进行修改即可实现目标了。

1. 修改任务元数据

```

<bean id="simpleOverrides" class="org.springframework.beans.factory.config.PropertyOverrideConfigurer">
    <property name="properties">
        <value>
# This Properties map is specified in the Java 'property list' text format
# http://java.sun.com/javase/6/docs/api/java/util/Properties.html#load%28java.io.Reader%29

metadata.operatorContactUrl=https://stu-ali.xvping.cn:8443
metadata.jobName=XP Crawl
metadata.description=XP crawl for Searching engine

###...more?...##
        </value>
    </property>
</bean>

```

通过文章后面对录中对各个Bean的介绍，simpleOverrides的作用是设置基本的一些任务元数据，我修改的这3个分别是操作人员的联系URL、任务的名字和任务的描述。这些信息在请求页面的时候会带上，告诉对方自己的身份描述信息。身份信息在robots.txt协议中有一定作用，网站可以通过设置该协议来拒绝特定的爬虫。协议具体细节请自行查询。包括是否遵守爬虫的robots.txt协议也可以在metadata里配置。

2. 设置种子站点 种子站点：爬虫从这些页面开始解析出来URL并加入待爬列表中。

```

<bean id="longerOverrides" class="org.springframework.beans.factory.config.PropertyOverrideConfigurer">
    <property name="properties">
        <props>
            <prop key="seeds.textSource.value">

# URLs HERE
http://you.ctrip.com/sitemap/spotdis/c0
http://you.ctrip.com/sitemap/spots.html

            </prop>
        </props>
    </property>
</bean>

```

3. 定制爬取范围

Heritrix提供了一种URL匹配的规则模式-- SURT。

```

<bean id="acceptSurts" class="org.archive.modules.deciderules.surt.SurtPrefixedDecideRule">
    <!-- <property name="decision" value="ACCEPT"/> -->
    <!-- <property name="seedsAsSurtPrefixes" value="true" /> -->
    <!-- <property name="alsoCheckVia" value="false" /> -->
    <!-- <property name="surtsSourceFile" value="" /> -->
    <!-- <property name="surtsDumpFile" value="${launchId}/surts.dump" /> -->
    <property name="surtsSource">
        <bean class="org.archive.spring.ConfigString">
            <property name="value">
                <value>
                    # example.com
                    # http://www.example.edu/path1/
                    # +http://(org,example,
                    +http://you.ctrip.com/sight/
                    +http://you.ctrip.com/countrysightlist/
                    -http://(com,qq,
                    -http://(com,gtimg,
                    -http://(com,c-ctrip,
                    -http://(com,ctrip,m,
                    -http://(cn,sinainmg,
                </value>
            </property>
        </bean>
    </property>
</bean>

```

通过官方文档中的[SURT Rules](#)，+ 表示符合后面表达式的接受，- 表示符合后面表达式的拒绝。表达式 `http://(com,qq,` 表示`qq.com`下的所有子域名都匹配；`http://(com,qq,www,)` 只匹配`www.qq.com`,不包含子域名；`http://(com,ctrip,you,)/sight/` 表示匹配`you.ctrip.com`子域名，并且路径为`sight`下的页面，注意最后的斜杠(slash)不能少。

4. 配置ACCEPT和REJECT规则

每一个规则都由一个Bean来配置。一起组成如下的规则序列。

```

<!-- SCOPE: rules for which discovered URIs to crawl; order is very
     important because last decision returned other than 'NONE' wins. -->
<bean id="scope" class="org.archive.modules.deciderules.DecideRuleSequence">
    <!-- <property name="logToFile" value="false" /> -->
    <property name="rules">
        <list>
            <!-- Begin by REJECTing all... -->
            <bean class="org.archive.modules.deciderules.RejectDecideRule" />
            <!-- ...then ACCEPT those within configured/seed-implied SURT prefixes... -->
            <ref bean="acceptSurts" />
            <!-- ...but REJECT those more than a configured link-hop-count from start... -->
            <bean class="org.archive.modules.deciderules.TooManyHopsDecideRule">
                <!-- <property name="maxHops" value="20" /> -->
            </bean>
        </list>
    </property>
</bean>

```

```

<!-- ...but ACCEPT those more than a configured link-hop-count from start... -->
<bean class="org.archive.modules.deciderules.TransclusionDecideRule">
    <!-- <property name="maxTransHops" value="2" /> -->
    <!-- <property name="maxSpeculativeHops" value="1" /> -->
</bean>
<!-- ...but REJECT those from a configurable (initially empty) set of REJECT SURTs
... -->
<bean class="org.archive.modules.deciderules.surt.SurtPrefixedDecideRule">
    <property name="decision" value="REJECT"/>
    <property name="seedsAsSurtPrefixes" value="false"/>
    <property name="surtsDumpFile" value="${launchId}/negative-surts.dump" />
    <!-- <property name="surtsSource">
        <bean class="org.archive.spring.ConfigFile">
            <property name="path" value="negative-surts.txt" />
        </bean>
    </property> -->
</bean>
<!-- ...and REJECT those from a configurable (initially empty) set of URI regexes.
... -->
<bean class="org.archive.modules.deciderules.MatchesListRegexDecideRule">
    <property name="decision" value="REJECT"/>
    <property name="listLogicalOr" value="true" />
    <property name="regexList">
        <list>
            <value>.*\jpg</value>
            <value>.*\png</value>
            <value>.*\gif</value>
            <value>.*\css</value>
        </list>
    </property>
</bean>
<!-- ...and REJECT those with suspicious repeating path-segments... -->
<bean class="org.archive.modules.deciderules.PathologicalPathDecideRule">
    <!-- <property name="maxRepetitions" value="2" /> -->
</bean>
<!-- ...and REJECT those with more than threshold number of path-segments... -->
<bean class="org.archive.modules.deciderules.TooManyPathSegmentsDecideRule">
    <!-- <property name="maxPathDepth" value="20" /> -->
</bean>
<!-- ...but always ACCEPT those marked as prerequisite for another URI... -->
<bean class="org.archive.modules.deciderules.PrerequisiteAcceptDecideRule">
</bean>
<!-- ...but always REJECT those with unsupported URI schemes -->
<bean class="org.archive.modules.deciderules.SchemeNotInSetDecideRule">
</bean>
</list>
</property>
</bean>

```

一个URL要按顺序由上到下经过各条规则，最终来决定是否接受这个URL，所以规则顺序就非常重要。在经过一条规则的时候，根据规则是拒绝型还是接受型来做不同的处理。如果是拒绝型，则在接受队列中找到符合规则的URL，取出放入拒绝队列中。反之，如果是接受型，则在拒绝队列中找到符合规则的URL，放入接受队列。最终，经过所有规则后，只保留接受队列里的URL。

第一条规则先拒绝所有的URL，然后第二条就是之前配置的SURT规则，接受指定URL路径下的页面。下面的规则就不一一列举了。为了避免已经拒绝了的非所需域名下的链接在之后的接受规则中又被纳入接受队列，注释掉了之后的关于接受的规则。（当然可以把域名过滤的这个规则放在最后一条，最后再过滤掉不符合所需域名的URL）

5. 存储相关

最后是关于写入压缩后的网页内容到硬盘的相关配置。Disposition中的Writer处理器就是处理内容的写入，默认使用的是WarcWriter，warc是Web页面内容的压缩后的格式。Heritrix会把每次请求获得的相关数据和每次请求的元数据，一条条写入warc文件，并且再用gz做压缩。相关内容的配置如下。

```
<bean id="warcWriter" class="org.archive.modules.writer.WARCWriterProcessor">
```

在上面的Bean可以配置压缩文件的文件名，最大容量，存储路径等等。还有执行的规则，可以拒绝掉不符合规则的内容的存储。warc文件最大容量可以不要设置太大，便于之后我们对数据的处理。一般用默认就行，这里不细说。

附录(各个**bean**的介绍)：

1. bean id=simpleOverrides
class=org.springframework.beans.factory.config.PropertyOverrideConfigurer 字面上的意思为简单的覆盖,的确这里只是简单的覆盖.设置最基本的信息.如抓取任务名字(metadata.jobName),操作URL(metadata.operatorContactUrl),描述信息(metadata.description)
2. bean id=metadata class=org.archive.modules.CrawlMetadata 如同simpleOverrides
3. bean id=seeds class=org.archive.modules.seeds.TextSeedModule 种子配置,可以从文件中读取种子,也可以直接设置种子
4. bean id=scope class=org.archive.modules.deciderules.DecideRuleSequence URL 规则控制,可以决定哪些URL要抓取,哪些URL拒绝,URL抓取深度等
5. bean id=candidateScoper class=org.archive.crawler.prefetch.CandidateScoper URL 范围控制,通过该范围的URL Heritrix方可接受,成为CrawlURI
6. bean id=preparer class=org.archive.crawler.prefetch.FrontierPreparer url预处理,如设置URL的抓取深度,队列,成本控制等
7. bean id=candidateProcessors class=org.archive.modules.CandidateChain 处理器,引用candidateScoper去控制URL是否可以成为CrawlURI,preparer去设置深度,队列,成本控制等
8. bean id=preselector class=org.archive.crawler.prefetch.Preselector 预先选择器,这里会过滤掉一部分URL.如blockByRegex为拒绝正则,allowByRegex为允许正则
9. bean id=preconditions class=org.archive.crawler.prefetch.PreconditionEnforcer 先决条件设置,如设置IP有效期,爬虫协议文件robots.txt有效期
10. bean id=fetchDns class=org.archive.modules.fetcher.FetchDNS 解析DNS,获得IP
11. bean id=fetchHttp class=org.archive.modules.fetcher.FetchHTTP 核心模块,获取URL内容,设置状态
12. bean id=extractorHttp class=org.archive.modules.extractor.ExtractorHTTP 核心模块,抽取URL,抽取出新的URL再次运行,如此爬虫才可以一直爬下去
13. bean id=extractorHtml class=org.archive.modules.extractor.ExtractorHTML 抽取HTML,包含JSP,ASP等,这里也会抽取JS,CSS等
14. bean id=extractorCss class=org.archive.modules.extractor.ExtractorCSS 抽取CSS,无需单独配置,ExtractorHTML会调用
15. bean id=extractorJs class=org.archive.modules.extractor.ExtractorJS 抽取JS,无需单独配置,ExtractorHTML会调用

参考资料

1. 搜索引擎搭建——Heritrix
2. 配置文件crawler-beans.cxml介绍
3. guoyunsky写的搜索引擎-爬虫-Heritrix系列
4. Heritrix 3.0 and 3.1 User Guide

Java 程序在解析 HTML 文档时，最常用的是 `htmlparser` 这个开源项目。但现在你有更好的选择，那就是 `Jsoup`。

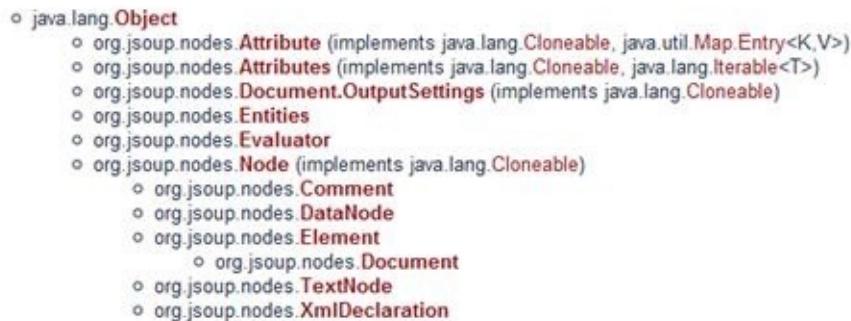
`jsoup` 是一款 Java 的 HTML 解析器，可直接解析某个 URL 地址、HTML 文本内容。它提供了一套非常省力的 API，可通过 DOM，CSS 以及类似于 `jQuery` 的操作方法来取出和操作数据。

`jsoup` 的主要功能如下：

1. 从一个 URL，文件或字符串中解析 HTML；
2. 使用 DOM 或 CSS 选择器来查找、取出数据；
3. 可操作 HTML 元素、属性、文本；`jsoup` 是基于 MIT 协议发布的，可放心使用于商业项目。

`jsoup` 的主要类层次结构如下图所示：

Class Hierarchy



Jsoup的基本用法

- [jsoup官网](#)

文件加载示例

`jsoup` 可以从包括字符串、URL 地址以及本地文件来加载 HTML 文档，并生成 `Document` 对象实例。

注意：本文中的代码仅仅是如何使用 `jsoup` 的实例片段，只是让读者了解如何使用，并不能直接运行

示例代码：

```

// 直接从字符串中输入 HTML 文档
String html = "<html><head><title> 开源中国社区 </title></head>" +
    "<body><p> 这里是 jsoup 项目的相关文章 </p></body></html>";
Document doc = Jsoup.parse(html);

// 从 URL 直接加载 HTML 文档
Document doc = Jsoup.connect("http://www.oschina.net/").get();
String title = doc.title();

Document doc = Jsoup.connect("http://www.oschina.net/")
    .data("query", "Java") // 请求参数
    .userAgent("I'm jsoup") // 设置 User-Agent
    .cookie("auth", "token") // 设置 cookie
    .timeout(3000) // 设置连接超时时间
    .post(); // 使用 POST 方法访问 URL

// 从文件中加载 HTML 文档
File input = new File("D:/test.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://www.oschina.net/");

```

请大家注意最后一种 HTML 文档输入方式中的 `parse` 的第三个参数，为什么需要在这里指定一个网址呢（虽然可以不指定，如第一种方法）？因为 HTML 文档中会有很多例如链接、图片以及所引用的外部脚本、css 文件等，而第三个名为 `baseURL` 的参数的意思就是当 HTML 文档使用相对路径方式引用外部文件时，jsoup 会自动为这些 URL 加上一个前缀，也就是这个 `baseURL`。例如 [开源软件](#) 会被转换成 [开源软件](http://www.oschina.net/)。

解析文件示例

jsoup 解析文件时除了提供传统的DOM方式的元素解析，最重要的是还提供了选择器来进行 DOM 元素的定位，其选择器语法和jQuery 的是一样的，这对于做过 web 开发的来说简直是福音，几乎不需要任何成本就可以用 java 愉快地解析 html 了呢。

传统的DOM方式示例代码：

```

File input = new File("D:/test.html");
Document doc = Jsoup.parse(input, "UTF-8", "http://www.oschina.net/");

Element content = doc.getElementById("content");
Elements links = content.getElementsByTag("a");
for (Element link : links) {
    String linkHref = link.attr("href");
    String linkText = link.text();
}

```

你可能会觉得 jsoup 的方法似曾相识，没错，像 `getElementById` 和 `getElementsByTag` 方法跟 JavaScript 的方法名称是一样的，功能也完全一致。你可以根据节点名称或者是 HTML 元素的 `id` 来获取对应的元素或者元素列表。

选择器示例代码：

```
File input = new File("D:\test.html");
Document doc = Jsoup.parse(input,"UTF-8","http://www.oschina.net/");

Elements links = doc.select("a[href]"); // 具有 href 属性的链接
Elements pngs = doc.select("img[src$=.png]");// 所有引用 png 图片的元素

Element masthead = doc.select("div.masthead").first();
// 找出定义了 class=masthead 的元素

Elements resultLinks = doc.select("h3.r > a"); // direct a after h3
```

可以看到，它的选择器和jQuery确实是一样的，另外它的选择器还支持表达式功能，我们将在最后一节介绍这个超强的选择器。

修改数据示例

在解析文档的同时，我们可能会需要对文档中的某些元素进行修改，例如我们可以为文档中的所有图片增加可点击链接、修改链接地址或者是修改文本等。

示例代码：

```
doc.select("div.comments a").attr("rel", "nofollow");
// 为所有链接增加 rel=nofollow 属性
doc.select("div.comments a").addClass("mylinkclass");
// 为所有链接增加 class=mylinkclass 属性
doc.select("img").removeAttr("onclick"); // 删除所有图片的 onclick 属性
doc.select("input[type=text]").val(""); // 清空所有文本输入框中的文本
```

首先利用 jsoup 的选择器找出元素，然后就可以通过以上的方法来进行修改，除了无法修改标签名外（可以删除后再插入新的元素），包括元素的属性和文本都可以修改。修改完直接调用 `Element(s)` 的 `html()` 方法就可以获取修改完的 HTML 文档。

HTML 代码过滤

有时候有的页面中会有一些恶意脚本，可能会破坏整个页面的结构，更严重的是获取一些机要信息，例如 XSS 跨站点攻击之类的。jsoup对这方面支持非常好。

示例代码：

```
String unsafe = "<p><a href='http://www.oschina.net/' onclick='stealCookies()'>开源中国  
社区</a></p>";  
String safe = Jsoup.clean(unsafe,Whitelist.basic());  
// 输出：  
// <p><a href="http://www.oschina.net/" rel="nofollow"> 开源中国社区 </a></p>
```

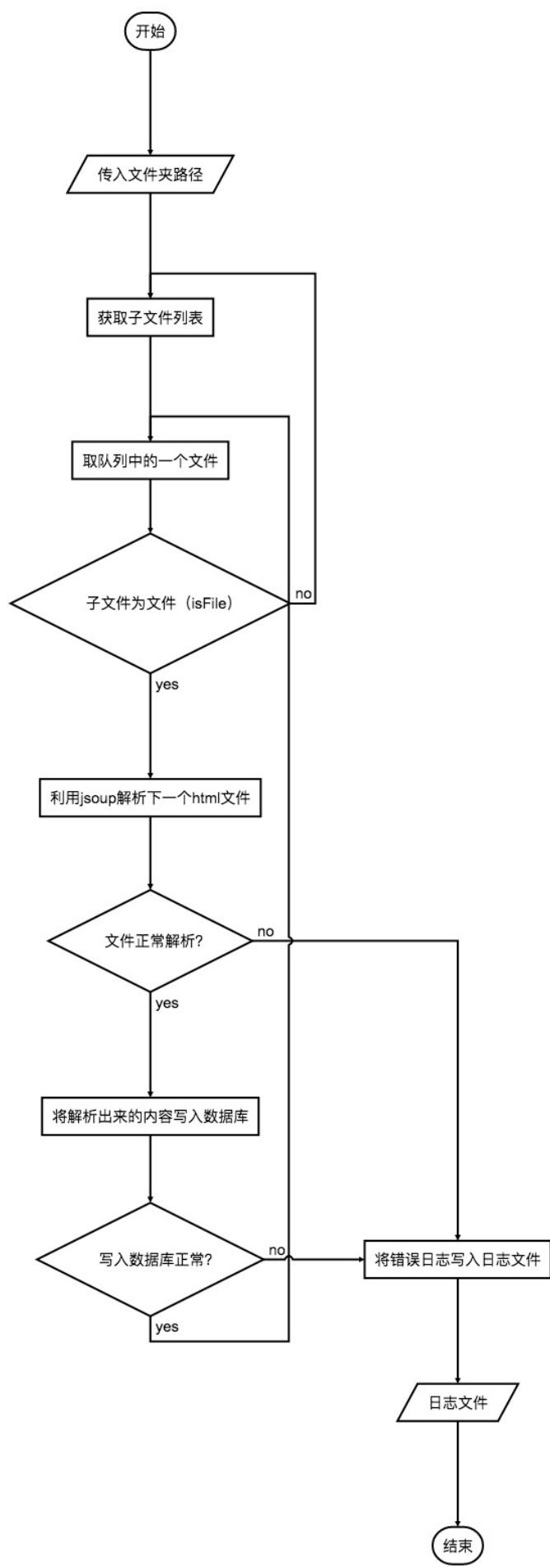
jsoup 使用一个 Whitelist 类用来对 HTML 文档进行过滤，该类提供几个常用方法：

方法名	简介
none()	只允许包含文本信息
basic()	允许的标签包括：a, b, blockquote, br, cite, code, dd, dl, dt, em, i, li, ol, p, pre, q, small, strike, strong, sub, sup, u, ul, 以及合适的属性
simpleText()	只允许 b, em, i, strong, u 这些标签
basicWithImages()	在 basic() 的基础上增加了图片
relaxed()	这个过滤器允许的标签最多，包括：a, b, blockquote, br, caption, cite, code, col, colgroup, dd, dl, dt, em, h1, h2, h3, h4, h5, h6, i, img, li, ol, p, pre, q, small, strike, strong, sub, sup, table, tbody, td, tfoot, th, thead, tr, u, ul

如果这五个过滤器都无法满足你的要求呢，例如你允许用户插入 flash 动画，没关系，Whitelist 提供扩展功能，例如 whitelist.addTags("embed","object","param","span","div"); 也可调用 addAttributes 为某些元素增加属性。

解析抓取下来的网页

我项目中的解析模块 [ExtractHtml-Github](#)。流程图如下，整个过程比较简单，有兴趣的可以看代码。



Jsoup的选择器

下面详细介绍选择器的使用方法，如果你用过jQuery的选择器，相信下面的这些用法你一定不会陌生。

1. 基本用法：

tagname	使用标签名来定位，例如 <code>a</code>
<code>ns:tag</code>	使用命名空间的标签定位，例如 <code>fb:name</code> 来查找 <code><fb:name></code> 元素
<code>#id</code>	使用元素 <code>id</code> 定位，例如 <code>#logo</code>
<code>.class</code>	使用元素的 <code>class</code> 属性定位，例如 <code>.head</code>
<code>[attribute]</code>	使用元素的属性进行定位，例如 <code>[href]</code> 表示检索具有 <code>href</code> 属性的所有元素
<code>[^attr]</code>	使用元素的属性名前缀进行定位，例如 <code>[^data-]</code> 用来查找 HTML5 的 <code>dataset</code> 属性
<code>[attr=value]</code>	使用属性值进行定位，例如 <code>[width=500]</code> 定位所有 <code>width</code> 属性值为 500 的元素
<code>[attr^=value], [attr\$=value], [attr*=value]</code>	这三个语法分别代表，属性以 <code>value</code> 开头、结尾以及包含
<code>[attr~=regex]</code>	使用正则表达式进行属性值的过滤，例如 ①（这里写正则显示异常，加在后面了）
<code>*</code>	定位所有元素

①: `img[src~=(?i)\.(png|jpe?g)]`

2. 组合用法：

用法	意义
<code>el#id</code>	定位 <code>id</code> 值某个元素，例如 <code>a#logo -> </code>
<code>el.class</code>	定位 <code>class</code> 为指定值的元素，例如 <code>div.head -> <div class=head>xxxx</div></code>
<code>el[attr]</code>	定位所有定义了某属性的元素，例如 <code>a[href]</code>
以上三个任意组合	例如 <code>a[href]#logo</code> 、 <code>a[name].outerlink</code>
<code>ancestor child , parent ></code> <code>child , siblingA + siblingB , siblingA ~</code> <code>siblingX , el, el, el</code>	这五种都是元素之间组合关系的选择器语法，其中包括父子关系、合并关系和层次关系。

除了一些基本的语法以及这些语法进行组合外，jsoup 还支持使用表达式进行元素过滤选择。下面是 jsoup 支持的所有表达式一览表：

3. 进行元素过滤选择：

表达式	意义
:lt(n)	例如 <code>td:lt(3)</code> 表示 小于三列
:gt(n)	<code>div p:gt(2)</code> 表示 div 中包含 2 个以上的 p
:eq(n)	<code>form input:eq(1)</code> 表示只包含一个 input 的表单
:has(selector)	<code>div:has(p)</code> 表示包含了 p 元素的 div
:not(selector)	<code>div:not(.logo)</code> 表示不包含 <code>class=logo</code> 元素的所有 div 列表
:contains(text)	包含某文本的元素，不区分大小写，例如 <code>p:contains(oschina)</code>
:containsOwn(text)	文本信息完全等于指定条件的过滤
:matches(regex)	使用正则表达式进行文本过滤： <code>div:matches((?i)login)</code>
:matchesOwn(regex)	使用正则表达式找到自身的文本

参考资料

1. jsoup 官方网站：<http://jsoup.org>。
2. 使用jsoup对HTML文档进行解析和操作-
IBM:<https://www.ibm.com/developerworks/cn/java/j-lo-jsouphml/>

有人可能会有疑问，Heritrix用的好好的，干嘛还要换别的呢？Heritrix固然很好，成熟、稳定、有管理界面、监控、多线程、开箱即用。但这真的适合我们这种垂直爬虫吗？我觉得未必，用Heritrix，你可以通过配置文件来确定接受规则和URL的发现规则，然后直接构建运行就好了。然而对于我来说，它的配置文件中的那些选项说的并不是那么清楚，相关文档也都是很简单的，想用一些复杂的规则都不知道这样对不对，至少要等到好久抓出来东西了，发现多抓了或者少抓了，你才发现配置文件写错了。并且配置文件中的项目也是老多的，这么多东西把人搞的头晕目眩。

好吧，总的来讲，如果你是爬虫老手，对Heritrix的架构了解比较清晰的，Heritrix是个很好的选择。但是对于一个新手，我觉得webmagic更加灵活高效。就拿我的体验来讲，当初用Heritrix抓了3天才抓了七千多数据（可能是我配置文件写的不好），而用webmagic一个白天就抓了一万三的数据。

webmagic介绍

认识webmagic是在知乎上有人推荐，我确定用它使因为该作者比较详细的中文文档（谁让咱英文不太好呢），这是该作者的一个业余项目，但他的代码是非常值得参考的。

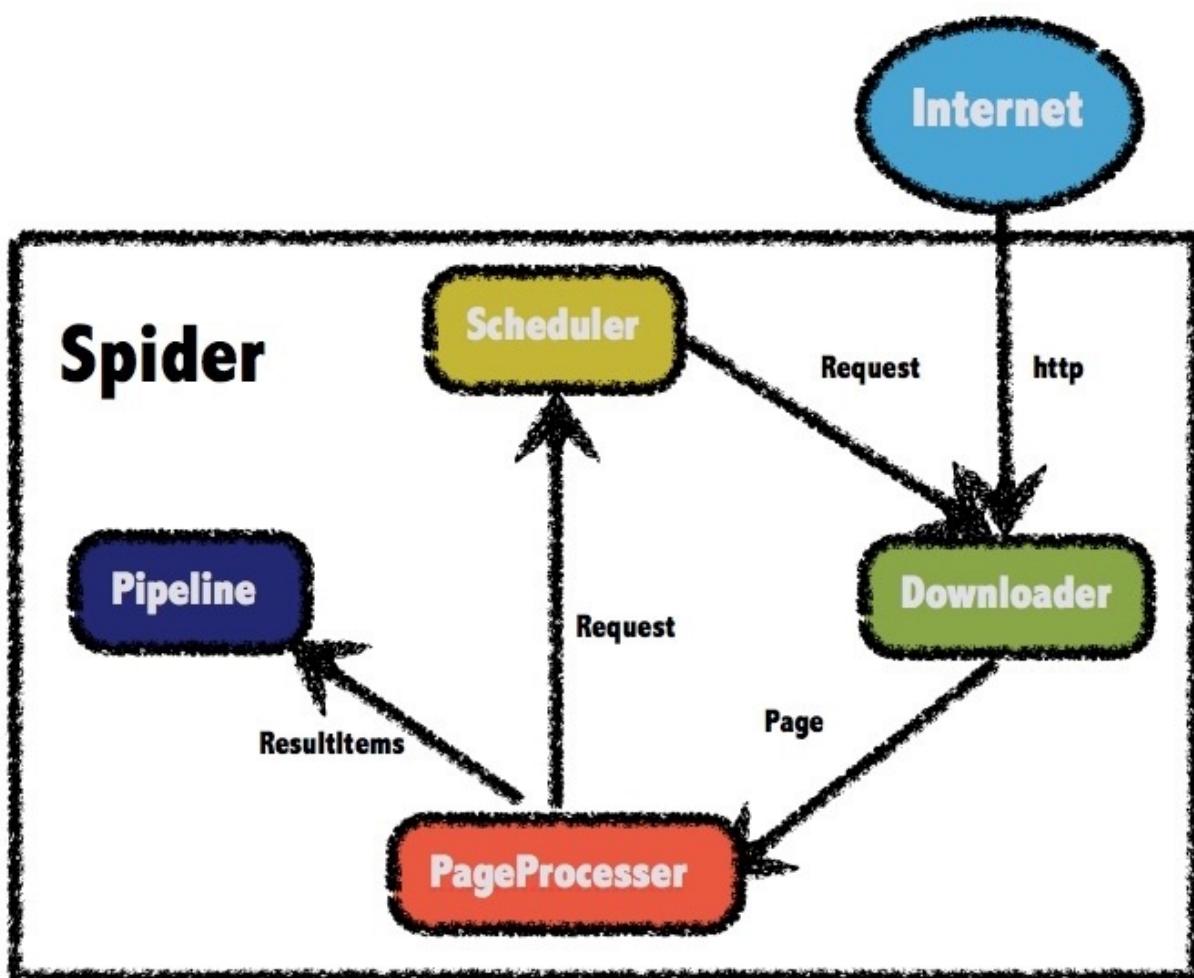
官网：[WebMagic](#)

可以在我的项目中查看这一部分的代码，这个模块叫做spider。

[trip-search - 码云](#)

[trip-search - Github](#)

构架简单介绍



四个组件介绍

Downloader

Downloader负责从互联网上下载页面，以便后续处理。WebMagic默认使用了Apache HttpClient作为下载工具。

PageProcessor

PageProcessor负责解析页面，抽取有用信息，以及发现新的链接。WebMagic使用Jsoup作为HTML解析工具，并基于其开发了解析XPath的工具Xsoup。在这四个组件中，PageProcessor对于每个站点每个页面都不一样，是需要使用者定制的部分。

Scheduler

Scheduler负责管理待抓取的URL，以及一些去重的工作。WebMagic默认提供了JDK的内存队列来管理URL，并用集合来进行去重。也支持使用Redis进行分布式管理。除非项目有一些特殊的分布式需求，否则无需自己定制Scheduler。

Pipeline

Pipeline负责抽取结果的处理，包括计算、持久化到文件、数据库等。WebMagic默认提供了“输出到控制台”和“保存到文件”两种结果处理方案。Pipeline定义了结果保存的方式，如果你要保存到指定数据库，则需要编写对应的Pipeline。对于一类需求一般只需编写一个Pipeline。

构建项目

怎么使用这里也不细讲，文档中说的很清楚。文章中有什么不懂的，可以参考webmagic官方文档，推荐先看完官网文档，并搞懂示例。

引入依赖

我这里用的是gradle来管理工程，所以只要在项目根目录的 `build.gradle` 文件中引入需要的依赖，gradle会自动解决相关的一系列依赖，前提是网好:-)（这里的网好是指能正常使用国际互联网，当然用国内maven镜像也是可以的，但根据我的经验，它总是抽风）。

```
dependencies {  
    testCompile group: 'junit', name: 'junit', version: '4.11'  
    compile "us.codecraft:webmagic-core:0.5.3"  
    compile "us.codecraft:webmagic-extension:0.5.3"  
}
```

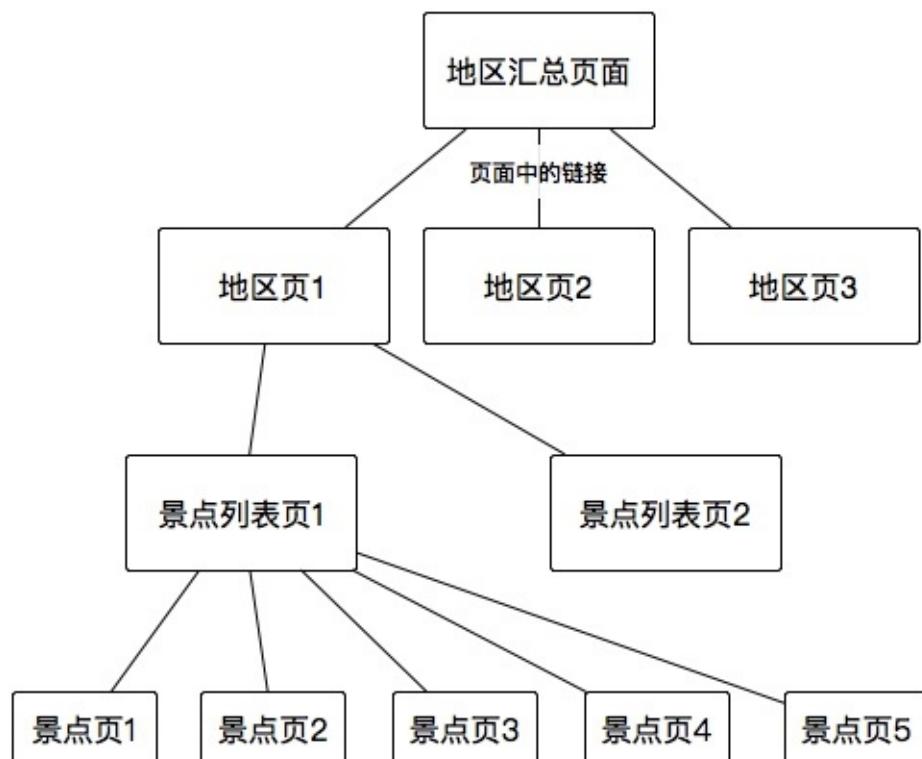
定制页面解析代码

四大组件的说明前面讲过了，对于一般的爬虫来讲，只要定制下 `PageProcessor` 就行了，这里也不啰嗦，别的都不讲，就说怎么抓。

正则表达式

对于爬虫最重要一项技能就是正则表达式，关于正则的只是非常多，一时半会也学不完，我也是现学现卖，有了基本的了解之后，后面有什么需要自己查就好了。

我抓取的是去哪儿网的景点页面，一共需要三个正则表达式。先介绍下我要抓取的站点的结构。



首先我找到了地区的汇总页面，这个汇总页面中有很多地区页的链接，我们要识别出来这些链接，并加入待抓取队列，然后如果抓到了地区页面，要在这个页面中找到这个地区的景点列表页；然后如果抓到了景点列表页，从这个页面我们要找到景点页的链接，并且还有景点列表页的不同页数的链接；最后才是我们需要的景点页，可以对其进行解析或者直接把页面保存下来。前面的所有链接都是帮助找到景点页的帮助页面，而景点页才是真正的目标页面。

注意：如不特殊说明，本文下面所提到的代码都是Git项目trip-search下的
的 `sightSpider/src/main/java/com/fliaping/trip/spider/QunarPageProcessor.java` 文件中。

我的种子页面是

```
public static final String URL_SEED = "http://travel.qunar.com/place/";
```

先来看下正则基础，可以参考这篇文章 [正则表达式30分钟入门教程](#)

表1.常用的元字符：

代码	说明
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线或汉字
\s	匹配任意的空白符
\d	匹配数字
\b	匹配单词的开始或结束
^	匹配字符串的开始
\$	匹配字符串的结束

表2.常用的限定符:

代码/语法	说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n到m次

另外，我们可以通过 [] 来进行字符匹配，例如 [aeiou] 就匹配任何一个英文元音字母，[0-9] 匹配一位范围是0~9的数字。[a-z] 匹配所有的小写字母。

注意：如果要匹配真实的 . * ? 之类的字符就需要在前面加上转义字符，如 \. * \? 。但是由于有些语言本身的相关转义规则，在写代码的时候可能有些稍微的改动。例如 java中需要将 \ 写成 \\ ，例如匹配一个 . 需要写成 \\. 。

按照前面的分析，我写了如下三个正则表达式：

```
public final String URL_PLACE = "^\http://travel\\.qunar\\.com/p-cs\\d+-[a-z]+$";
//这里的需要匹配的是地区页，像这样的http://travel.qunar.com/p-cs299878-shanghai
```

```
public final String URL_SIGHT_LIST = "^\http://travel\\.qunar\\.com/p-cs\\d+-[a-z]+-jingdian$";
//这里的需要匹配的是景点列表页，像这样的http://travel.qunar.com/p-cs299878-shanghai-jingdian-1-2
```

```
public final String URL_SIGHT = "^\http://travel\\.qunar\\.com/p-oi\\d+-[a-z]+$";
//这里的需要匹配的是景点页，像这样的http://travel.qunar.com/p-oi704314-huaihailushangyejie
```

PageProcessor的实现

webmagic有个注释模式来写爬虫，非常简洁优美，但是这个规则有点复杂，不太适合那样写，最后还是决定用实现 PageProcessor 的方式来写。

实现 PageProcessor 最重要的是实现页面处理函数 `public void process(Page page)`，在这个函数中我们会对页面进行一些解析，并且将我们需要的链接加入待抓列表中。例如：

```
//从种子页得到地区页，分析如果是种子页，则提取地区页链接，加入抓取队列
if (page.getUrl().regex(URL_SEED).match()){
    List<String> place_links = page.getHtml().links().regex(URL_PLACE).all();
    page.addTargetRequests(place_links);
}
```

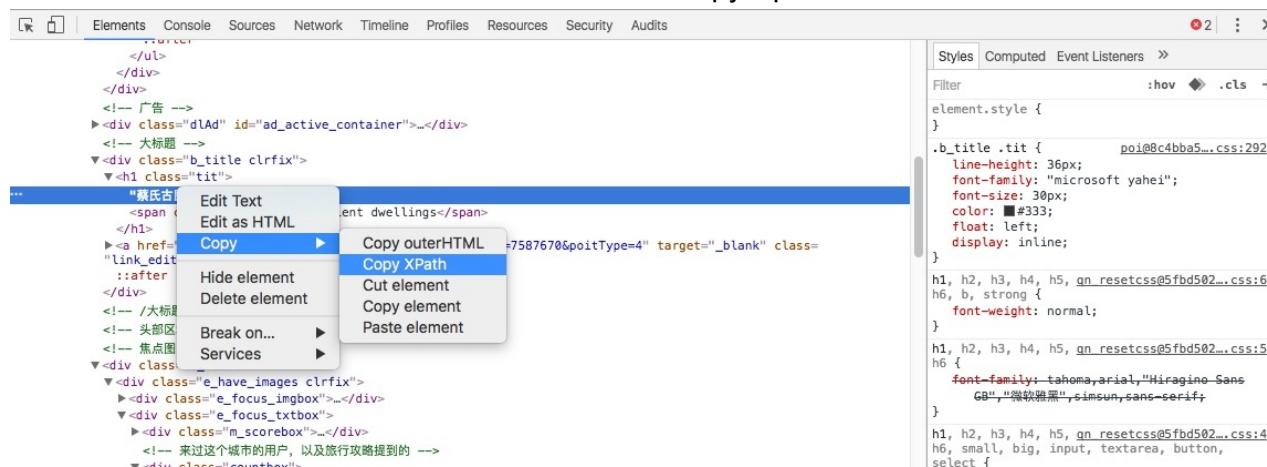
如果是目标页面，对其进行分析，并将需要的信息加入结果中，最后由Pipeline进行输出

```
else if (page.getUrl().regex(URL_SIGHT).match()){

    System.out.println("景点:"+page.getHtml().xpath("//*[@id=\"js_mainleft\"]/div[3]/h1/text()").toString());
    //得到景点名字
    page.putField("sight_name", page.getHtml().xpath("//*[@id=\"js_mainleft\"]/div[3]/h1/text()").toString());
    if(page.getResultItems().get("sight_name") == null){
        page.setSkip(true); //跳过这个页面，不进行输出
        page.addTargetRequest(page.getUrl().toString());
    }
    //获取评分
    Elements els = page.getHtml().getDocument().getElementsByClass("cur_score");
    if( els.size() > 0){
        page.putField("sight_score_qunar", els.get(0).text());
    }
    //获取坐标
    page.putField("sight_coordinate", page.getHtml().xpath("//*[@id=\"js_mainright\"]/div/div[2]/div/@latlng").toString());
    //获取图片
    page.putField("sight_cover", page.getHtml().xpath("//*[@id=\"idNum\"]/li[0]/div[2]/img/@src").toString());
    //将页面输出
    page.putField("html1",page.getHtml().toString());

}
```

如果利用jsoup来解析页面中的信息，请参考之前专门讲jsoup的文章。如果用XPath进行元素的选取，可以利用Chrome浏览器的审查元素中的Copy Xpath这个功能。



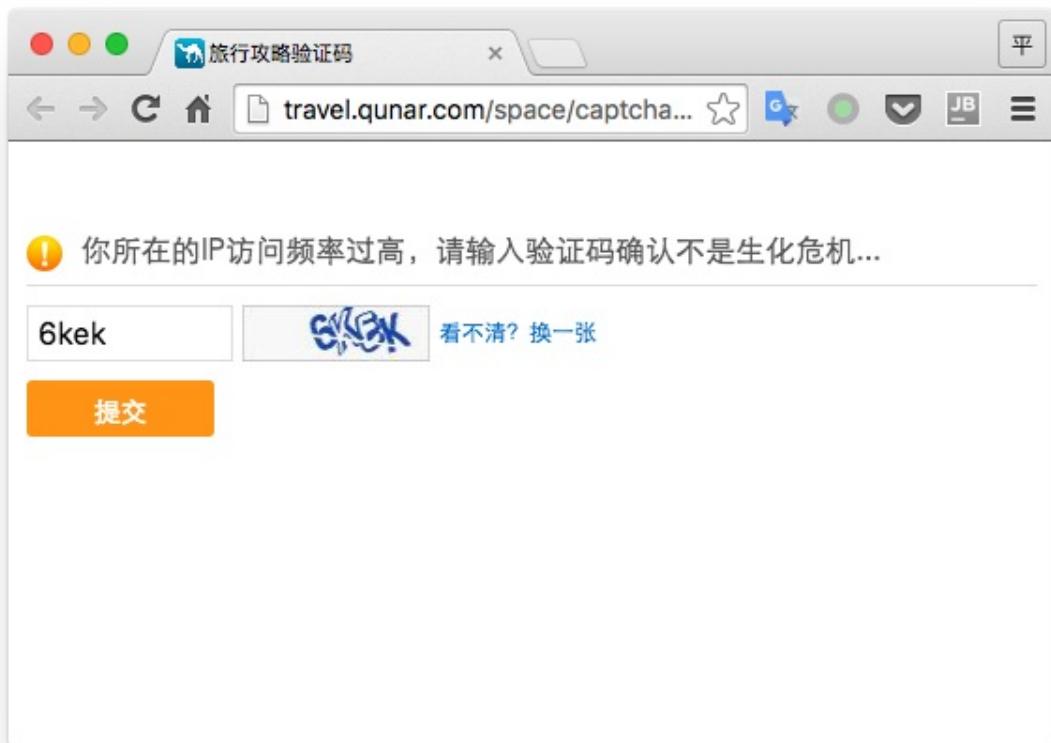
抓取策略及验证码

因为网站基本都有反爬虫策略，所以我们需要进行一些摸索，知道反爬虫系统的一些脾气，经过测试我设置的是爬取间隔30秒，每次25个线程同时抓取，这样大概20-30分钟会触发一次验证码，因为机器识别验证码的学习成本比较大，我就简单点，每次我手动输入了。

站点的爬取的相关信息设置：

```
private Site site = Site
    .me()
    .setDomain("travel.qunar.com")
    .setSleepTime(30000)
    .setUserAgent("Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.102 Safari/537.36");
```

如果出现让输入验证码的页面，就自动用浏览器打开这个页面。



```
if (page.getHtml().regex("访问频率过高").match()){
    //因为是多线程,必须进行同步
    synchronized (Main.qunarSpider){
        long thisTime = System.currentTimeMillis();

        //认证时间超过10秒的才回再次出发验证码,因为有可能刚输完验证码,当前的这个线程刚离开,
        //另外一个阻塞的线程进来再次让输验证码。
        if(thisTime-Main.lastVerifyTime > 10000){
            Main.qunarSpider.stop();
            openUrlInBrowser(page.getUrl().toString()); //自己写的函数,在默认浏览器
        }
        打开这个链接
        Scanner scanner = new Scanner(System.in);
        scanner.next(); //输入一个字符,手动确认已经填过验证码

        Main.lastVerifyTime = System.currentTimeMillis();
        Main.qunarSpider.start();
    }
    page.addTargetRequest(page.getUrl().toString());
}
```

默认浏览器中打开链接

```

public void openUrlInBrowser(String url){
    try {
        java.net.URI uri = java.net.URI.create(url);

        // 获取当前系统桌面扩展

        java.awt.Desktop dp = java.awt.Desktop.getDesktop();

        // 判断系统桌面是否支持要执行的功能

        if (dp.isSupported(java.awt.Desktop.Action.BROWSE)) {
            //File file = new File("D:\\aa.txt");
            //dp.edit(file); // 编辑文件

            dp.browse(uri); // 获取系统默认浏览器打开链接
            // dp.open(file); // 用默认方式打开文件
            // dp.print(file); // 用打印机打印文件
        }

        } catch (java.lang.NullPointerException e) {

            // 此为uri为空时抛出异常

            e.printStackTrace();
        }

        } catch (java.io.IOException e) {

            // 此为无法获取系统默认浏览器

            e.printStackTrace();
        }
    }
}

```

启动抓取

我将结果存在json文件中，因为有些页面的结构有些不同，在抓的时候进行调试不靠谱，所以我把页面保存下来，有问题本地分析比较快。

以下代码片段是 `SightSpider/src/main/java/com/fliaping/trip/spider/Main.java` 文件中

```

qunarSpider = Spider.create(new QunarPageProcessor());
qunarSpider.setScheduler(new FileCacheQueueScheduler("data"))
    .addUrl(QunarPageProcessor.URL_SEED)
    //.addPipeline(new ConsolePipeline())
    .addPipeline(new JsonFilePipeline("data"))
    .thread(25)
    .run();

```

运行该文件的 `main` 函数即可进行抓取。

3. 整理信息部分

本章将数据的导入和建立索引，主要是关于Solr的相关配置使用，包括分词器、空间搜索、数据导入等相关配置使用。

Solr5的主要配置文件有 `solrconfig.xml` 和 `managed-schema`，另外一些还有 `solr.xml`，数据导入配置，ZooKeeper配置等。

这里详细介绍两个主要的配置文件。

solrconfig.xml

`solrconfig.xml`文件是solr的主配置文件，主要配置高亮、数据源、索引大小、索引合并等所有的索引策略。

该文件位于 `$SolrHome/$Core/conf` 文件夹中，这个配置包含了solr自身的一些参数，主要包括 lib、数据目录、索引配置、更新处理、查询处理、缓存、请求分发、高亮等。可参考官方文档：[Configuring solrconfig.xml](#),以下内容整理自[系统全面的认识Solr](#)。

`<luceneMatchVersion>` 声明使用的lucene的版本。

`<lib>` 配置solr用到的jar包，具体语法示例中基本都有了。

`<dataDir>` 如果不用 `$SolrHome/data` 目录，可以指定其它别的目录来存放所有索引数据。如果使用了 replication，它可以匹配 replication 配置。如果这个目录不是绝对的，那会是当前 servlet 容器工作目录下的相对目录。

`<directoryFactory>` 索引文件的类型，默认 `solr.NRTCachingDirectoryFactory` 这个文件类型包装了 `solr.StandardDirectoryFactory` 和小文件内存缓存的类型，来提供NRT搜索性能。NRT(near-real-time)近实时。

`<indexConfig>` 主要索引相关配置

`<writeLockTimeout>` IndexWriter写锁过时的时间，默认1000

`<maxIndexingThreads>` 最大索引的线程数，默认8

`<useCompoundFile>` 是否使用混合文件，Lucene默认是 `true`，solr默认是 `false`

`<ramBufferSizeMB>` 使用的内存的大小，默认100，这个实际用的时候应该修改大一点。

`<ramBufferdDocs>` 内存中最大的文档数，默认1000

`<mergePolicy>` 索引合并的策略。默认 `TiereMergePolicy`，合并大小相似的段，与 `LogByteSizeMergePolicy` 相似。这个可以合并不相邻的段，能够设置一次合并多少个段，`maxMergeAtOnce` 以及每层能合并多少个段 `segmentsPerTier`。

`<mergeFactor>` 每次合并索引的时候获取多少个段，默认10。等同于同时设置了 `maxMergeAtOnce` 和 `segmentsPerTier` 两个参数。

`<mergeScheduler>` 段合并器，背后有一个线程负责合并，默认 `ConcurrentMergeScheduler`。

<lockType> 文件锁的类型，默认 native，使用 NativeFSLockFactory。

<unlockOnStartup> 默认 false

<termIndexInterval> Lucene每次加载到内存的terms数，默认128

<reopenReaders> 如果是 true 时，IndexReaders 能够被reopened，而不是先关闭再打开，
默认 true

<deletionPolicy> 删除策略，用户可以自己定制，solr默认的是 SolrDeletionPolicy，是solr
标准的删除策略，允许在一定时间内保存索引提交点，来支持索引复制，以及快照等特性。
可以设置 maxCommitsToKeep 保存提交的数量、maxOptimizedCommitsToKeep 保存的优化条件的
数量、maxCommitAge 删除所有commit points的时间。

<infoStream> 为了调试，Lucene提供了这个参数，如果是 true 的话，IndexWriter 会像设
置的文件中写入debug信息。

<jmx> 一般不需要设置具体可以查看[wiki文档](#)

<updateHandler> 更新的Handler，默认 DirectUpdateHandler2

<updateLog><str name="dir"> 配置更新日志的存放位置

<autoCommit> 硬自动提交，可以配置maxDocs即从上次提交后达到多少文档后会触发自动提
交；maxTime时间限制；openSearcher，如果设为false，导致索引变化的最新提交，不需要
重新打开searcher就能看到这些变化，默认false。

<autoSoftCommit> 如自动提交，与前面的 <autuCommit> 相似，但是它只是让这些变化能够看
到，并不保证这些变化会同步到磁盘上。这种方法比硬提交要快，而且更接近实时更友好。

<listerner event> update 时间监听器配置，postCommit 每一次提交或优化命令后触发，

poatOptimize 每次优化命令后触发。RunExecutableListener 每次调用后执行一些其他操作。
配置项：

```
exe - the name of the executable to run
dir - dir to use as the current working directory. (default=".")
wait - the calling thread waits until the executable returns.
       (default="true")
args - the arguments to pass to the program. (default is none)
env - environment variables to set. (default is none)
```

<indexReaderFactory> 这个配置项用户可以自己扩展 IndexReaderFactory，可以自己实现自
己的

IndexReader。如果要明确声明使用的Factory则可以如下配置：

```
<indexReaderFactory name="IndexReaderFactory"
                     class="solr.StandardIndexReaderFactory">
  <int name="setTermIndexDivisor">12</int>
</indexReaderFactory >
```

`<query>` 配置检索词相关参数以及缓存配置参数。

`<maxBooleanClauses>` 每个BooleanQuery中最大Boolean Clauses的数目，默认1024。

`<filterCache>` 为 IndexSearcher 使用，当一个 IndexSearcher Open时，可以被重新赋于原来的值，或者使用旧的 IndexSearcher 的值，例如使用LRUCache时，最近被访问的Items将被赋予 IndexSearcher 。Solr默认是 FastLRUCache 。

cache介绍：<http://blog.csdn.net/phinecos/article/details/7876385> filterCache存储了无序的lucene documentid集合，该cache有3种用途：

1) filterCache存储了filterqueries("fq"参数)得到的document id集合结果。Solr中的query参数有两种，即q和fq。如果fq存在，Solr是先查询fq（因为fq可以多个，所以多个fq查询是个取结果交集的过程），之后将fq结果和q结果取并。在这一过程中，filterCache就是key为单个fq（类型为Query），value为document id集合（类型为DocSet）的cache。对于fq为range query来说，filterCache表现出其有价值的一面。

2) filterCache还可用于facet查询，facet查询中各facet的计数是通过对满足query条件的documentid集合（可涉及到filterCache）的处理得到的。因为统计各facet计数可能会涉及到所有的doc id，所以filterCache的大小需要能容下索引的文档数。

3) 如果solrconfig.xml中配置了 `<useFilterForSortedQuery/>`，那么如果查询有filter（此filter是一需要过滤的DocSet，而不是fq，我未见得它有什么用），则使用 filterCache 。

`<queryResultCache>` 缓存查询的结果集的docs的id。

`<documentCache>` 缓存document对象，因为document中的内部id是transient,所以autowarmed为0，不能被autowarmed。

`<fieldValueCache>` 字段缓存

`<cache name="">` 用户自定义一个cache，用来缓存指定的内容，可以用来缓存常用的数据，或者系统级的数据，可以通过 `SolrIndexSearcher.getCache()` , `cacheLookup()` , and `cacheInsert()` 等方法来操作。

`<enableLazyFieldLoading>` 保存的字段，如果不需的话就懒加载，默认true。

`<useFilterForSortedQuery>` 一般来讲用不到，只有当你频繁的重复同一个搜索，并且使用不同的排序，而且它们都不用“score”

`<queryResultWindowSize>` queryResultCache的一个参数。

`<queryResultMaxDocsCached>` queryResultCache的一个参数。

`<listener event="newSearcher" class="solr.QuerySenderListener">` query的事件监听器。

`<useColdSearcher>` 当一个检索请求到达时，如果现在没有注册的searcher，那么直接注册正在预热的searcher并使用它。如果设为false则所有请求都要block，直到有searcher完成预热。

`<maxWarmingSearchers>` 后台同步预热的searchers数量。

`<requestDispatcher handleSelect="false">` solr接受请求后如何处理，推荐新手使用false

`<requestParsers enableRemoteStreaming="true" multipartUploadLimitInKB="2048000" formdataUploadLimitInKB="2048" />` 使系统能够接收远程流。

`<httpCaching never304="true">` http cache参数，solr不输出任何HTTP Caching相关的头信息。

`<requestHandler>` 接收请求，根据名称分发到不同的handler。

```
"/select" 检索SearchHandler
"/query" 检索SearchHandler
"/get" RealTimeGetHandler
"/browse" SearcherHandler
"/update" UpdateRequestHandler
"/update/json" JsonUpdateRequestHandler
"/update/csv" CSVRequestHandler
"/update/extract" ExtractingRequestHandler
"/analysis/field" FieldAnalysisRequestHandler
"/analysis/document" DocumentAnalysisRequestHandler
"/admin" AdminHandlers
"/replication" 复制，要有主，有从
```

`<searchComponent>` 注册searchComponent。 `<queryResponseWriter>` 返回数据 `<admin>` `<defaultQuery>` 默认的搜索词

managed-schema

managed-schema在solr5之前叫schema.xml，文件主要配置索引和查询的字段信息，定义了所有的数据类型和各索引字段的信息（如类型，是否建立索引，是否存储原始信息等）。

fields块配置

```
<field name="" type="" indexed="" stored="" required="" multiValued="" omitNorms="" termVectors="" termPositions="" termOffsets="">
```

- name：名称
- type：类型从 fieldType中取
- indexed：是否索引

- stored : 是否保存
- required : 是否必须
- multiValuer : 在同一篇文档中可以有多个值
- omitNorms : true的话忽略norms
- termVectors : 默认false , 如果是true的话 , 要保存字段的term vector
- termPositions : 保存term vector的位置信息
- termOffsets : 保存term vector的偏移信息
- default : 字段的默认值

`<dynamicField>` 动态字段 , 当不确定字段名称时采用这种配置 `<CopyField>`

types 块配置 `<types>` 块内 , 声明一系列的 `<fieldtype>` , 以 Solr fieldtype 类为基础 , 如同默认选项一样来配置自己的类型。

任何 `FieldType` 的子类都可以作为 `field type` 来使用 , 使用时可以用完整的包名 , 如果 `field type` 类在 solr 里 , 那可以用 “solr” 代替包名。提供多种不同实现的普通数据类型 (`integer, float` 等) 。想知道怎么样被 Solr 正确地加载自定义的数据类型 , 请看 : SolrPlugins 通用的选项有 :

- name : 类型名称
- class : 对应于 solr fieldtype 类
- sortMissingLast=true|false 如果设置为 true , 那么对这个字段排序的时候 , 包含该字段的文档就排到不包含该字段的文档前面。
- sortMissingFirst=true|false 如果设置为 true , 那么对这个字段排序的时候 , 没有该字段的文档排在包含该字段的文档前面
- precisionStep 如何理解precisionStep呢 ? 需要一步一步来 : 参考文档 : <http://blog.csdn.net/fancyerii/article/details/7256379> 1) precisionStep是在做range search的起作用的 , 默认值是 4 2) 数值类型 (int float double) 在 Lucene 里都是以 string 形式存储的 , 当然这个 string 是经过编码的 3) 经过编码后的 string 保证是顺序的 , 也就是说 num1 > num2 , 那么 strNum1 > strNum2 4) precisionStep 用来分解编码后的 string , 例如有一个precisionStep , 默认是 4 , 也就是隔 4 位索引一个前缀 , 比如 0100,0011,0001,1010 会被分成下列的二进制位 “0100,0011,0001,1010” , “0100,0011,0001” , “0100,0011” , “0100”。这个值越大 , 那么索引就越小 , 那么范围查询的性能 (尤其是细粒度的范围查询) 也越差 ; 这个值越小 , 索引就越大 , 那么性能越差。
- positionIncrementGap 和 multiValued 一起使用 , 设置多个值之间的虚拟空白的数量。字段有多个值时使用 , 如果一篇文档有两个 title

title1: ab cd title2: xy zz

如果 `positionIncrementGap=0` , 那么这四个 term 的位置为 0,1,2,3 。如果检索 “cd xy” 那么能够找到 , 如果你不想让它找到 , 那么就需要调整 `positionIncrementGap` 值。如 100 , 那么这是位置为 0,1,100,101 。这样就不能匹配了。

<fieldType name="random" class="solr.RandomSortField" indexed="true" /> 这个字段类型可以实现伪随机排序。

analyzer配置

```
- <fieldType name="text_general" class="solr.TextField" positionIncrementGap="100">
  - <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true" />
    + <!-- -->
    <filter class="solr.LowerCaseFilterFactory" />
  </analyzer>
  - <analyzer type="query">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" enablePositionIncrements="true" />
    <filter class="solr.SynonymFilterFactory" synonyms="synonyms.txt" ignoreCase="true" expand="true" />
    <filter class="solr.LowerCaseFilterFactory" />
  </analyzer>
</fieldType>
```

包括

tokenizer和filter，可以配置多个filter

其他配置 <uniqueKey> 唯一字段，除非这个字段标记了“required=false”，否则默认为 required
 字段 <copyField> 一个源字段一个目的字段，将源字段的内容拷贝到目的字段，可以将多个
 字段合并，也可以对同一个字段，不同索引方式。 <defaultSearchField> 默认的搜索字段
 <solrQueryParser defaultOperator="OR"/> 默认的检索词间的关系

数据导入配置

这个配置文件的名字是在 solrconfig.xml 中配置的，不过通常我们用 data-config.xml 这个名字，它是配置数据库信息，比如配置何种数据源datasource，全量索引，增量索引的数据库查询等。

由于这个配置项要讲的东西较多，单独在[导入Mysql数据到Solr中](#)中讲解。

solr.xml

solr.xml主要是配置solr主目录中的索引库即SolrCore，一个solr服务可以配置多个SolrCore，即可以管理多个索引库。

```
<solr>

<solrcloud>

<str name="host">${host:}</str>
<int name="hostPort">${jetty.port:8983}</int>
<str name="hostContext">${hostContext:solr}</str>

<bool name="genericCoreNodeNames">${genericCoreNodeNames:true}</bool>

<int name="zkClientTimeout">${zkClientTimeout:30000}</int>
<int name="distribUpdateSoTimeout">${distribUpdateSoTimeout:600000}</int>
<int name="distribUpdateConnTimeout">${distribUpdateConnTimeout:60000}</int>

</solrcloud>

<shardHandlerFactory name="shardHandlerFactory"
  class="HttpShardHandlerFactory">
  <int name="socketTimeout">${socketTimeout:600000}</int>
  <int name="connTimeout">${connTimeout:60000}</int>
</shardHandlerFactory>

</solr>
```

这是默认的内容，一般情况下不需要更改这里的东西，并且我也没怎么理解，所以等到我明白了再说。

分词的基础概念

为什么要进行分词

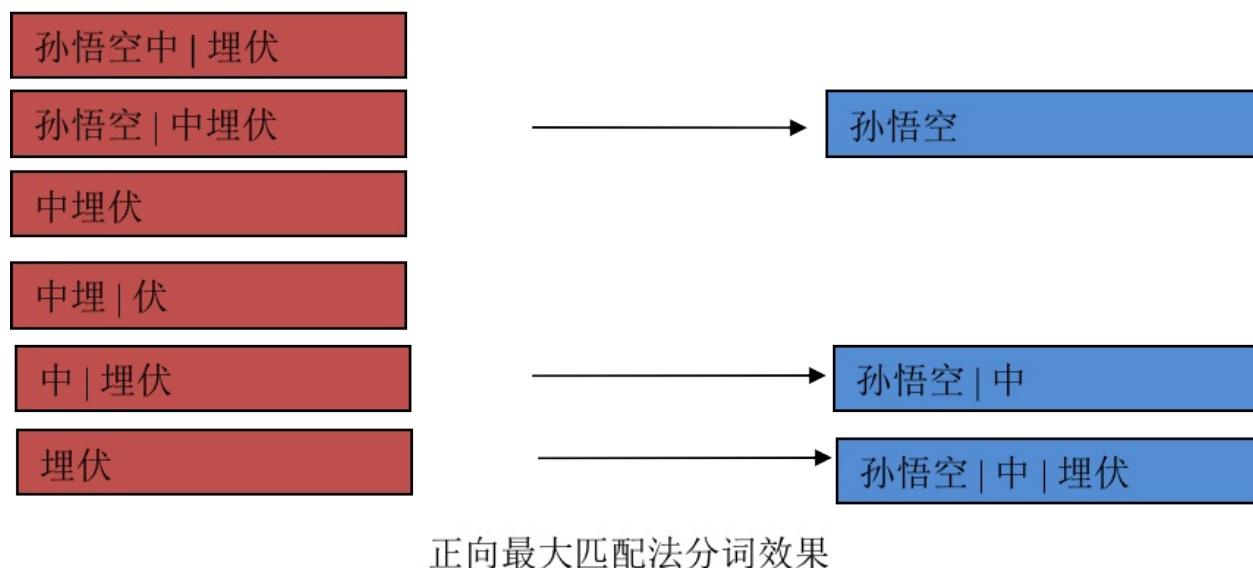
中文分词(Chinese Word Segmentation) 指的是将一个汉字序列切分成一个一个单独的词。

分词就是将连续的字序列按照一定的规范重新组合成词序列的过程。我们知道，在英文的行文中，单词之间是以空格作为自然分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，唯独词没有一个形式上的分界符，虽然英文也同样存在短语的划分问题，不过在词这一层上，中文比之英文要复杂的多、困难的多。

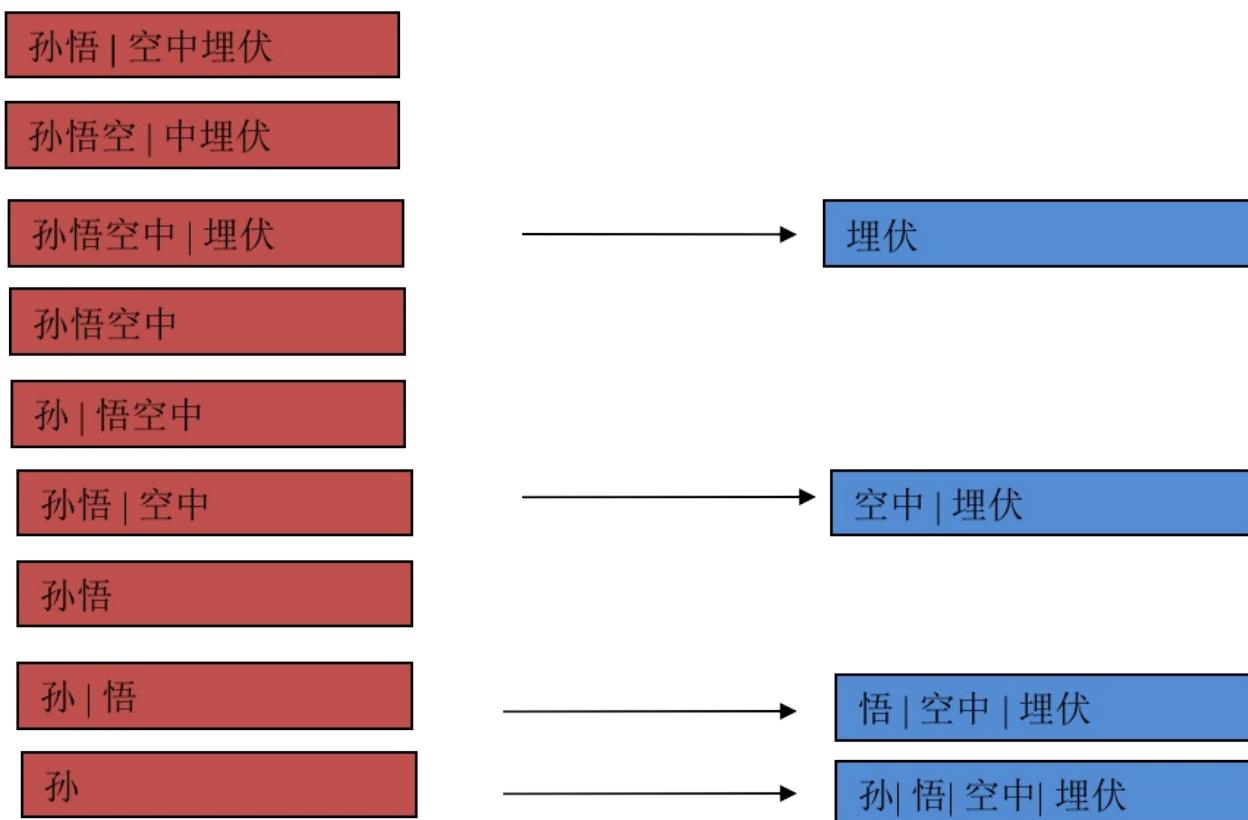
分词方法

1. 基于字符串匹配的分词方法（机械分词法）：将待分字符串与词典进行匹配

- 正向最大匹配法（由左到右的方向）
- 逆向最大匹配法（由右到左的方向）
- 最小切分法（每句话切分的词数最少）
- 双向最大匹配法（进行由左到右、由右到左两次扫描）



正向最大匹配法分词效果



逆向最大匹配法分词效果

1. 基于理解(Semantic)的分词法：在分词的同时引入句法和语义信息处理歧义
2. 基于统计的分词方法：相邻字频率越高越成词；多用于新词识别(补充词典)
3. 复合分词法（上述3种方法的综合运用和相互补充）

那些高级的分词方法难度较大，所以用最简单的基于字符串匹配的分词方法，我们选择的mmseg4j分词器有Simple和Complex两种模式，都是基于正向最大匹配。

常用分词器介绍

- mmseg4j 用 Chih-Hao Tsai 的 MMSeg 算法(<http://technology.ctsai.org/mmseg/>)实现的中文分词器，并实现 lucene 的 analyzer 和 solr 的 TokenizerFactory 以方便在Lucene和Solr中使用，并且最后更新时间为2015年，支持Solr5。
- IKAnalyzer，采用了特有的“正向迭代最细粒度切分算法”，具有60万字/秒的高速处理能力。采用了多子处理器分析模式，支持：英文字母、数字、中文词汇等分词处理，兼容韩文、日文字符。可以说IK也是很不错的分词器，不过由于他对Solr5兼容不是很好，所以最后也没用它。
- paoding，Paoding's Knives 中文分词具有极高效率和高扩展性。引入隐喻，采用完全的面向对象设计，构思先进。但是已经很久不更新了。

下表出自[中文分词器性能比较](#)

名称	最近更新	速度(网上情报)	扩展性支持、其它
mmseg4j	2013	complex 60W字/s (1200 KB/s) simple 100W字/s (1900 KB/s)	使用sougou词库，也可自定义 (complex\simple\MaxWord)
IKAnalyzer	2012	IK2012 160W字/s (3000KB/s)	支持用户词典扩展定义、支持自定义停止词 (智能\细粒度)
Ansj	2014	BaseAnalysis 300W字/s hlAnalysis 40W字/s	支持用户自定义词典，可以分析出词性，有新词发现功能
paoding	2008	100W字/s	支持不限制个数的用户自定义词库

整合mmseg4j到Solr5.5

下载：[mmseg4j-solr](#)

有很多个版本，支持Solr5的为 `mmseg4j-solr-2.3.0.jar`，如果你在代码中使用还可以用maven仓库来直接添加依赖。要想整合进Solr，就需要另外的包 `mmseg4j-core-1.10.jar`，其实 `mmseg4j-core` 这个包才是分词器的核心，`mmseg4j-solr` 只是兼容Solr的接口。

放置jar包到正确位置

将这两个包放进servlet的类库目录中，这里的路径就是 `$Solr.Install.Dir/server/solr-webapp/webapp/WEB-INF/lib`。如果你用的是Tomcat作为servlet，那么可以路径就应该是在 `$TomcatDir/webapps/solr/WEB-INF/lib`。

创建一个SolrCore

首先保证Solr正常启动了，下面有三中创建方法。

- 方法一：在命令行中新建。转到 \$Solr.Install.Dir ,输入一下命令：

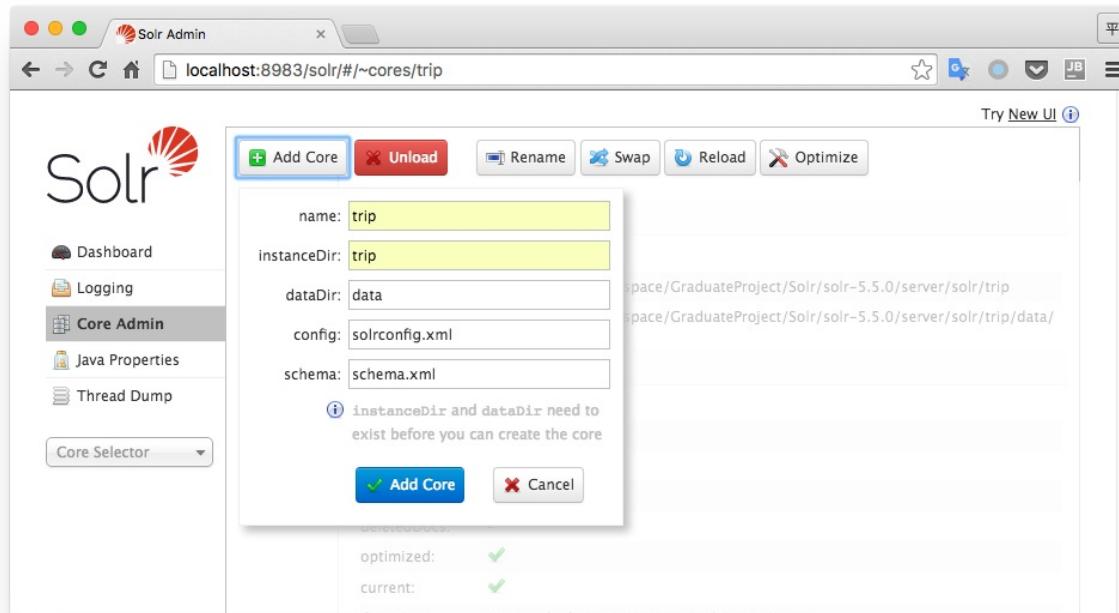
```
bin/solr create -c trip
```

成功可以看到如下输出日志

```
Copying configuration to new core instance directory:  
/Users/Payne/Workspace/GraduateProject/Solr/solr-5.5.0/server/solr/trip  
  
Creating new core 'trip' using command:  
http://localhost:8983/solr/admin/cores?action=CREATE&name=trip&instanceDir=trip  
  
{  
  "responseHeader":{  
    "status":0,  
    "QTime":1217},  
  "core":"trip"}  

```

- 方法二：在管理界面创建，首先在你的 \$SolrHome 目录下建立要创建Core的文件夹，然后把同目录下的 configsets/basic_configs/conf 文件夹copy到你新建的Core文件夹中，之后再管理界面就可新建了，如下图所示。



- 方法三：在方法二copy完成配置文件的基础上，可以通过URL-API来创建，假如我要创建一个名为trip的core，可以用如下的链接

```
http://localhost:8983/solr/admin/cores?action=CREATE&name=trip&instanceDir=trip
```

如果创建成功页面中会返回

```
<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">275</int>
  </lst>
  <str name="core">trip</str>
</response>
```

其实命令行创建就是把这个方法自动化的

修改配置文件

找到Core的配置文件夹，就是方法二中复制过来的那个，修改文件 \$SolrHome/trip/conf/managed-schema (在5.0前，该文件是shcema.xml，当然可以将该文件重命名为schema.xml,但不建议这么做) ，加入下面的内容并重启Solr，即可在Solr Admin 的console中看到新增的这些field了。

```

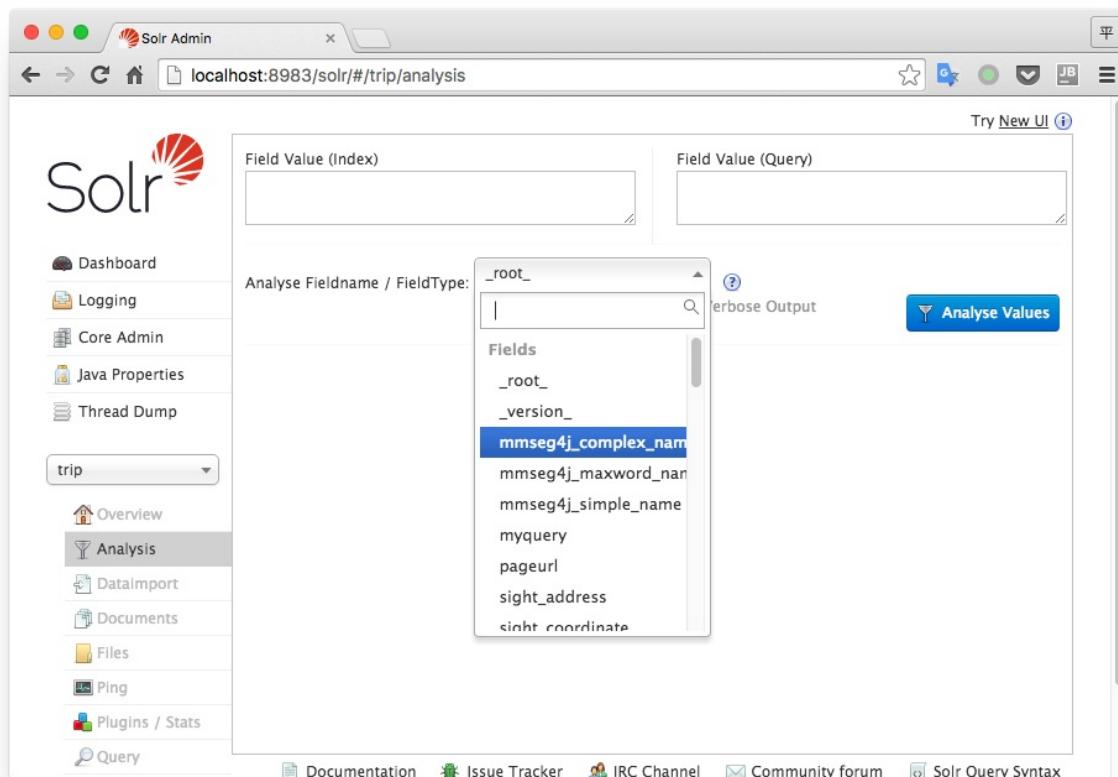
<!-- mmseg4j-->
    <field name="mmseg4j_complex_name" type="text_mmseg4j_complex" indexed="true" stored="true"/>
        <field name="mmseg4j_maxword_name" type="text_mmseg4j_maxword" indexed="true" stored="true"/>
            <field name="mmseg4j_simple_name" type="text_mmseg4j_simple" indexed="true" stored="true"/>

            <fieldType name="text_mmseg4j_complex" class="solr.TextField" positionIncrementGap="100" >
                <analyzer>
                    <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="complex" dicPath="/Users/Payne/Workspace/GitHub/trip-search/SolrHome/trip/conf"/>
                    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                </analyzer>
            </fieldType>
            <fieldType name="text_mmseg4j_maxword" class="solr.TextField" positionIncrementGap="100" >
                <analyzer>
                    <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="max-word" dicPath="/Users/Payne/Workspace/GitHub/trip-search/SolrHome/trip/conf"/>
                    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                </analyzer>
            </fieldType>
            <fieldType name="text_mmseg4j_simple" class="solr.TextField" positionIncrementGap="100" >
                <analyzer>
                    <tokenizer class="com.chenlb.mmseg4j.solr.MMSegTokenizerFactory" mode="simple" dicPath="/Users/Payne/Workspace/GitHub/trip-search/SolrHome/trip/conf"/>
                    <filter class="solr.StopFilterFactory" ignoreCase="true" words="stopwords.txt" />
                </analyzer>
            </fieldType>
<!-- mmseg4j-->

```

这里要注意 `dicPath`，是这个 `tokenizer` 的词库文件的路径，最好用绝对位置。

重启 Solr 后，即可在新创建的 trip 这个 core 的 Analysis 中看到 mmseg4j 新增的 field。



停用词字典

概念：

在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据（或文本）之前或之后会自动过滤掉某些字或词，这些字或词即被称为 Stop Words(停用词)。

停用词分为两类：功能词和词汇词。在这里我们不做区分，主要是屏蔽一些如“是，的，地，得”等一些功能性的词汇。

编辑配置目录中的 `stopwords.txt` 文件，添加停用词，每个词占一行。添加了如下停用词之后（记得要重启 Solr 哦），对 “九寨沟的水真是美丽极了” 这句话分词的结果如图所示，看到所分出来的词 “的，真是” 在停用词文件中有匹配，所以被屏蔽掉了。

```
是  
的  
啊  
哈  
嗯  
真是
```

MMST		text	九	寨	沟	的	水	真是	美丽	极了
		raw_bytes	[e4 b9 9d]	[e5 af a8]	[e6 b2 9f]	[e7 9a 84]	[e6 b0 b4]	[e7 9c 9f e6 98 af]	[e7 be 8e e4 b8 bd]	[e6 9e 81 e4 ba 86]
		start	0	1	2	3	4	5	7	9
		end	1	2	3	4	5	7	9	11
		positionLength	1	1	1	1	1	1	1	1
		type	word	word	word	word	word	word	word	word
		position	1	2	3	4	5	6	7	8

SF		text	九	寨	沟	水	美丽	极了
		raw_bytes	[e4 b9 9d]	[e5 af a8]	[e6 b2 9f]	[e6 b0 b4]	[e7 be 8e e4 b8 bd]	[e6 9e 81 e4 ba 86]
		start	0	1	2	4	7	9
		end	1	2	3	5	9	11
		positionLength	1	1	1	1	1	1
		type	word	word	word	word	word	word
		position	1	2	3	5	7	8

增加词库

mmseg4j默认是使用mmseg4j-core-1.10.0.jar中的words.dic,总共只有不到15万的中文词，另外我做的是旅游搜索，所以会有很多地名、经典名之类的专有词，所以，我们还需要另外增加词库。这些词库已经有很多现成的资源可以供我们使用，各大输入法厂商都有专有名词包供我们下载，但它们一般都是私有的二进制格式，不是文本文件，所幸有人做了词库转换软件，我们可以下载输入法的词库文件然后转成本文。

- 深蓝词库转换：[imewlconverter-github](#)，[下载](#)
- 搜狗词库：[细胞词库](#)
- 百度词库：[词库](#)

将做好的词库文件命名为 `word.dic`，放在前面 `dicPath` 配置的文件夹下，我这里就放在 `SolrHome` 的 `conf` 目录下。(名字一定要是 `word.dic`)

重启 Solr，再次进行Analysis，这次关闭了详细参数的显示。

Field Value (Query)	九寨沟的水真是美丽极了
---------------------	-------------

Verbose Output

Analyse Values

MMST | 九寨沟 | 的 | 水 | 真是 | 美丽 | 极了

SF | 九寨沟 | | 水 | | 美丽 | 极了

至此，已经完成mmseg4j的整合。

complex和maxword两种类型的区别

引用自：[Solr 5.x的搭建（Solr自带的Jetty Server）与mmseg4j中文分词](#)

我们假定我们的分词库中存在着“林书豪”，“书豪”，“林书”3个词。在mmseg4j-complex算法中，“林书豪”会被完整分词为“林书豪”，而mmseg4j-maxword中由于只支持两个字的分词，“林书豪”会被分词为“林书”，“书豪”。这也就以为这如果你选的是mmseg4j-complex算法，你要搜索出含有“林书豪”的内容，则你必须完整的输入“林书豪”才会能够搜得出结果，而在mmseg4j-maxword算法中，你只需要输入“林书”或者“书豪”就可以得出想要的结果了。

所以在实际开发过程中，我们常常需要在精度和广度之间得出权衡的时候，可以选择性的丰富词库，更改词库，比方我希望输入“林书”或者“书豪”的时候就可以得到我想要的结果，那么我就可以在词库中加入“林书”和“书豪”，并且使用mmseg4j-maxword算法，但是我希望的是输入完整的林书豪才能得到我希望的搜索结果，那么就需要使用mmseg4j-complex算法，而且词库中需要加入“林书豪”。

注意：

- 在words.dic中分词的顺序是很重要的，比如对于上面的例子“林书豪来中国了”，如果选择mmseg4j-complex算法，并且在词库的最后加入“来中国”，那么你可以看到分词的结果为后面的“来中国”将替代前面的“中国”。

既然涉及的是旅游搜索，那就不能少了空间搜索的内容。现在移动设备之广泛想必不必多说。根据eMarketer的新数据，2015年全球智能手机用户将达到19.1亿，2016年该指数将增长12.6%达到21.6亿。智能手机都会带有GPS模块，现在一个APP不获取下用户位置都觉得是吃亏，不过这也用户隐私的侵犯也很严重，尤其是在Android生态圈中，不过现在各大手机厂商的系统都对这一块做了一些限制。

空间搜索，又名Spatial Search，基于空间搜索技术，可以做到：

1. 对Point（经纬度）和其他的几何图形建索引
2. 距离计算：根据给定点计算它到其他点的距离。
3. 限定框过滤器：查找某些特定区域内所有匹配项（文档）。
4. 排序：根据到固定点的距离对搜索结果进行排序。
5. 相关度改进：使用距离作为记录中的增强因素，同时允许其他因素发挥作用。
6. 查询解析：在给出位置的地址或其他一些用户规定时，创建可用于根据索引数据进行搜索的编码表示。

空间搜索原理

在Solr中，空间搜索主要基于GeoHash和Cartesian Tiers 2个概念来实现。下面先来讲解一下这两个算法，虽然这两种算法可以很容易在网上找到资料，但是我在那里对其进行总结，并根据之前自己的知识和相似的技术、算法思想进行对比。后面将会是和Solr相关的如何进行配置和查询，虽然这些算法的实现比较复杂，好在Solr已经帮我们做好了，通过配置就可以使用。当然这里最需要感谢的美团的技术团队，这么无私的把这些技术分享出来，我基本上是靠美团的这篇文章来完成了这个模块的工作。

GeoHash算法

GeoHash即Geology和Hash的组合，使用hash算法的方法对地理信息进行编码，要想充分了解GeoHash，我们先来了解Hash。

各种Hash及其核心思想

Hash(散列)算法，是一种从任何一种数据中创建小的数字“指纹”的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（hash values, hash codes, hash sums, 或hashes）的指纹。

计算机中有关Hash的应用有很多的，例如java的HashMap、Hash Table、数据加密、数据完整性、错误校正等方面都有应用。它的基本思想上面的Wikipedia已经定义的比较好，我来分享下我的见解，hash就是把数据按照某种规则进行重组再表示，这个再表示比原始的数据

量要小，其实是有损的转换，因此不能从再表示中还原数据。另外由于再表示的取值空间比较小，会有不同的原始数据经过Hash之后得到相同的再表示，这也就意味着冲突，好的转换规则能最低限度的降低冲突概率。

和GeoHash比较相近的应用是Hash Table，它通过关键码值（Key value）来访问数据，把key通过一个hash函数生成Hashcode，这个Hashcode对应一个位置，而其value就放在这个位置，相当于是key和value所在位置的一个映射。我们知道线性的数据结构能够通过index(下标)来快速获取数据（非线性需要遍历才能找到），我们可以直接把下标当做key，这样的`<key, value>`也可以快速找到，但希望的是key可以是任何字符，那么经过hash函数的转换，Hash Table应运而生。

Hash Table是为了解决查询速度的问题，GeoHash当然也是为了解决这个问题，它主要解决的是地理位置的查找，不过其复杂程度比Hash Table高一些，不过也算比较容易理解。

GeoHash初识

其实GeoHash就是将二维的经纬度转换成字符串，并且这些字符串有个特征是同一个地区的前缀N位是相同的，如果两点的GeoHash字符串前缀相同的位数越多表示这两点越接近，因此将距离的计算转换成字符串相同位数的比较，虽然是其基本原理如此常见，但运算量的减小确是非常巨大的。

为了能让大家对我上面说的概念有个更直观的了解，于是我很恬不知耻地借用别人的图片来给大家看，也是蛮拼(lan)的。



上图可以看到每一个字符串代表了一个矩形区域（为啥是矩形不是方形？下面再说），只要是这个区域内的所有点经过GeoHash算法之后前5位一定是相同的。这就有个好处，那就是缓存啊，把数据按照区域缓存起来加快访问速度和减小负载，真是机智。



上面的图是啥意思呢？字符串越长，表示的范围越精确。5位的编码能表示10平方千米范围的矩形区域，而6位编码能表示更精细的区域（约0.34平方千米），也就是说在大格子里面画小个子，也从正面说明了前缀相同的位数越多表示这两点越接近。

GeoHash算法步骤

我们都知道地球的纬度为-90~90，经度为-180~180，我就用美团这篇文章的数据来讲吧。例如纬度为39.92324，经度为116.3906（这货真懒）

二分编码

1) 对区间[-90,90]进行二分为[-90,0],[0,90]，称为左右区间，可以确定39.92324属于右区间[0,90]，给标记为1； 2) 接着将区间[0,90]进行二分为 [0,45],[45,90]，可以确定39.92324属于左区间 [0,45]，给标记为0； 3) 递归上述过程，39.92324总是属于某个区间[a,b]，随着每次迭代区间[a,b]总在缩小，并越来越逼近39.928167； 4) 如果给定的纬度（39.92324）属于左区间，则记录0，如果属于右区间则记录1，这样随着算法的进行会产生一个序列1011 1000 1100 0111 1001，序列的长度跟给定的区间划分次数有关。

纬度范围	划分区间0	划分区间1	39.92324所属区间
(-90, 90)	(-90, 0.0)	(0.0, 90)	1
(0.0, 90)	(0.0, 45.0)	(45.0, 90)	0
(0.0, 45.0)	(0.0, 22.5)	(22.5, 45.0)	1
(22.5, 45.0)	(22.5, 33.75)	(33.75, 45.0)	1
(33.75, 45.0)	(33.75, 39.375)	(39.375, 45.0)	1
(39.375, 45.0)	(39.375, 42.1875)	(42.1875, 45.0)	0
(39.375, 42.1875)	(39.375, 40.7812)	(40.7812, 42.1875)	0
(39.375, 40.7812)	(39.375, 40.0781)	(40.0781, 40.7812)	0
(39.375, 40.0781)	(39.375, 39.7265)	(39.7265, 40.0781)	1
(39.7265, 40.0781)	(39.7265, 39.9023)	(39.9023, 40.0781)	1
(39.9023, 40.0781)	(39.9023, 39.9902)	(39.9902, 40.0781)	0
(39.9023, 39.9902)	(39.9023, 39.9462)	(39.9462, 39.9902)	0
(39.9023, 39.9462)	(39.9023, 39.9243)	(39.9243, 39.9462)	0
(39.9023, 39.9243)	(39.9023, 39.9133)	(39.9133, 39.9243)	1
(39.9133, 39.9243)	(39.9133, 39.9188)	(39.9188, 39.9243)	1
(39.9188, 39.9243)	(39.9188, 39.9215)	(39.9215, 39.9243)	1

5) 同理，地球经度区间是

[-180,180]，对经度116.3906进行编码的过程也类似的二分编码，过程如下：

经度范围	划分区间0	划分区间1	116.3906所属区间
(-180, 180)	(-180, 0.0)	(0.0, 180)	1
(0.0, 180)	(0.0, 90.0)	(90.0, 180)	1
(90.0, 180)	(90.0, 135.0)	(135.0, 180)	0
(90.0, 135.0)	(90.0, 112.5)	(112.5, 135.0)	1
(112.5, 135.0)	(112.5, 123.75)	(123.75, 135.0)	0
(112.5, 123.75)	(112.5, 118.125)	(118.125, 123.75)	0
(112.5, 118.125)	(112.5, 115.312)	(115.312, 118.125)	1
(115.312, 118.125)	(115.312, 116.718)	(116.718, 118.125)	0
(115.312, 116.718)	(115.312, 116.015)	(116.015, 116.718)	1
(116.015, 116.718)	(116.015, 116.367)	(116.367, 116.718)	1
(116.367, 116.718)	(116.367, 116.542)	(116.542, 116.718)	0
(116.367, 116.542)	(116.367, 116.455)	(116.455, 116.542)	0
(116.367, 116.455)	(116.367, 116.411)	(116.411, 116.455)	0
(116.367, 116.411)	(116.367, 116.389)	(116.389, 116.411)	1
(116.389, 116.411)	(116.389, 116.400)	(116.400, 116.411)	0
(116.389, 116.400)	(116.389, 116.394)	(116.394, 116.400)	0

组码 &Base32 编码

通过上述计算（图中只显示16次二分，共20次），纬度产生的编码为 1011 1000 1100 0111 1001，经度产生的编码为 1101 0010 1100 0100 0100。偶数位放经度，奇数位放纬度。因为经纬度都分别二分了20次，现在要组合成一个二十位数，这个数从左往右（纬度表示为lat，经度表示为lng）

40	39	38	37	36	...
lng[20]	lat[20]	lng[19]	lat[19]	lng[18]	...
1	1	1	0	0	...

把2串编码组合生成新串： 11100 11101 00100 01111 00000 01101 01011 00001。

这里解释下为什么前面说这些字符串分的是矩形格子，说到这里，相信大家都明白了，经度的取值范围是纬度的一倍，但都是只二分了10次，相同位数的表示，经度所能表示的长度也是纬度的一倍，在地图上就是宽宽的格子。

最后使用用0-9、b-z（去掉a, i, l, o）这32个字母进行base32编码，首先将 11100 11101 00100 01111 00000 01101 01011 00001 转成十进制 28, 29, 4, 15, 0, 13, 11, 1，十进制对应的编码就是wx4g0ec1。同理，将编码转换成经纬度的解码算法与之相反，具体不再赘述。

十进制	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
base32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
十进制	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
base32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

GeoHash Base32编码长度与精度

下表摘自维基百科：<http://en.wikipedia.org/wiki/Geohash> 可以看出，当geohash base32编码长度为8时，精度在19米左右，而当编码长度为9时，精度在2米左右，编码长度需要根据数据

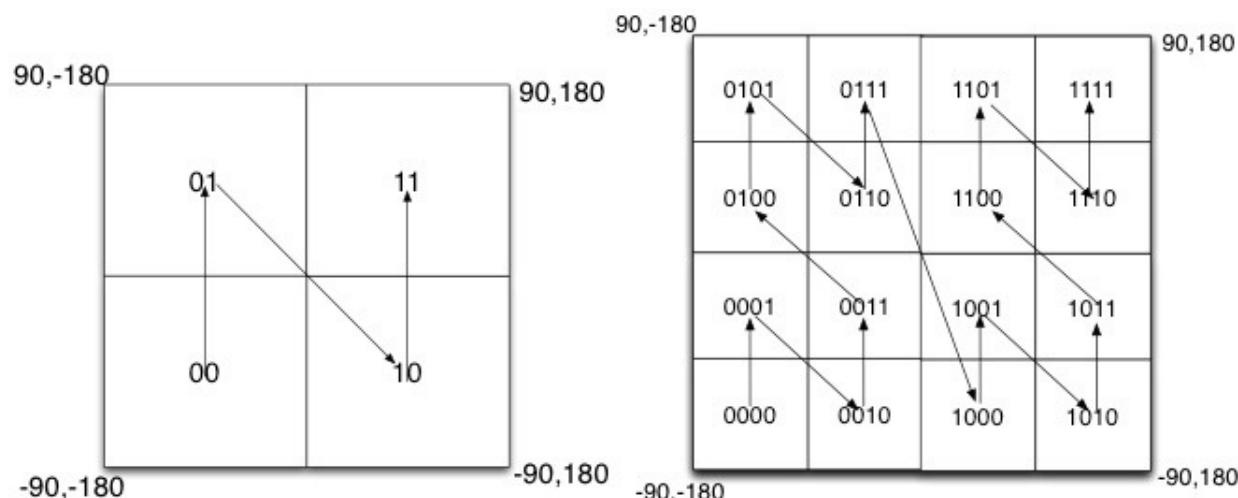
geohash length	lat bits	lng bits	lat error	lng error	km error
1	2	3	± 23	± 23	± 2500
2	5	5	± 2.8	± 5.6	± 630
3	7	8	± 0.70	± 0.7	± 78
4	10	10	± 0.087	± 0.18	± 20
5	12	13	± 0.022	± 0.022	± 2.4
6	15	15	± 0.0027	± 0.0055	± 0.61
7	17	18	± 0.00068	± 0.00068	± 0.076
8	20	20	± 0.000085	± 0.00017	± 0.019

情况进行选择。

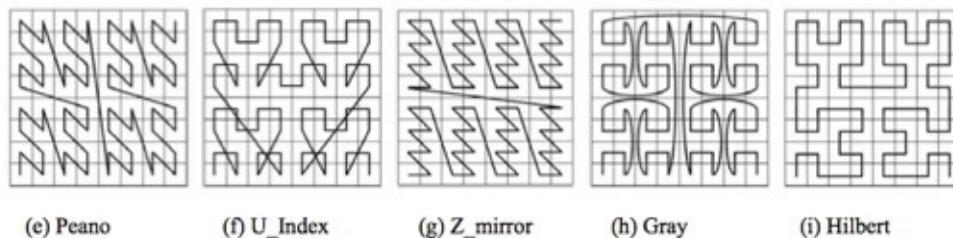
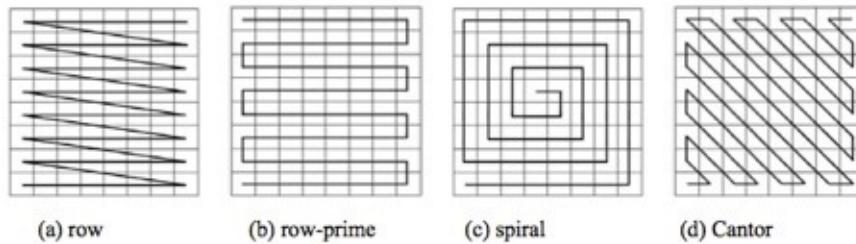
GeoHash算法依据的数学原理

算法有了，但是为什么是这样的呢，大牛们是怎么想出这个算法的呢？一切还要从数学说起，可怜我数学水平一般，下面的这些东西一知半解，直接搬过来了。

如图所示，我们将二进制编码的结果填写到空间中，当将空间划分为四块时候，编码的顺序分别是左下角00，左上角01，右下脚10，右上角11，也就是类似于Z的曲线，当我们递归的将各个块分解成更小的子块时，编码的顺序是自相似的（分形），每一个子快也形成Z曲线，这种类型的曲线被称为Peano空间填充曲线。这种类型的空间填充曲线的优点是将二维空间转换成一维曲线（事实上是分形维），对大部分而言，编码相似的距离也相近，但Peano空间填充曲线最大的缺点就是突变性，有些编码相邻但距离却相差很远，比如0111与1000，编码是相邻的，但距离相差很大。



除Peano空间填充曲线外，还有很多空间填充曲线，如图所示，其中效果公认较好是Hilbert空间填充曲线，相较于Peano曲线而言，Hilbert曲线没有较大的突变。为什么GeoHash不选择Hilbert空间填充曲线呢？可能是Peano曲线思路以及计算上比较简单吧，事实上，Peano曲线就是一种四叉树线性编码方式。



应注意的问题

1) 由于GeoHash是将区域划分为一个个规则矩形，并对每个矩形进行编码，这样在查询附近POI信息时会导致以下问题，比如红色的点是我们的位置，绿色的两个点分别是附近的两个餐馆，但是在查询的时候会发现距离较远餐馆的GeoHash编码与我们一样（因为在同一个GeoHash区域块上），而较近餐馆的GeoHash编码与我们不一致。这个问题往往产生在边界处。2)

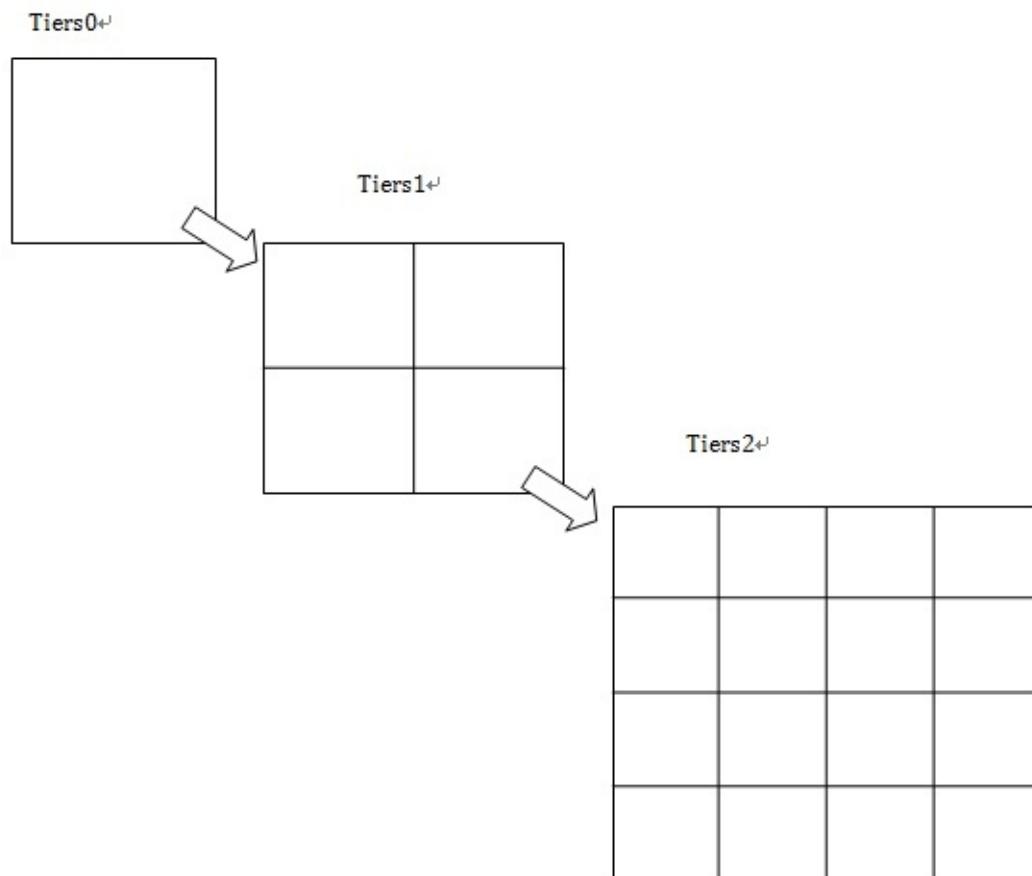


解决的思路很简单，我们查询时，除了使用定位点的GeoHash编码进行匹配外，还使用周围8个区域的GeoHash编码，这样可以避免这个问题。

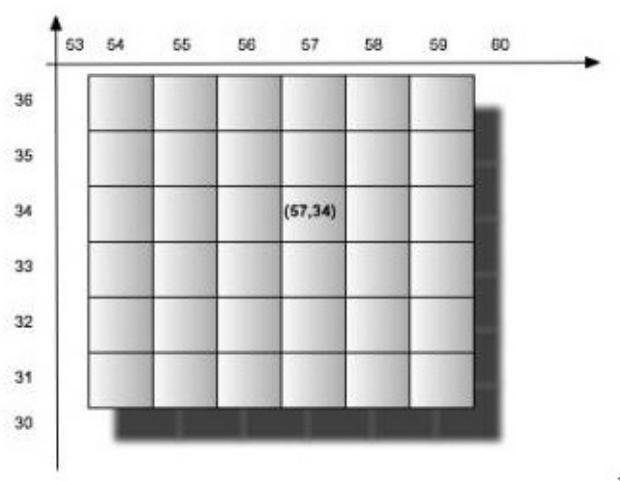
2) 我们已经知道现有的GeoHash算法使用的是Peano空间填充曲线，这种曲线会产生突变，造成了编码虽然相似但距离可能相差很大的问题，因此在查询附近餐馆时候，首先筛选GeoHash编码相似的POI点，然后进行实际距离计算。

Cartesian Tiers 算法

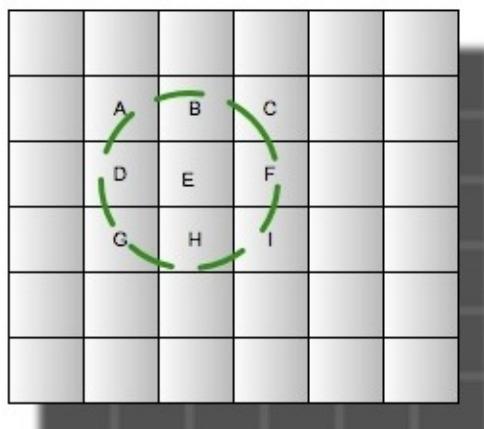
Cartesian Tiers 是笛卡尔层的意思，笛卡尔我们应该很熟悉，是一个伟大的哲学家和数学家，这里讲到笛卡尔是和笛卡尔坐标系有关。笛卡尔分层模型的思想是将经纬度转换成更大粒度的分层网格，该模型创建了很多的地理层，每一层在前一层的基础上细化切分粒度，每一个网格被分配一个ID，代表一个地理位置。



每层以2的平方递增，所以第一层为4个网格，第二层为16个，所以整个地图的经纬度将在每层的网格中体现：



那么如何构建这样的索引结构呢，其实很简单，只需要对应笛卡尔层的层数来构建域即可，一个域或坐标对应多个tiers层次。也即是tiers0->field_0, tiers1->field_1,tiers2->field_2,……，tiers19->field_19。（一般20层即可）。每个对应笛卡尔层次的域将根据当前这条记录的经纬度通过笛卡尔算法计算出归属于当前层的网格，然后将gridId（网格唯一标示）以term的方式存入索引。这样每条记录关于笛卡尔0-19的域将都会有一个gridId对应起来。但是查询的时候一般是需要查周边的地址，那么可能周边的范围超过一个网格的范围，那么实际操作过程是根据经纬度和一个距离确定出需要涉及查询的从19-0（从高往低查）若干层对应的若干网格的数据。那么一个经纬度周边地址的查询只需要如下图圆圈内的数据：

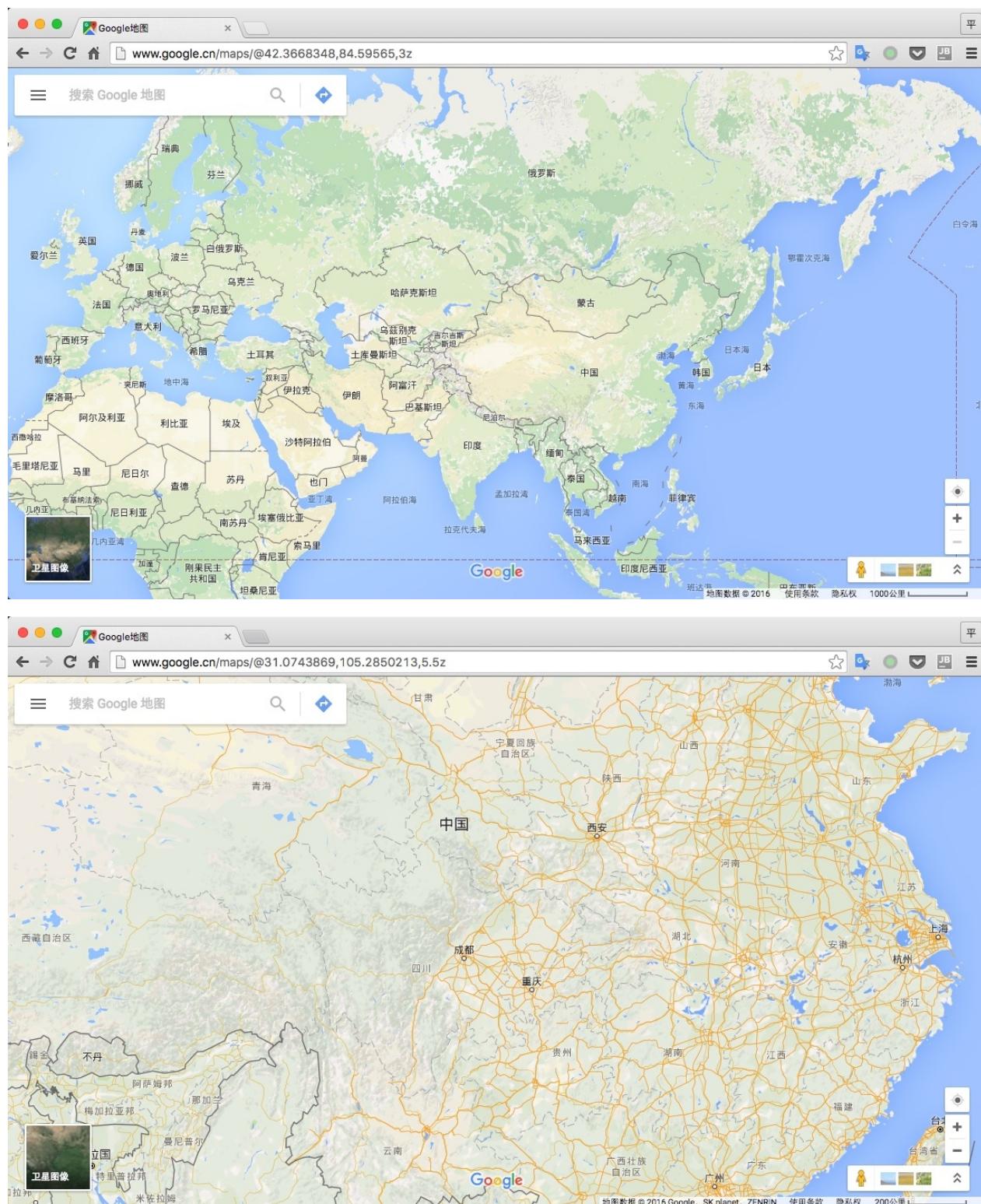


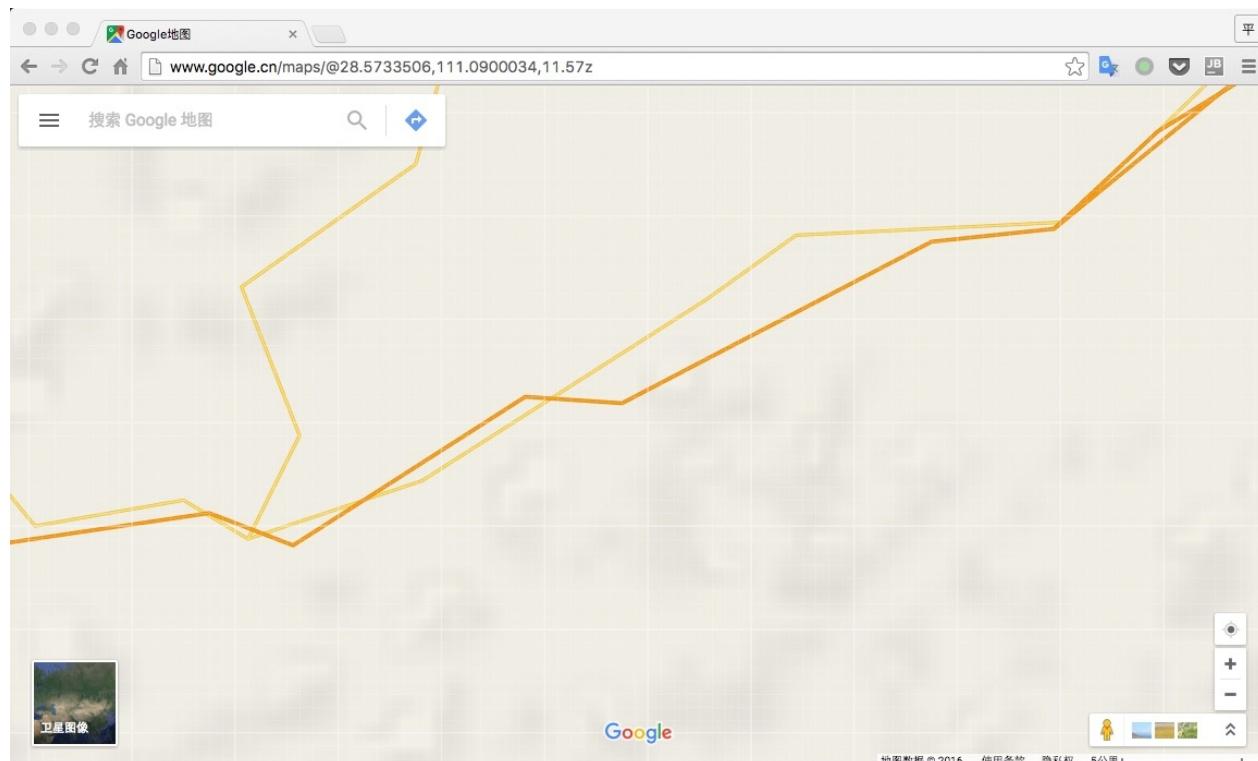
由上可知，基于Cartesian Tier的搜索步骤为：1、根据Cartesian Tier层获得坐标点的地理位置gridId 2、与系统索引gridId匹配计算 3、计算结果集与目标坐标点的距离返回特定范围内的结果集合 使用笛卡尔层，能有效缩减少过滤范围，快速定位坐标点。

笛卡尔层在Lucene中对空间地理位置查询最大的用处在查找周边地址的时候有效的减少查询量，即将查询量可以控制在分层后最小的网格中的若干docId。

这个思想的另外一个应用在地图的展现上，整个世界地图相当于是第一层，只有各个国家名字；我们对某个块进行放大，例如中国这一块，会加载省份，这相当于进入第二层；对某个省份进行放大会继续加载城市，这相当于第三层。当然实际的分层应该比较多的，分层越多缩放的时候当然越流畅。

3.3 空间搜索原理及相关算法





在数据还没有加载过来的时候，我们可以很容易看到地图中划分的网格。

参考资料

1. [Solr空间搜索原理分析与实践](#)
2. [使用 Apache Lucene 和 Solr 进行位置感知搜索](#)
3. [GeoHash核心原理解析](#)

前面讲了空间搜索的两个常用原理，应该还算是比较容易理解的，毕竟只是算法，并不需要我们来实现，我们通过简单地配置就可以用上Solr的空间搜索的功能。下面就来讲讲如何配置空间搜索。

前面说的GeoHash和Cartesian Tiers原理，在Solr中对应的是实现分别是GeohashPrefixTree类和QuadPrefixTree类，通过这两个类在数据索引阶段按照两个不同的原理进行索引的建立。Solr中默认是使用GeohashPrefixTree的方式。

索引的构建

通过前面的[Solr5配置文件参数解析](#)一文，我们应该了解到建立索引，首先我们以该先配置field，这里也一样，要想对空间信息（即坐标）进行索引，首先我们应该先配置坐标的field。

配置项和参数说明

需要配置 `fieldType` 和 `field` 两项。

```
<fieldType name="location_rpt" class="solr.SpatialRecursivePrefixTreeFieldType" spatialContextFactory="com.spatial4j.core.context.jts.JtsSpatialContextFactory" distErrPct="0.025" maxDistErr="0.000009" units="degrees"/>

<field name="sight_coordinate" type="location_rpt" indexed="true" stored="true" multiValued="false" />
```

这些配置项的一些属性，下面做一些说明：`SpatialRecursivePrefixTreeFieldType`

用于深度遍历前缀树的FieldType，主要用于获得基于Lucene中的RecursivePrefixTreeStrategy。

`JtsSpatialContextFactory`

当有Polygon多边形或者linestrings线段时，会使用jts(需要把jts.jar放到solr服务的lib下)，而基本形状使用SpatialContext(spatial4j的类)。

`distErrPct`

定义非Point图形的精度，范围在0-0.5之间。该值决定了非Point的图形索引或查询时的level(如geohash模式时就是geohash编码的长度)。当为0时取maxLevels，即精度最大，精度越大将花费更多的空间和时间去建索引。

`geo`

默认为true，值为true的情况下坐标基于球面坐标系，采用Geohash的方式；值为false的情况下坐标基于2D平面的坐标系，采用Euclidean/Cartesian的方式。

worldBounds

世界坐标值：“minX minY maxX maxY”。 geo=true 即 geohash 模式时，该值默认为“-180 -90 180 90”。 geo=false 即 quad 时，该值为 Java double 类型的正负边界，此时需要指定该值，设置成“-180 -90 180 90”。

distCalculator

设置距离计算算法， geo=true 默认是 haversine， geo=false 默认是 cartesian(笛卡尔计算方式)，值可以为 "lawOfCosines"(余弦定理), "vincentySphere"(文森特球面公式) 或 "cartesian^2"。

prefixTree

Solr 将地球映射为网格，prefixTree 定义了网格的实现方式，每个网格在下一层中可以分解成多个子网格， geo=true prefixTree 只能取 GeoHash， geo=false prefixTree 可取 quad(quadTree 一种四分树地理位置索引，对应笛卡尔分层策略)

maxDistErr/maxLevels

maxDistErr 定义了索引数据的最高层 maxLevels，上述定义为 0.000009，根据 GeohashUtils.lookupHashLenForWidthHeight(0.000009, 0.000009) 算出编码长度为 11 位，精度在 1 米左右，直接决定了 Point 索引的 term 数。 maxLevels 优先级高于 maxDistErr，即有 maxLevels 的话 maxDistErr 失效。详见 SpatialPrefixTreeFactory.init() 方法。不过一般使用 maxDistErr。

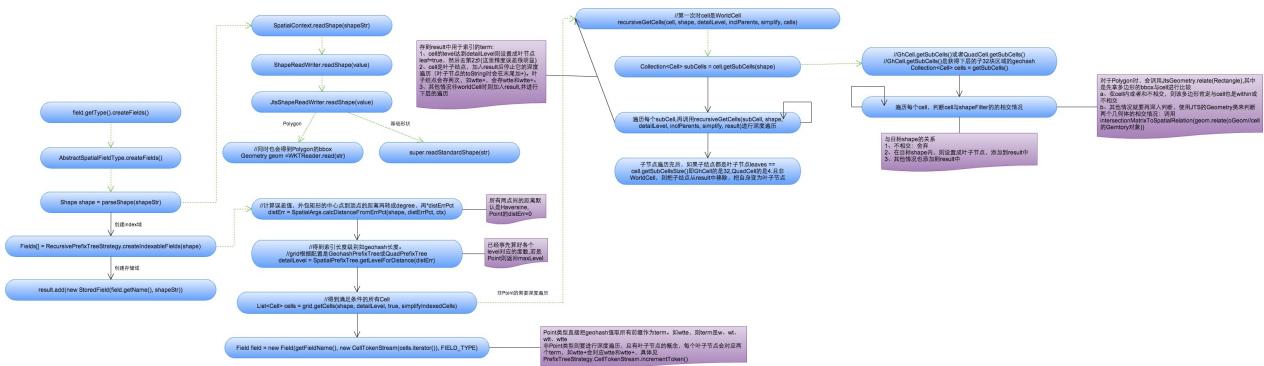
units

单位是 degrees，不适用于 geofilt, bbox, or geodist (单位为 km)

用代码构建索引

```
doc.setField("poi_location_p",      "32.52162,120.31778") //point类型
//或者
doc.setField("poi_location_p",      "POLYGON((120.35330414772034
31.58268495951037,120.35190939903259      31.57923921490961,120.35330414772034
31.58268495951037))" ) //多边形类型
```

构建流程：



下面主要说明Point类型的term创建过程。1、将空间索引域的shapeStr解析成相应的Shape（这里指Point，复杂Shape如Polygon要使用JTS中的WTKReader来解析）。2、创建索引域，具体过程参考 org.apache.lucene.spatial.prefix.RecursivePrefixTreeStrategy 中的 createIndexableFields 方法。a、根据distErrPct字段，计算距离的误差值，对于Point来说默认为0（而对于非Point类型来说，是通过外包矩形中心点到矩形顶点的距离再乘以distErrPct来计算误差值的）

```

double distErr = SpatialArgs.calcDistanceFromErrPct(shape, distErrPct, ctx);

public static double calcDistanceFromErrPct(Shape shape, double distErrPct, SpatialContext ctx) {
    if (distErrPct < 0 || distErrPct > 0.5) {
        throw new IllegalArgumentException("distErrPct " + distErrPct + " must be between [0 to 0.5]");
    }
    if (distErrPct == 0 || shape instanceof Point) {
        return 0;
    }
    Rectangle bbox = shape.getBoundingBox();
    //Compute the distance from the center to a corner. Because the distance
    //to a bottom corner vs a top corner can vary in a geospatial scenario,
    //take the closest one (greater precision).
    Point ctr = bbox.getCenter();
    double y = (ctr.getY() >= 0 ? bbox.getMaxY() : bbox.getMinY());
    double diagonalDist = ctx.getDistCalc().distance(ctr, bbox.getMaxX(), y);
    return diagonalDist * distErrPct;
}

```

b、根据上述计算出的误差值，得到索引的geohash编码长度，对于Point类型来说值为maxLevels。

```

public int getLevelForDistance(double dist) {
    if (dist == 0)
        return maxLevels;//short circuit
    final int level = GeohashUtils.lookupHashLenForWidthHeight(dist, dist);
    return Math.max(Math.min(level, maxLevels), 1);
}

```

c、根据编码长度得到满足所有条件的cells（每个cell表示一个前缀值），并将Cells都放在CellTokenStream中，同时构建索引域。Point类型每个Cell表示geohash的一个前缀值。

```

public List<Cell> getCells(Point p, int detailLevel, boolean inclParents){
    Cell cell = getCell(p, detailLevel);
    if (!inclParents) {
        return Collections.singletonList(cell);
    }

    String endToken = cell.getTokenString();
    assert endToken.length() == detailLevel;
    List<Cell> cells = new ArrayList<Cell>(detailLevel);
    for (int i = 1; i < detailLevel; i++) {
        cells.add(getCell(endToken.substring(0, i)));
    }
    cells.add(cell);
    return cells;
}

Field field = new Field(getFieldName(),
    new CellTokenStream(cells.iterator()), FIELD_TYPE);

```

3、构建存取域存储索引

```

if (field.stored()) {
    if (shapeStr == null)
        shapeStr = shapeToString(shape);
    result.add(new StoredField(field.getName(), shapeStr));
}

```

4、结果如经纬度41.79452,123.41555，对应的geohash为wxrvb2kqexu（maxLevels=11），则其对应的term有11个（如w、wx、wxr、wxrv...）。

查询

在这里之前，请确保已经看过[查询方法及参数说明//TODO](#)这篇文章，对查询常用到的参数比较熟悉。

查询语法实例：

```
q={!geofilt pt=45.15,-93.85 sfield=poi_location_p d=5 score=distance}
q={!bbox pt=45.15,-93.85 sfield=poi_location_p d=5 score=distance}
q=poi_location_p:"Intersects(-74.093 41.042 -69.347 44.558)" //a bounding box (not in
WKT)
q=poi_location_p:"Intersects(POLYGON((-10 30, -40 40, -10 -20, 40 20, 0 0, -10 30)))"
//a WKT example
```

空间搜索涉及到的特有的查询参数有： **sfield**：指定坐标索引字段，如**sfield=geo** **pt**：坐标点，如**pt=54.729696,-98.525391**

其中有几种常见的**Solr**支持的几何操作：**WITHIN**：在内部 **CONTAINS**：包含关系
DISJOINT：不相交 **Intersects**：相交（存在交集）

我的这个项目实现了范围搜索和图形搜索，其中范围搜索用的是圆形范围，图形搜索用的是矩形边界搜索。更加详细的内容请看美团的文章。

参考资料

1. [Solr空间搜索原理分析与实践 - 美团](#)

一般存储数据都会用到数据库，之前十几年关系型数据库大行其道，现在非关系性数据库（**NoSql**）如日中天，随着数据越来越越来越多，人们发现关系型数据库的性能已经不能满足需要，经历了一番挣扎，从主-从（读-写）分离，到分库分表，虽然维持了一段时间，但是数据量很快就上来了，于是**NoSql**越来越显示出其在大数据时代的价值。

咳咳，不过这篇文章讲的却是从最流行的关系型数据库中导入数据到**Solr**，没办法，笔者还没用过**NoSql**，所以还是老老实实讲**Mysql**，哈哈。

导入需要的jar包

做过数据库开发的童鞋都知道，要想用数据可就需要连接数据库的接口，java上叫**JDBC**(Java Data Base Connectivity)，别的语言也有类似的接口。

那么，这次我们需要两个jar包，分别是 `solr-dataimporthandler-5.5.0.jar` 和 `mysql-connector-java-5.1.38-bin.jar`，前一个可以在**Solr**的解压目录下的**dist**目录中获取，后一个我想大家都可以找到的。

将这两个jar包复制到 `$Solr.Install.Dir/server/solr-webapp/webapp/WEB-INF/lib` 这个目录下。

配置导入设置

1. 默认**dataimport**功能在**Solr5**中是禁用的，需要在 `$SolrHome/conf/solrconfig.xml` 中添加如下配置开启数据导入功能：

```
<!-- Data import from mysql 要放在<config></config>中哦-->
<requestHandler name="/dataimport" class="org.apache.solr.handler.dataimport.DataImportHandler">
  <lst name="defaults">
    <str name="config">data-config.xml</str>
  </lst>
</requestHandler>
```

2. 因为前面定义了导入的配置文件是 `data-config.xml`，所以在**solrconfig.xml**同级目录下新建这个文件，贴出我的配置，内容如下：

```

<?xml version="1.0" encoding="UTF-8" ?>
<dataConfig>
<dataSource name="fromMysql"
    type="JdbcDataSource"
    driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/tripsearch"
    user="root"
    password="root"/>
<document>
    <entity name="sight" query="SELECT * FROM sight" transformer="RegexTransformer">
        <field column="sight_id" name="sight_id"/>
        <field column="sight_name" name="sight_name"/>
        <field column="sight_score_ctrip" name="sight_score_ctrip"/>
        <field column="sight_intro" name="sight_intro"/>
        <field column="sight_address" name="sight_address"/>
        <field column="sight_coordinate" name="sight_coordinate"/>
        <field column="sight_type" name="sight_type" splitBy=","/>
        <field column="pageurl" name="pageurl"/>
    </entity>
</document>

</dataConfig>

```

其中fromMysql为数据源自定义名称，随便取，没什么约束，type这是固定值，表示JDBC数据源，后面的driver表示JDBC驱动类，这跟你使用的数据库有关，url即JDBC连接URL，后面的user，password分别表示链接数据库的账号密码，下面的entity映射有点类似hibernate的mapping映射，column即数据库表的列名称，name即schema.xml中定义的域名称。

还有些设置项、参数不知道什么意思，后面后会说的。现在只要清楚field这个标签是设置数据库表中的列（column）和Solr中的字段的映射关系的。

另外：Solr中field还没有配置的，请先阅读[3.1 Solr5配置文件参数解析](#)

进行数据导入

1. 重启Solr，进入管理页面，选中trip这个Core，进入Dataimport这个选项。如果一切正常会出现如下图所示的界面。

The screenshot shows the Solr Admin interface at localhost:8983/solr/index.html#/trip/dataimport//dataimport. The left sidebar shows the 'trip' core selected. The main panel is titled '/dataimport' and displays the following configuration:

- Command:** full-import
- Options:** Verbose (unchecked), Clean (checked), Commit (checked), Optimize (unchecked), Debug (unchecked).
- Entity:** dropdown set to 'trip'.
- Start, Rows:** Start is 0, Rows is 10.
- Custom Parameters:** key1=val1&key2=val2
- Buttons:** Execute, Refresh Status, Auto-Refresh Status (unchecked).

The right panel shows the XML configuration code:

```

<?xml version="1.0" encoding="UTF-8" ?>
<dataConfig>
<dataSource type="JdbcDataSource">
    driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/tripsearch"
    user="root"
    password="root"/>
<document>
    <entity name="sight" query="SELECT * FROM sight" transformer="RegexTransformer">
        <field column="sight_id" name="sight_id"/>
        <field column="sight_name" name="sight_name"/>
        <field column="sight_score_ctrip" name="sight_score_ctrip"/>
        <field column="sight_intro" name="sight_intro"/>
        <field column="sight_address" name="sight_address"/>
        <field column="sight_coordinate" name="sight_coordinate"/>
        <field column="sight_type" name="sight_type" splitBy=","/>
        <field column="pageurl" name="pageurl"/>
        <field column="sight_place_1" name="sight_place_1"/>
        <field column="sight_place_2" name="sight_place_2"/>
        <field column="sight_place_3" name="sight_place_3"/>
        <field column="sight_place_4" name="sight_place_4"/>
    </entity>
</document>

```

右面的那些代码是你点击Configuration后出现的，是 `data-config.xml` 文件中的内容，也是应该首先检查的地方，如果没有这些配置信息，说明数据库导入的配置文件没生效，是 `solrconfig.xml` 文件中开启导入功能的地方出错。

2. 开始导入，要注意command参数，它有两个选项，如下图：

Command

- full-import
- full-import
- delta-import

Clean

Commit

Optimize

Debug

Entity

Start, Rows

0

Custom Parameters

key1=val1&key2=val2

Execute Refresh Status

Auto-Refresh Status

full-import: 全量导入，它会覆盖原有的索引
 delta-import: 即增量导入，它会在原有索引的基础上追加

下面的几个多选框含义解释如下：
verbose: 这个选项设为true的话，会打印导入的一些中间过程的详细信息，有利于调试以及了解内部操作细节
clean: 表示是否在导入数据创建索引之前先清空掉原有的索引
commit: 表示是否立即提交索引
optimize: 表示是否优化索引
debug: 表示是否开启调试模式

3. 然后选择需要导入的Entity, 点击Execute按钮开始执行数据导入操作，如图：

3.5 导入Mysql数据到Solr中

/dataimport

Command
full-import

Verbose
 Clean
 Commit
 Optimize
 Debug

Entity
user

Start, Rows
0 | a

Custom Parameters
key1=val1&key2=val2

Execute Refresh Status

Auto-Refresh Status

正常的话就开始进行导入了，如下图

/dataimport

Command
full-import

Verbose
 Clean
 Commit
 Optimize
 Debug

Entity
user

Start, Rows
0 | a

Custom Parameters
key1=val1&key2=val2

Execute Refresh Status

Task-Header: 2013-07-30 13:37:39
Indexing ... 表示正在创建索引中..... Requests: 0, Fetched: 0, Skipped: 0, Processed: 0

Abort Import

Raw Status-Output Configuration

Debug-Mode Reload

Documentation Issue Tracker IRC Channel Community forum Solr Query Syntax

我们可以通过 Refresh Status 这个按钮刷新状态，如果出现错误或者Fetched一直是0，那就表明有问题了，你要查看日志进行检查。如果导入成功，就会看到下图所示的情况：

3.5 导入Mysql数据到Solr中

The screenshot shows the Solr Data Import Handler interface. On the left, there's a configuration panel with fields like 'Command' (set to 'full-import'), 'Verbose' (unchecked), 'Clean' (checked), 'Commit' (checked), 'Optimize' (unchecked), 'Debug' (unchecked), 'Entity' (empty dropdown), 'Start, Rows' (0 to 10), 'Custom Parameters' (key1=val1&key2=val2), and buttons for 'Execute' and 'Refresh Status'. On the right, a green status bar displays: 'Last Update: 22:08:53', 'Indexing completed. Added/Updated: 2 documents. Deleted 0 documents.', 'Requests: 1, Fetched: 2, Skipped: 0, Processed: 2', and 'Started: about a minute ago'. Below the status bar are sections for 'Raw Status-Output' and 'Configuration', and a 'Debug' button. A red arrow points from the text '绿色表示导入成功' to the green success message in the status bar.

绿色表示导入成功
Fetched表示提取即成功从数据库提取到2条数据
Processed表示成功创建了2个Document

查看OverView菜单，会看到文档信息。

The screenshot shows the Solr Admin OverView page for the 'trip' core. The left sidebar has a navigation menu with items like Dashboard, Logging, Core Admin, Java Properties, Thread Dump, and a dropdown for 'trip'. Under 'trip', there are links for Overview, Analysis, Dataimport, Documents, Files, Ping, Plugins / Stats, Query, Replication, and Schema. The main content area has tabs for Statistics, Instance, Replication (Master), and Healthcheck. The Statistics tab shows: Last Modified: about a month ago, Num Docs: 8685 (highlighted with a red box), Max Doc: 8685, Heap Memory: -1, Usage: 112, Deleted Docs: 0, Version: 112, Segment 1, Count: 0, Optimized: ✓, and Current: ✓. The Instance tab shows: CWD: /Users/Payne/Workspace/GraduateProject/Solr/solr-5.5.0/server, Instance: /Users/Payne/Workspace/GraduateProject/Solr/solr-5.5.0/server/solr/trip, Data: /Users/Payne/Workspace/GitHub/trip-search/SolrHome/trip/data, Index: /Users/Payne/Workspace/GitHub/trip-search/SolrHome/trip/data/index, and Impl: org.apache.solr.core.NRTCachingDirectoryFactory. The Replication (Master) tab shows two rows: Master (Searching) with Version 1459939405413, Gen 29, Size 18.17 MB, and Master (Replicable) with Version -. The Healthcheck tab states: 'Ping request handler is not configured with a healthcheck file.' At the bottom, there are links for Documentation, Issue Tracker, IRC Channel, Community forum, and Solr Query Syntax.

在查询中点击 Execute Query 按钮,就能看到我们导进去并建好索引的信息,更具体的查询用法后面会讲到,下面是默认的查询,显示文档的所有信息。

The screenshot shows the Solr Admin interface with the URL `localhost:8983/solr/trip/index.html#/trip/query`. The left sidebar has a 'trip' section selected. The main area contains a search form with fields like Request-Handler (qt), q (*.*), fq, sort, start, rows (10), fl, df, Raw Query Parameters (key1=val1&key2=val2), wt (json), indent, and debugQuery. A blue 'Execute Query' button is at the bottom. To the right is the search results panel, which displays a JSON response. The response includes a 'responseHeader' with status 0, QTime 38, and params (q: *, indent: on, wt: json, _id: 146364997719). The 'response' object has 'numFound': 6685 and 'start': 0. It lists two documents. Document 1 is about the Tiananmen Square, and Document 2 is about the Forbidden City.

```

{
  "responseHeader": {
    "status": 0,
    "QTime": 38,
    "params": {
      "q": "*.*",
      "indent": "on",
      "wt": "json",
      "_id": "146364997719"
    }
  },
  "response": {
    "numFound": 6685,
    "start": 0,
    "docs": [
      {
        "sight_intro": "天安门广场位于北京的中轴线上，是共和国的中心广场，天安门城楼、人民大会堂、国家博物馆、人民英雄纪念碑等众多国家象征都坐落于北京，中国。",
        "sight_id": 228,
        "sight_place_3": "天安门广场",
        "sight_place_to": ["天安门广场", "北京", "中国"],
        "sight_place_2": "北京",
        "sight_place_1": "中国",
        "sight_score_ctrip": 0.0,
        "pageurl": "http://you.ctrip.com/sight/beijing1/228.html",
        "sight_name": "天安门广场",
        "sight_coordinate": "39.9054881306264,116.397633984685",
        "sight_type": ["地标", "广场"],
        "sight_type_to": ["地标", "广场"],
        "_version_": 1530857414972145664
      },
      {
        "sight_intro": "绝大多数来北京的游客，尤其是初游者，都会把故宫当作必去之处。故宫又称紫禁城，是明、清两代的皇宫，也是古老中国的标志和象征。",
        "sight_id": 229,
        "sight_place_3": "故宫",
        "sight_place_to": ["故宫", "北京", "中国"],
        "sight_place_2": "北京",
        "sight_place_1": "中国",
        "sight_score_ctrip": 10.0,
        "pageurl": "http://you.ctrip.com/sight/beijing1/229.html",
        "sight_name": "故宫"
      }
    ]
  }
}

```

较为复杂的字段映射

数据库单表多个字段到solr多值字段

例如我数据库一个表中有关于地址的几个字段，分别是国家、省份、地区这样的字段

(`sight_place_1`, `sight_place_2`, `sight_place_3..`) 我需要把这个字段放到solr中一个多值字段`sight_place`中。

题外话：对于关系型数据库不应该这样建表的，至少要满足第二范式，但这里用mysql只是为了存数据。

其实对于这个需求比较容易实现，修改 `data-config.xml` 文件中的entity就行了，配置如下

```

<field column="sight_place_1" name="sight_place"/>
<field column="sight_place_2" name="sight_place"/>
<field column="sight_place_3" name="sight_place"/>
<field column="sight_place_4" name="sight_place"/>
<field column="sight_place_5" name="sight_place"/>

```

数据库单表单字段到solr多值字段

前提：数据库中的单字段中的数据包含多值信息，并且用分隔符分开。例如数据库中 sight_type 字段的值是

sight_name	sight_type
故宫	古迹,世界文化遗产,历史建筑,博物馆

```
<entity name="sight" query="SELECT * FROM sight" transformer="RegexTransformer">
    <field column="sight_type" name="sight_type" splitBy=","/>
</entity>
```

这样，导入到sight_type 中的数据就是多值的

```
sight_type{
    "博物馆",
    "历史建筑",
    "世界文化遗产",
    "古迹"
}
```

特别注意： entity 的属性 transformer 要为 RegexTransformer ，字段映射 field 属性 splitBy 设置为你的分隔符

数据库中多表联查到solr多值字段

1. 对于熟悉sql语句的人来讲，通过相对复杂的查询语句可以查到任何条件的结果，例如下面要说的 group_concat

```
select sight_place_1, GROUP_CONCAT(sight_place_2) as subplace FROM sight GROUP BY sight_place_1
```

这条查询的结果是

sight_place_1	subplace
中国	北京,台湾,江苏,香港,宁夏,陕西

这样就回到了上面的方法了。当然这并不是多表，只是个简单的实例，你可以用更复杂的sql语句实现多表联查。

2. 另外solr也提供了子查询功能，例如下面的实例，更复杂的可以查看官方Wiki

```
<entity name="comment" pk="id" query="SELECT * FROM comment">
    <field column="blogpost_id" name="blogpost_id"/>
    <field column="comment_text" name="comment_text" />
    <entity name="comment_tags" pk="comment_id" query="SELECT * FROM comment_tags
WHERE comment_id='${comment.id}'">
        <field column="tag" name="tag" />
    </entity>
</entity>
```

一些问题

不可行：多值字段按照下标取值 本来我想着取多值字段的sight_place的第二个值，就是省份，但最后发现这并不可行，solr中的多值应该就是个无序列表，但却可以发现在查询结果中，先插入的在下面，后插入的在上面，也不是真的无序。但没找到有序访问的方法。下面是关于这个问题的链接

[Query specifically indexed value in multivalued field](#)

[Query a specific index value in a multivalue field in Solr](#)

答案中建议我们可以用动态域（dynamic field）或者前缀的方式解决。

参考链接

1. [Uploading Structured Data Store Data with the Data Import Handler - solr5 Wiki](#)
2. [Data Import Request Handler - solr4 Wiki](#)
3. [Importing multi-valued field into Solr from MySQL using Solr Data Import Handler](#)
4. [solr5.3.1从mysql导入索引](#)
5. [跟益达学Solr5之从MySQL数据库导入数据并索引](#)

接受查询部分

本章将介绍相关查询的内容，利用Solr提供的Restful API很容易实现各种查询，我主要利用Solrj来提供对外的搜索接口，并利用Ionic开发了跨平台的客户端。

Solr默认有三种查询解析器（Query Parser）：

- Standard Query Parser
- DisMax Query Parser
- Extended DisMax Query Parser (eDisMax)

第一种是标准的Parser，最后一种是最强大的。

本文中所提到的参数并不能包含Solr所有参数，具体的使用和更详细的参数请参考官方文档。

常用参数

- defType 选择查询解析器类型，例如dismax, edismax
- q 主查询参数 (field_name:value)
- sort 排序，例如score desc, price asc
- start 起始的数据偏移offset，用于分页
- raws 一次返回的数量，用于分页
- fq filter query 返回结果的过滤查询
- fl fields to list 返回的字段 (*, score)
- debug 返回调试信息，debug=timing, debug=results
- timeAllowed 超时时间
- wt response writer返回的响应格式

下面是DisMax Parser可以使用的：

- qf query fields，指定查询的字段，指定solr从哪些field中搜索，没有值的时候使用df
- mm 最小匹配比例
- pf phrase fields
- ps phrase slop
- qs query phrase slop

结果高亮：

- hl 是否高亮，如hl=true
- hl.fl 高亮field ,hl.fl=Name,SKU
- hl.snippets 默认是1,这里设置为3个片段
- hl.simple.pre 高亮前面的格式
- hl.simple.post 高亮后面的格式

facet统计：

- facet 是否启动统计
- facet.field 统计field

Solr运算符

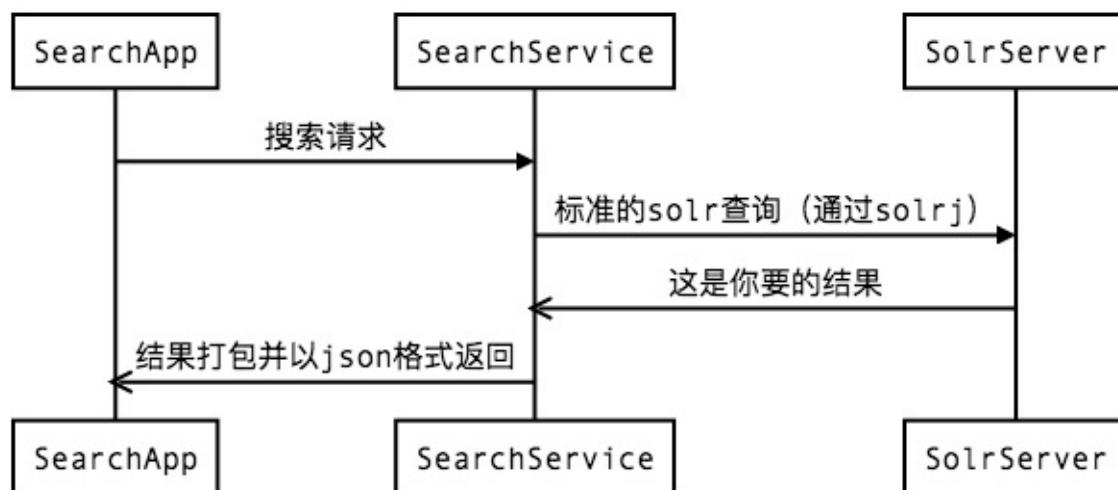
- ： 指定字段查指定值，如返回所有值:
- ？ 表示单个任意字符的通配
- * 表示多个任意字符的通配（不能在检索的项开始使用 * 或者 ? 符号）
- 表示模糊检索，如检索拼写类似于"roam"的项这样写: roam~ 将找到形如foam和roams的单词； roam~0.8 ，检索返回相似度在0.8以上的记录。
- 邻近检索，如检索相隔10个单词的"apache"和"jakarta"， "jakarta apache"~10
- ^ 控制相关度检索，如检索"jakarta apache"，同时希望去让"jakarta"的相关度更加好，那么在其后加上 ^ 符号和增量值，即 jakarta^4 apache
- 布尔操作符 AND 、 ||
- 布尔操作符 OR 、 &&
- 布尔操作符 NOT 、 ! 、 - （排除操作符不能单独与项使用构成查询）
- + 存在操作符，要求符号"+”后的项必须在文档相应的域中存在
- () 用于构成子查询
- [] 包含范围检索，如检索某时间段记录，包含头尾， date:[200707 TO 200710]
- { } 不包含范围检索，如检索某时间段记录，不包含头尾 date:{200707 TO 200710}
- / 转义操作符，特殊字符包括 + - && || ! () { } [] ^ " ~ * ? : /

注意：“+”和“-”表示对单个查询单元的修饰，and 、or 、 not 是对两个查询单元是否做交集或者做差集还是取反的操作的符号。

比如:AB:china +AB:america ,表示的是AB:china忽略不计可有可无，必须满足第二个条件才是对的,而不是你所认为的必须满足这两个搜索条件

如果输入:AB:china AND AB:america ,解析出来的结果是两个条件同时满足，即+AB:china AND +AB:america或+AB:china +AB:america

这篇文章主要是关于solr和普通用户之间的桥梁 SearchService ,简单了解下整过工作流程.

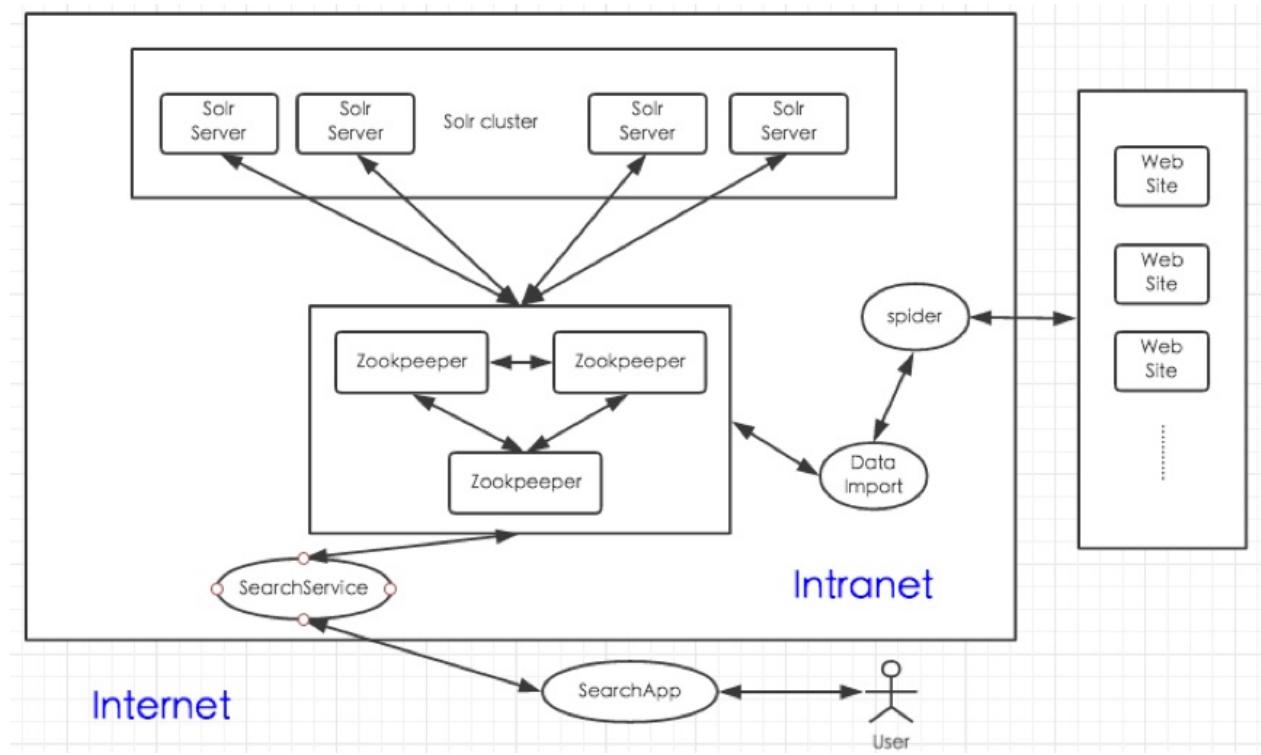


为何要在**Solr**和用户之间加一层？

因为**solr**提供了url方式（[REST风格](#)）的API来进行增删改查，因此如果不加安全策略，别人在查询的同时可以修改你的数据，这是绝对不允许的。但是把**solr**的服务端口开放然后加安全策略的方式是不科学的，这个安全策略难以配置，并且漏洞很多，所以我们通过建立一个独立的**web**服务器来提供对外服务，**solr**服务器只对内网开放，这样就比较安全并容易控制。

在这里，我不直接提供传统意义上的**web**服务，而是采用 **WebService** 的模式，参照[REST风格](#)风格提供API服务。优点是可以应对各种不同平台。

这样做的另一个好处是能轻松应对**SolrCloud**的扩展和并发量的剧增，如果后期并发增加，可以扩展 **SearchService** 到多台**web**服务器，然后通过**nginx**做反向代理和负载均衡，将客户端的请求分散到不同的**web**服务器上。



用 **Servlet** 提供服务

- 代码请参考 [SearchService-Github](#).
- 开发IDE为 [IntelliJ IDEA 15](#)，用gradle管理，使用Jetty插件。
- 项目依赖(项目根目录的build.gradle)：

```
dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    testCompile 'org.slf4j:slf4j-simple:1.7.20'
    providedCompile 'javax.servlet:javax.servlet-api:3.0.1'
    compile 'org.apache.solr:solr-solrj:5.5.0'
    compile 'com.google.code.gson:gson:2.6.2'
    compile 'com.squareup.okhttp:okhttp:2.7.5'
}
```

定义API参数

参数	意义（取值范围）	默认值
keyword	关键词	null
point	坐标点	null
distance	距离	100KM
bound_type	界限方式{geofilt,bbox,linestring,polygon}	null
boundary	范围	null
sort_order	排序方式 {score,distance}	null
sight_type	景点类型	null
place	行政区划	null
page	页数	1
rows	每页记录条数	10
query_type	查询方式{near,key,map}	near

Solrj的使用

1.先来构建出来 SolrClient

代码位置：github中trip-search项目

中 SearchService/src/main/java/com/fliaping/trip/search/MySolrClient.java

```
public class MySolrClient {
    private static final String urlString = "http://localhost:8983/solr/trip";
    private static HttpSolrClient solrClient ;
    private static MySolrClient mySolrClient ;
    private MySolrClient(){}
    public static synchronized MySolrClient getInstance(){
        if (mySolrClient == null) mySolrClient = new MySolrClient();
        return mySolrClient;
    }
    public static HttpSolrClient getSolrClient(){
        if (solrClient == null){
            solrClient = new HttpSolrClient(urlString);
            solrClient.setSoTimeout(1000); // socket read timeout
            solrClient.setAllowCompression(true); // allow Compression
        }
        return solrClient;
    }
}
```

2. 设置查询参数，这里的内容较多，方法也各不相同，详细请参考 [SolrJ Tutorial](#) 和官方 [Using SolrJ](#)

示例代码：

代码位置：github中trip-search项目

中 SearchService/src/main/java/com/fliaping/trip/search/DoQuery.java

```
solrQuery.setStart(start)
    .setRows(rows)
    .setQuery("{!geofilt}")
    .set("indent","true")
    .set("pt","22.5347168,113.971228") //当前坐标
    .set("sfield",Setting.sfield) //位置域
    .set("fl","_dist_:geodist(),*"); //返回结果中添加_dist_字段(到当前坐标的距离)
```

3. 查询并得到结果

```
QueryResponse queryResponse = solrClient.query(solrQuery);
```

4. 对结果进行自定义包装

代码位置：github中trip-search项目

中 SearchService/src/main/java/com/fliaping/trip/search/DoQuery.java

```
/**
 * 对查询结果包装成为客户端可用的格式
 * @param queryResponse solr的请求结果
 * @param hasFacet 是否需要facet
 * @return
 */
private String wrapResult(QueryResponse queryResponse, boolean hasFacet){
    SightList sightList = new SightList();

    if(hasFacet){
        List<SightFacet> sightFacets = new ArrayList<SightFacet>();
        //整理facet数据
        List facet = queryResponse.getFacetFields();
        for (int i = 0; i < facet.size(); i++) {
            //取得每个facet的信息
            FacetField ff = (FacetField) facet.get(i);
            //System.out.println("name:"+ff.getName()+" valuecount:"+ff.getValueCount());
        }

        SightFacet sightFacet = new SightFacet(); //facet对象
        sightFacet.setFacetName(ff.getName()); //设置facet的field名字
        List<SightFacet.Item> items = new ArrayList<SightFacet.Item>();
    }
}
```

```

    //取得每个facet中的每个item
    List<FacetField.Count> countList = ff.getValues();
    for (int j = 0; j < countList.size(); j++) {
        FacetField.Count count = countList.get(j);
        if (count.getCount() <= 0) continue; //facet中没有数据的,不加入结果集
        //System.out.println("name:"+count.getName()+" count:"+count.getCount());
        SightFacet.Item item = new SightFacet.Item(count.getName(),count.getCount());
        items.add(item);
    }

    sightFacet.setFacetCount(items.size()); //设置facet中的个数
    sightFacet.setItems(items); //设置facet中的items
    sightFacets.add(sightFacet); //添加到facet列表中
}
sightList.setSightFacets(sightFacets);
}

//整理景点数据
List<Sight> list = queryResponse.getBeans(Sight.class);
SolrDocumentList documentList = queryResponse.getResults();

sightList.setSights(list);
sightList.setTime(queryResponse.getQTime());
sightList.setTotalNum(documentList.getNumFound());
sightList.setNowPage((int) (documentList.getStart()/documentList.size()) + 1);
sightList.setTotalPage((int) (documentList.getNumFound()/documentList.size())
+ 1);

//美观型json
Gson gson2 = new GsonBuilder().setPrettyPrinting().create();
String json = gson2.toJson(sightList);

//紧凑型json
//String json = sightList.toJson();
return json;
}

```

三种查询方式

- 附近查询、关键字查询、地图查询

根据参数判断执行的动作，通过 `query_type` 参数确定查询方式调用不同函数。

附近搜索

代码位置：github中trip-search项目

中 SearchService/src/main/java/com/fliaping/trip/search/DoQuery.java

```

/**
 * 附近搜索
 */
public void nearQuery(){
    //http://localhost:9090/ss?query_type=near&distance=10&point=22.5347168,113.97
1228

    //设置查询Query
    solrQuery.setQuery("{!geofilt}");

    //设置距离distance
    int distance = notNull(request.getParameter(UrlP.distance.name()),3000); //距离
默认值3000公里
    if (distance >= 0 && distance < 40076) { //判断距离值有效
        solrQuery.set("d",distance);
    }

    //设置坐标点
    String point = notNull(request.getParameter(UrlP.point.name()),"22.5347168,113
.971228"); //默认坐标世界之窗
    solrQuery.set("pt",point);

    //设置排序
    solrQuery.setSort("geodist()", SolrQuery.ORDER.asc);

    //设置显示字段 f1
    solrQuery.setFields("_dist_:geodist()", "*");

    //设置facet
    solrQuery.setFacet(true)
        .setFacetMissing(true)
        .set("facet.field","sight_type");

    System.out.println(solrQuery.toQueryString());

    try {
        QueryResponse queryResponse = solrClient.query(solrQuery);

        //对solr的查询结果，整合后返回
        respJson(wrapResult(queryResponse, false));
        System.out.println("distance:"+distance);

    } catch (SolrServerException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

}

}

关键词搜索

代码位置：github中trip-search项目

中 SearchService/src/main/java/com/fliaping/trip/search/DoQuery.java

```

public void keyQuery(){
    //http://localhost:8983/solr/trip/select?start=0&rows=10&sfield=sight_coordina
    te&fl=_dist_:geodist(),*&q=石河子&pt=44.3093867,86.0555942&sort=geodist() asc&facet=true&facet.missing=true&facet.field=sight_type
    //设置过滤
    //solrQuery.set("fq", request.getParameter(UrlP.keyword.name()));

    //设置关键字
    String keyword = notNull(request.getParameter(UrlP.keyword.name()), "*:*");
    solrQuery.setQuery(keyword);

    //设置坐标点
    String point = notNull(request.getParameter(UrlP.point.name()), "22.5347168,113
    .971228"); //默认坐标世界之窗
    solrQuery.set("pt",point);

    //设置排序
    String sortOrder = notNull(request.getParameter(UrlP.sort_order.name()),SortOr
    der.distance.get()); //默认距离排序

    if (SortOrder.distance.is(sortOrder)) { //距离排序
        solrQuery.setSort("geodist()", SolrQuery.ORDER.asc);

    }else if (SortOrder.score.is(sortOrder)){ //评分排序

        solrQuery.setSort("sight_score_ctrip", SolrQuery.ORDER.desc);

    }else if (SortOrder.price.is(sortOrder)){ //价格排序
        // TODO: 5/15/16 价格排序

    }else if (SortOrder.best.is(sortOrder)){ //综合排序
        // TODO: 5/15/16 综合排序

    }

    //设置过滤
    //设置景点类型过滤
    String sight_type = request.getParameter(UrlP.sight_type.name());
}

```

```
if(sight_type != null){
    String[] type = sight_type.split(",");
    for (String item : type) {
        if (item != null){
            solrQuery.addFilterQuery("sight_type:"+item);
        }
    }
}

//设置高亮
/*solrQuery.setHighlight(true)
    .setHighlightSimplePre("<em>")
    .setHighlightSimplePost("</em>")
    .set("hl.fl", "sight_intro");*/
//设置facet
solrQuery.setFacet(true)
    .setFacetMissing(true)
    .set("facet.field","sight_type");

System.out.println(solrQuery.toQueryString());
try {
    QueryResponse queryResponse = solrClient.query(solrQuery);
    //对solr的查询结果,整合后返回
    respJson(wrapResult(queryResponse,true));
} catch (SolrServerException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
}
```

实现对搜索结果的筛选和排序，需要利用Solr的sort功能和facet功能，这两个是垂直搜索中比较常用的功能。对于如何使用这两个功能，其实在索引建好之后我们并不需要做太多的工作就能使用，只用在查询的时候指定相关的参数，Solr会根据参数来执行相应的查询，获得相应的结果。

因为项目中使用SolrJ作为Solr的客户端，并通过servlet提供对外的服务接口，本文将以介绍如何用SolrJ来实现，此外也会涉及利用HTTP接口的参数使用。

结果排序

但字段排序来说相对简单，通过改变sort参数的值即可，对于根据多字段，然后利用不同字段的权值来进行排序，相对来说复杂点。

对于单字段排序，在HTTP参数中，例如加上：

```
&sort=geodist() asc
```

这即是按照距离升序排列，其中geodist()为solr中的距离函数，得到point点到文档中所标记坐标的距离。

对于多字段参与的排序，即Solr权重

Solr利用Lucene的权重算法，也就是通过一个公式计算每个Documents的得分，然后按得分高低排序，公式如下：

$$\text{Score}(q,d) = \sum_{t \in q} \left(\text{tf}(t \text{ in } d) \cdot \text{idf}(t)^2 \cdot t.\text{getBoost()} \cdot \text{norm}(t,d) \right) \cdot \text{coord}(q,d) \cdot \text{queryNorm}(q)$$

Where:

t = term; d = document; q = query; f = field

$\text{tf}(t \text{ in } d) = \text{numTermOccurrencesInDocument}^{1/2}$

$\text{idf}(t) = 1 + \log (\text{numDocs} / (\text{docFreq} + 1))$

$\text{coord}(q,d) = \text{numTermsInDocumentFromQuery} / \text{numTermsInQuery}$

$\text{queryNorm}(q) = 1 / (\text{sumOfSquaredWeights}^{1/2})$

$\text{sumOfSquaredWeights} = q.\text{getBoost()}^2 \cdot \sum_{t \in q} (\text{idf}(t) \cdot t.\text{getBoost()})^2$

$\text{norm}(t,d) = d.\text{getBoost()} \cdot \text{lengthNorm}(f) \cdot f.\text{getBoost()}$

Figure 3.7 DefaultSimilarity scoring algorithm. Each component in this formula will be explained in detail in the following sections.

其中：

Tf Term frequency，就是条目出现的次数。**Idf** Inverse document frequency，就是用来描述在一个搜索关键字中，不同字词的稀有程度。比如搜索The Cat in the Hat，那么很明显The和in远没有Cat和Hat重要。**Boosting** 这个是我们设置权重的重点，例如在景点结果排序的时候，不光想考虑距离，还想考虑评分、评论人数等，而boosting是不同的Filed有不同权重，之后根据公式计算得分。所以可以看到，我们并不能直接影响solr搜索结果的排序，需要改变权重，进而改变不同Document的得分，从而影响排序。

其中还有很多因子和公式的解释，有兴趣的同学可以参考Solr in action这本书，里面有比较详细的解释。

前一节讲到Query Parser有三种：Standard、Dismax、Extended Dismax，这里权重排序使用到了Dismax。

在HTTP参数中，例如：

```
&defType=dismax&qf=sight_name^10+sight_coordinate^5
```

此外还可以设置bf设置其他Field的权重，可以使用很多Function Query，我没有用这个，所以不能细讲。

```
&bf=sum(div(sight_score,0.01),if(exists(near_hotels),20000,0))
```

这里简单了解下，比如div,代表相除、exists代表如果near_hotels如果不为空那么设置它的权重为20000，为空则为0。记住最后要sum起来，因为从上面的公式可以看出来，boosting是一个变量，所以最好要有一个和值。

项目中实现了距离排序、评分排序、评论数量排序、关键词最佳匹配、综合排序

```

//设置排序
String sortOrder = notNull(request.getParameter(UrlP.sort_order.name()),SortOrder.distance.get()); //默认距离排序


if (SortOrder.distance.is(sortOrder)) { //距离排序
    solrQuery.setSort("geodist()", SolrQuery.ORDER.asc);

}else if (SortOrder.score.is(sortOrder)){ //评分排序

    solrQuery.setSort("sight_score", SolrQuery.ORDER.desc);

}else if (SortOrder.comment.is(sortOrder)){ //评论数量排序
    solrQuery.setSort("sight_comment_num", SolrQuery.ORDER.desc);

}else if (SortOrder.keyword.is(sortOrder)){ //关键词最佳匹配
    // 关键词最佳匹配
    solrQuery.set("defType", "dismax");
    solrQuery.set("qf", "sight_name^2 sight_intro^1 sight_comments^0.8");

}else if (SortOrder.best.is(sortOrder)){ //综合排序
    // &defType=dismax&qf=sight_name^10+sight_coordinate^5
    solrQuery.set("defType", "dismax");
    solrQuery.set("qf", "sight_name^2 sight_intro^1");
    //solrQuery.set("bf", "sum(div(sight_score,0.01),if(exists(near_hotels),20000,0))");
    solrQuery.addSort("geodist()", SolrQuery.ORDER.asc);
    solrQuery.addSort("sight_score", SolrQuery.ORDER.desc);
}

```

结果过滤

利用**FacetField**进行分类

这个比较简单，只用传入要分类的字段即可，例如

```
&facet=true&facet.field=sight_type
```

这样就会对sight_type进行facet，列出这个字段不重复的值

利用**RangeFacet**划分区间

如果想对某一数值字段进行范围划分，需要用到 `facet.range`，例如：

```
&facet=true&facet.range=sight_score
&f.sight_score.facet.range.start=1
&f.sight_score.facet.range.end=5.1
&f.sight_score.facet.range.gap=1
```

这需要划分的字段是sight_score，并且设置划分的起始值为1，终止值为5.1，间隔为1，也就是说划分出来的区间为 [1,2), [2,3), [3,4), [4,5.1)，因为区间是右开的，所以终止值要多一点。

用FacetQuery根据函数查询

如果我们想对距离进行区间划分，这时不能再用RangeFacet功能，只能利用FacetQuery来通过函数查询。

例如： facet.query={!frange l=0 u=5}geodist() 表示先根据geodist()函数得到距离，然后限制距离是1-5KM的景点。

```
//设置facet
solrQuery.setFacet(true)
    .setFacetMissing(true);
//景点类型facet
solrQuery.addFacetField(new String[]{"sight_type"});

//评分范围facet
solrQuery.add("facet.range","sight_score");

solrQuery.add("f.sight_score.facet.range.start","1")
    .add("f.sight_score.facet.range.end","5.1")
    .add("f.sight_score.facet.range.gap","1");

//距离范围facet
solrQuery.addFacetQuery("{!frange l=0 u=5}geodist()")
    .addFacetQuery("{!frange l=5.001 u=50}geodist()")
    .addFacetQuery("{!frange l=50.001 u=500}geodist()")
    .addFacetQuery("{!frange l=500.001 u=5000}geodist()");
```

参考链接

- Solr高亮与Field权重

客户端的设计，利用的是Hybrid APP的构建技术，核心是HTML5技术，下面先简单介绍下 Hybrid应用的以及Ionic框架。

技术介绍

Hybrid App介绍

移动app可以大致被分为三种，native、hybrid和web app。如果使用native app，你可以使用设备和操作系统的所有能力，同时，平台的性能负荷最小。然而，构建web app可以让你的代码跨平台，使得开发时间和成本大大减少。而hybrid app把这两者的优点都结合起来，使用一套共同代码，在许多不同的平台上部署类似原生的app。

有两种构建hybrid app的方法：

1) Webview app：HTML,CSS和Javascript基础代码在一个内部的浏览器（叫做WebView）中运行，这个浏览器打包在一个原生的app中，一些原生的API可以通过这个包被Javascript获得，比如Adobe PhoneGap和Trigger.io。2) 被编译的hybrid app：用一种语言编写代码（如C#或者Javascript），对于每一种支持的平台都把代码编译进原生代码中，这样做的结果是，每一个平台都有一个原生的app，但是在开发过程中少了一些自由空间。可以看一下这些例子，Xamarin，Appcelerator Titanium，Embarcadero FireMonkey。

优点：

- 开发人员可以使用现有的网页技术
- 对于多种平台使用一套基础代码
- 减少开发时间和成本
- 使用响应式网页设计可以非常简便的设计出多样的元素（包括平板）
- 一些设备和操作系统特征的访问
- 高级的离线特性
- 可见度上升，因为app可以原生发布（通过app store），也可以发布给移动端浏览器（通过搜索引擎）

缺点：

- 某些特定app的性能问题（那些依赖于复杂的原生功能或者繁重的过渡动画的app，如3D游戏）
- 为了模拟native app的UI和感官所增加的时间和精力
- 并不完全支持所有的设备和操作系统
- 如果app的体验并不够原生化，有被Apple拒绝的风险（比如说一个简单的网站）
- 这些缺点比较显著，不能忽略，它们告诉我们，并不是所有的app都适合混合模式，你需要小心的预计你的目标用户、他们对平台的选择和对app的需求。对于许多app来说，好

处都是大于坏处的，比如内容驱动的app。

Ionic介绍

Ionic是一个强大的HTML5应用程序开发框架(HTML5 Hybrid Mobile App Framework)。可以帮助您使用Web技术，比如HTML、CSS和Javascript构建接近原生体验的移动应用程序。ionic主要关注外观和体验，以及和你的应用程序的UI交互，特别适合用于基于Hybird模式的HTML5移动应用程序开发。ionic是一个轻量的手机UI库，具有速度快，界面现代化、美观等特点。为了解决其他一些UI库在手机上运行缓慢的问题，它直接放弃了IOS6和Android4.1以下的版本支持，来获取更好的使用体验。Ionic特点：

1. ionic 基于Angular语法，简单易学。
2. ionic 是一个轻量级框架。
3. ionic 完美的融合下一代移动框架，支持Angularjs的特性，MVC，代码易维护。
4. ionic 提供了漂亮的设计，通过SASS构建应用程序，它提供了很多UI组件来帮助开发者开发强大的应用。
5. ionic 专注原生，让你看不出混合应用和原生的区别
6. ionic 提供了强大的命令行工具。
7. ionic 性能优越，运行速度快。

Ionic学习

前导知识

- HTML5技术
- JavaScript
- AngularJS

基础学习

请参考网络其他教程，例如：

- [ionic 教程-菜鸟教程](#)
- [ionic中文教程-皓眸大前端](#)

当然最好的还是官方的文档 [Ionic Documentation](#)

安装ionic：`npm install -g cordova ionic`

本篇文章针对已经入门Ionic的童鞋，没有入门的请先完成基础学习。

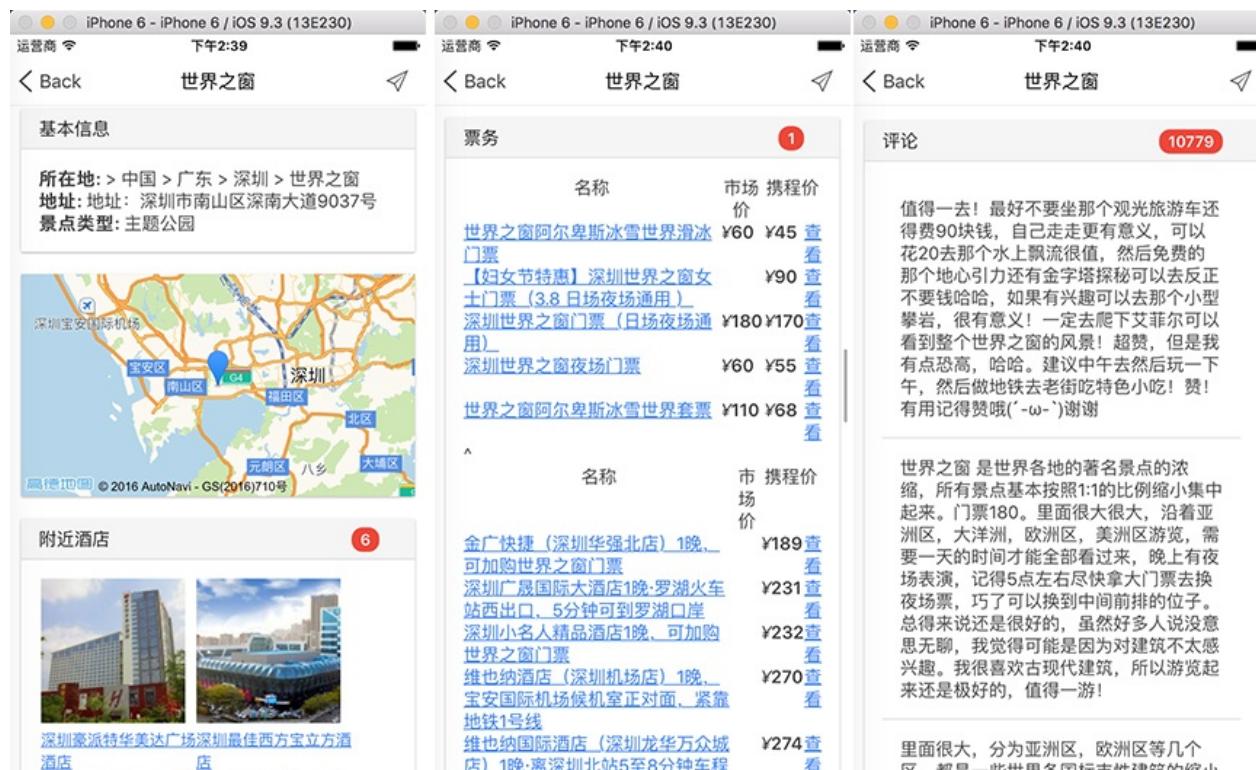
界面介绍

界面截图

先界面贴出效果。

下图：主界面和景点详情页面





下图：原始链接页面



深圳景点 第1名

世界之窗正像它的名字那样，是一扇展现世界名胜的窗口的主题乐园，里面有世界各知名景点的缩影。除外，这里还有世界各地的风情表演，日本茶道、非洲风情歌舞秀等，等待你的观赏。

[查看完整攻略 >](#)

A map showing the location of the "Shenzhen Window of the World" attraction. The map includes street names like "白石街" and "沙河" and shows the surrounding urban environment. A blue pin marks the exact location of the attraction. Below the map, there is a button labeled "写点评" (Write Review) with a pen icon. At the bottom of the screen, there is a URL "http://m.ctrip.com/html5/you/sight/shenzhe..." and a "Done" button.

下图：AND和OR查询

4.4 基于Ionic的客户端设计

This screenshot shows the search results for '北京 OR 天文馆' (Beijing OR Observatory). It lists various landmarks with their names, distances from Beijing, ratings, and review counts. A '加载更多' (Load More) button is visible at the bottom right of the list.

地点	距离	评分	评论数量
故宫	2535.23 km	4.7	21576
简介:绝大多数来北京的游人，尤其是初游者，都会...			
颐和园	2522.07 km	4.7	7992
简介:颐和园原是清朝帝王的行宫和花园，又称清漪...			
拙政园	3330.32 km	4.4	6186
简介:拙政园是苏州最大、最著名的园林，全园以水...			
天安门广场	2535.56 km	4.6	6008
简介:天安门广场位于北京的中轴线上，是共和国的...			
八达岭长城	2487.73 km	4.5	5874
简介:八达岭长城位于延庆县军都山关沟古道北口，...			
夫子庙	3143.50 km	4.2	3553
简介:夫子庙始建于宋代，位于秦淮河北岸的贡院街...			
圆明园	2523.55 km	4.5	3402
简介:圆明园是清代著名的皇家园林，由圆明园、长...			

下方有排序、地图、筛选按钮。

下图：结果排序页面

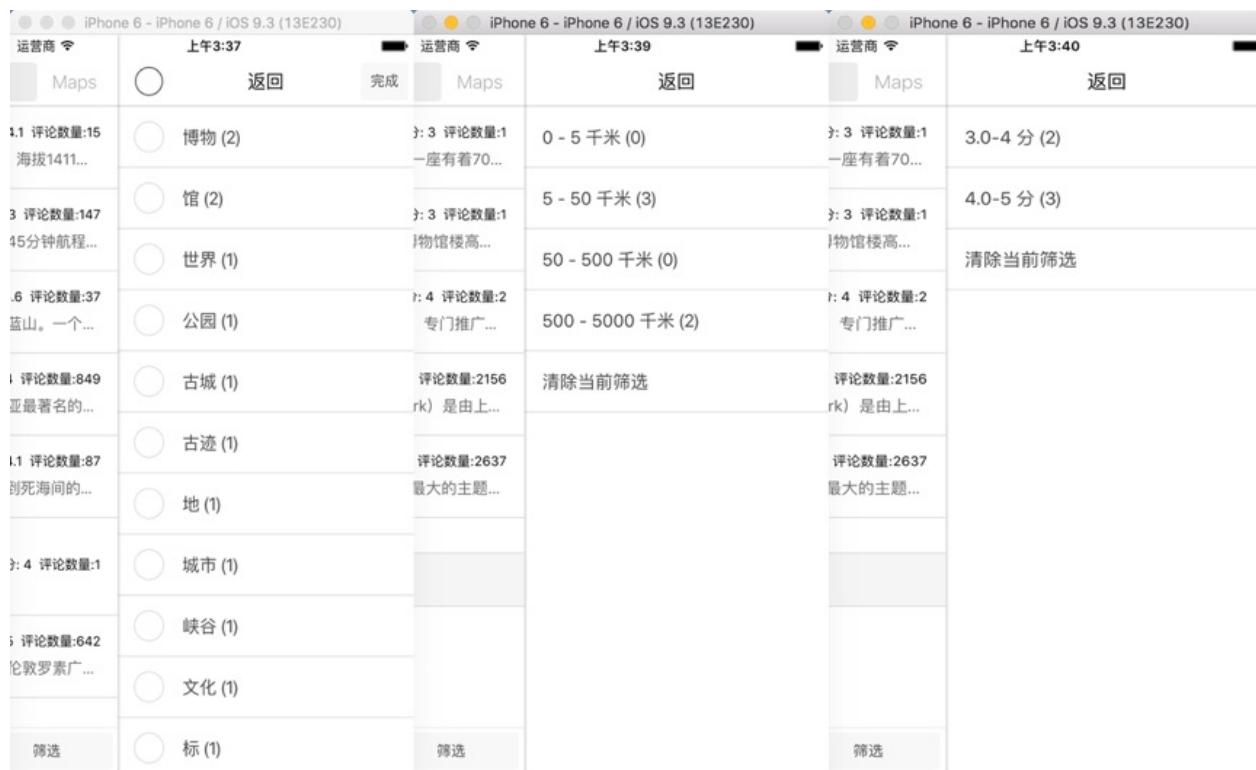
This screenshot shows the search results for 'Happy'. It lists various locations with their names, distances from Hong Kong, ratings, and review counts. A '加载更多' (Load More) button is visible at the bottom right of the list. On the left, there is a sidebar with sorting options: '按距离排序' (Sort by Distance), '按评分排序' (Sort by Rating), '按评论数量排序' (Sort by Number of Reviews), '综合排序' (Comprehensive Sorting), and '关键词最佳匹配' (Best Match for Keywords). The '按距离排序' option is currently selected.

地点	距离	评分	评论数量
景贤里	37.34 km	3	1
简介:景贤里位于香港岛湾仔司徒拔道，是一座有着70...			
F11博物馆	37.46 km	3	1
简介:F11博物馆是香港首个摄影博物馆，博物馆楼高...			
YY9画廊	38.00 km	4	2
简介:YY9画廊由一位家居设计顾问所创立，专门推广...			
凯蒂猫家园	1065.95 km	4.2	2156
简介:中国首个凯蒂猫家园 (Hello Kitty Park) 是由上...			
欢乐谷	1941.85 km	4.7	2637
简介:北京欢乐谷位于北京东四环，是北京最大的主题...			
凯蒂猫家园	1065.95 km	4.2	2156
简介:中国首个凯蒂猫家园 (Hello Kitty Park) 是由上...			
欢乐谷	1941.85 km	4.7	2637
简介:北京欢乐谷位于北京东四环，是北京最大的主题...			

下方有排序、地图、筛选按钮。

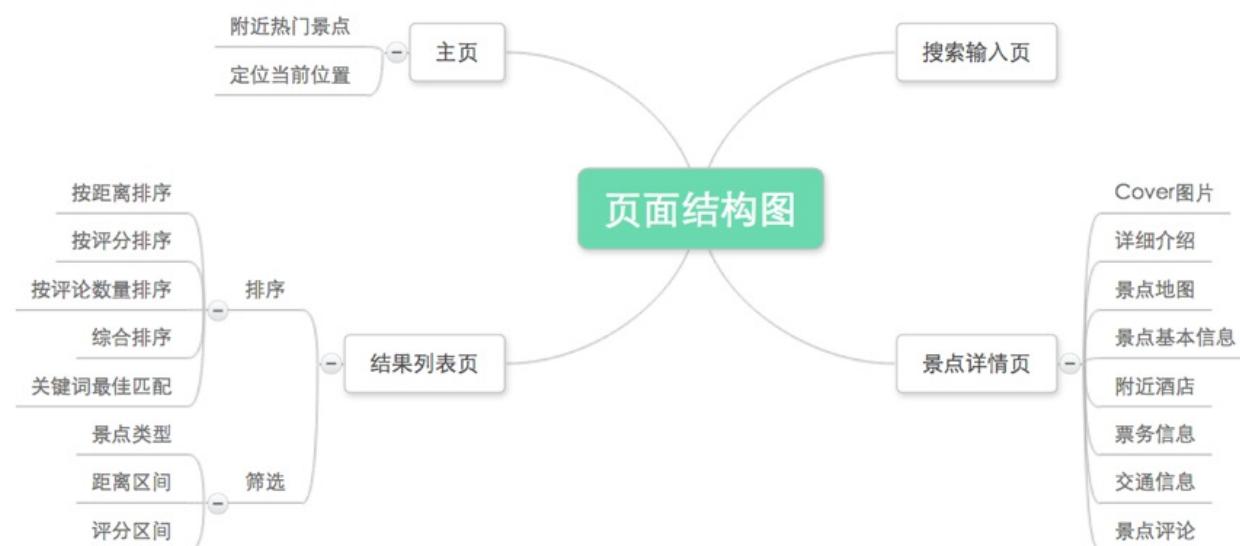
下图：结果筛选页面

4.4 基于Ionic的客户端设计



界面逻辑结构

页面结构图如下：



项目构建

关于Ionic框架的详情使用，请参考基础学习内容和项目中的源码。

本文源码在git项目trip-search下的 SearchApp/www 中。

ngCordova插件模块

- 在源码起始HTML页面（index.html）中，引入 `ng-cordova.js` 文件，如下，把该文件放在项目目录的 `lib/ngCordova/dist` 目录下，并在 `index.html` 中添加如下代码片段。

```
<script src="lib/ngCordova/dist/ng-cordova.js"></script>
<script src="cordova.js"></script>
```

- 将 `ng-cordova` 模块注入应用，如下代码，在 `angular` 中构建 `module` 时加入 `ngCordova` 模块。

```
var app = angular.module('searchIndex', ['ionic', 'ngCordova']);
```

定位插件

本项目中使用了设备的定位功能，如果简单使用HTML的定位功能，在当前Ionic环境下并没有效果，因为可能Ionic构建出来的应用并不是标准的浏览器，并没有调用设备的定位硬件，但Ionic的最大好处是Hybrid方式，既可以通过中间层 `Cordova` 来调用设备硬件。

插件介绍页：<http://ngcordova.com/docs/plugins/geolocation/>

以下JS代码是 `SearchApp/www/js/searchIndex.js`

- 安装插件：

```
cordova plugin add cordova-plugin-geolocation
```

- 在 controller 中注入 `$cordovaGeolocation` 插件。

```
app.controller("homeCtrl", ['$scope', '$cordovaGeolocation',
  function ($scope, $cordovaGeolocation){
    //.....
  }
])
```

- 在代码中使用该插件的定位功能

```

var isIOS = ionic.Platform.isIOS();
var isAndroid = ionic.Platform.isAndroid();
console.log("Platform:"+ionic.Platform.platform());

if(isAndroid || isIOS){
    var posOptions = {timeout: 10000, enableHighAccuracy: false};
    $cordovaGeolocation
        .getCurrentPosition(posOptions)
        .then(function (position) {
            window.localStorage['latitude'] = position.coords.latitude;
            window.localStorage['longitude'] = position.coords.longitude;
            //scope.closeLocation();
            $scope.loading(false);
        }, function(err) {
            // error
        });
}

```

应用内网页浏览

插件介绍页：<http://ngcordova.com/docs/plugins/inAppBrowser/>

- 安装插件：

```
cordova plugin add cordova-plugin-inappbrowser
```

- 在controller中注入 \$cordovaInAppBrowser 插件。

```

app.controller("homeCtrl", ['$scope', '$cordovaInAppBrowser',
    function ($scope, $cordovaInAppBrowser){
        //.....
    }
])

```

- 在代码中使用该插件，实现应用内打开链接

```

$scope.openUrl = function (url) {
    var isIOS = ionic.Platform.isIOS();
    var isAndroid = ionic.Platform.isAndroid();
    console.log("Platform:"+ionic.Platform.platform());

    if(isAndroid || isIOS){

        var options = {
            location: 'yes',
            clearcache: 'yes',
            toolbar: 'yes'
        };

        $cordovaInAppBrowser.open(url, '_blank', options)

    }else {
        $window.open(url);
    }

};

```

调试

Android平台

由于Android平台的开放特性，发布该平台软件比较简单，直接利用Ionic提供的命令即可。

在本项目的目录下执行如下命令

```

ionic platform android      #添加android平台
ionic build android         #构建android平台代码
ionic emulate android       #在android模拟器上运行程序
ionic run android           #在运行android机器上运行程序

```

`emulate` 命令是打开android模拟器并加载应用，`run` 命令需要将手机插到电脑上，并确认Debug功能打开，并在手机上对该电脑进行了认证。

IOS平台

因为ios平台的封闭性，如果想在ios平台进行调试，可以利用Xcode提供的模拟器，当然需要在OSX平台上进行。此外也可以利用Xcode直接加载Ionic构建好的工程，在真机上调试。

```
ionic platform ios #添加ios平台  
ionic build ios   #构建ios平台代码  
ionic emulate ios #在ios模拟器上运行程序
```

如果在真机上进行调试，需要在构建ios平台代码之后打开Xcode，加载Ionic应用目录下的platforms/ios/TripSearch.xcodeproj工程。之后登陆Apple账号，获得免费的ios开发授权之后即可对项目编译并在真机上运行。

整个过程还是比较麻烦的，需要一定的经验和问题解决能力，因为这不是我们的重点，这里不再多说。

发布

真正的发布需要签名之后发布到各大应用市场，笔者没有相关经验，但觉得这些都还简单，应该难不倒聪明的大家。

进阶部分

之前的内容介绍都是单机下的，这里将介绍集群相关内容，当然笔者真正实践经验，做的也只是原型，只有参考价值。

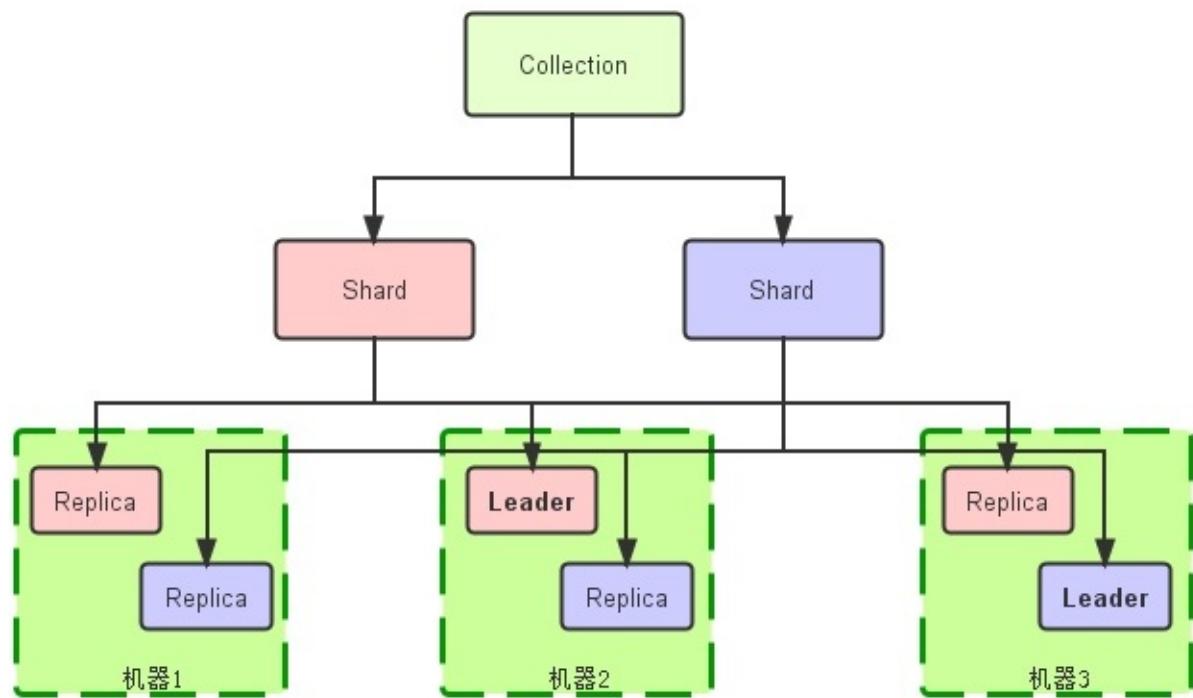
本节是SolrCloud基础理论知识，我也是从网上学习到，这里只是进行一些整理。参考的博客比本文更好，更有深度，有耐心的请看参考的原文-- [SolrCloud之分布式索引及与Zookeeper的集成](#)

SolrCloud基本概念

SolrCloud模式下有Cluster，Node，Collection，Shard，LeaderCore，ReplicationCore等重要概念。

1. **Cluster** 集群：Cluster是一组Solr节点，逻辑上作为一个单元进行管理，整个集群必须使用同一套schema和SolrConfig。
2. **Node** 节点：一个运行Solr的JVM实例。
3. **Collection**：在SolrCloud集群中逻辑意义上的完整的索引，常常被划分为一个或多个Shard，这些Shard使用相同的Config Set，如果Shard数超过一个，那么索引方案就是分布式索引。SolrCloud允许客户端用户通过Collection名称引用它，这样用户不需要关心分布式检索时需要使用的和Shard相关参数。
4. **Core**: 也就是Solr Core，一个Solr中包含一个或者多个Solr Core，每个Solr Core可以独立提供索引和查询功能，Solr Core的提出是为了增加管理灵活性和共用资源。SolrCloud中使用的配置是在Zookeeper中的，而传统的Solr Core的配置文件是在磁盘上的配置目录中。
5. **Config Set**: Solr Core提供服务必须的一组配置文件，每个Config Set有一个名字。最小需要包括solrconfig.xml和schema.xml，除此之外，依据这两个文件的配置内容，可能还需要包含其它文件，如中文索引需要的词库文件。Config Set存储在Zookeeper中，可以重新上传或者使用upconfig命令进行更新，可使用Solr的启动参数bootstrap_confdir进行初始化或更新。
6. **Shard** 分片：Collection的逻辑分片。每个Shard被分成一个或者多个replicas，通过选举确定哪个是Leader。
7. **Replica**: Shard的一个拷贝。每个Replica存在于Solr的一个Core中。换句话说一个Solr Core对应着一个Replica，如一个命名为“test”的collection以numShards=1创建，并且指定replicationFactor为2，这会产生2个replicas，也就是对应会有2个Core，分别存储在不同的机器或者Solr实例上，其中一个会被命名为test_shard1_replica1，另一个命名为test_shard1_replica2，它们中的一个会被选举为Leader。
8. **Leader**: 赢得选举的Shard replicas，每个Shard有多个Replicas，这几个Replicas需要选举来确定一个Leader。选举可以发生在任何时间，但是通常他们仅在某个Solr实例发生故障时才会触发。当进行索引操作时，SolrCloud会将索引操作请求传到此Shard对应的leader，leader再分发它们到全部Shard的replicas。
9. **Zookeeper**: Zookeeper提供分布式锁功能，这对SolrCloud是必须的，主要负责处理Leader的选举。Solr可以以内嵌的Zookeeper运行，也可以使用独立的Zookeeper，并且Solr官方建议最好有3个以上的主机。

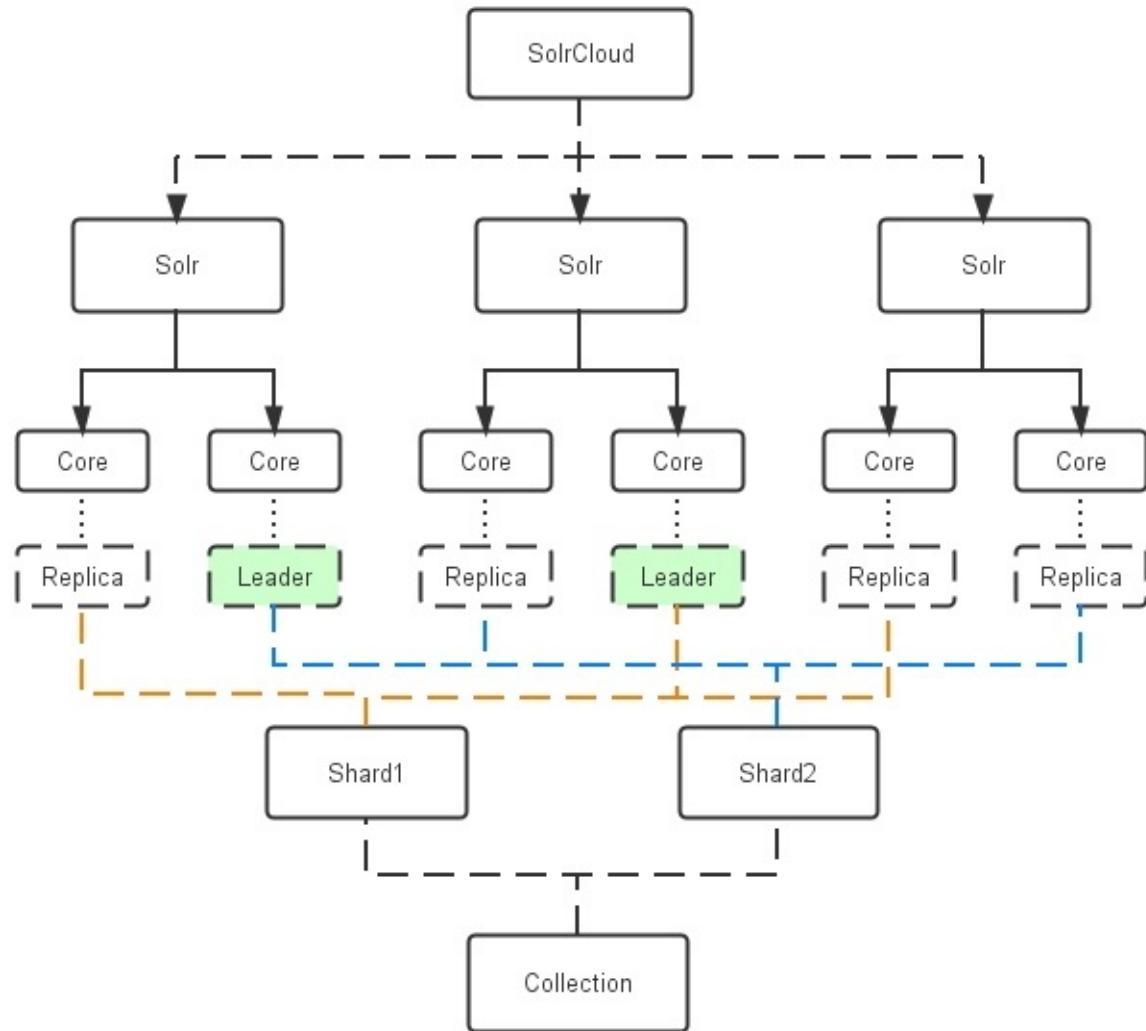
Collection逻辑图



解释：

- 从上图可以看到，有一个collection，被分为两个shard，每个shard分为三个replica，其中每个shard从自己的三个replica选择一个作为Leader。
- 每个shard的replica被分别存储在三台不同的机器（Solr实例）中，这样的目的是容灾处理，提高可用性。如果有一个机器挂掉之后，因为每个shard在别的机器上有复制品，所以能保证整个数据的可用，这是Solrcloud就会在还存在的replica中重新选举一个作为这个shard的Leader。

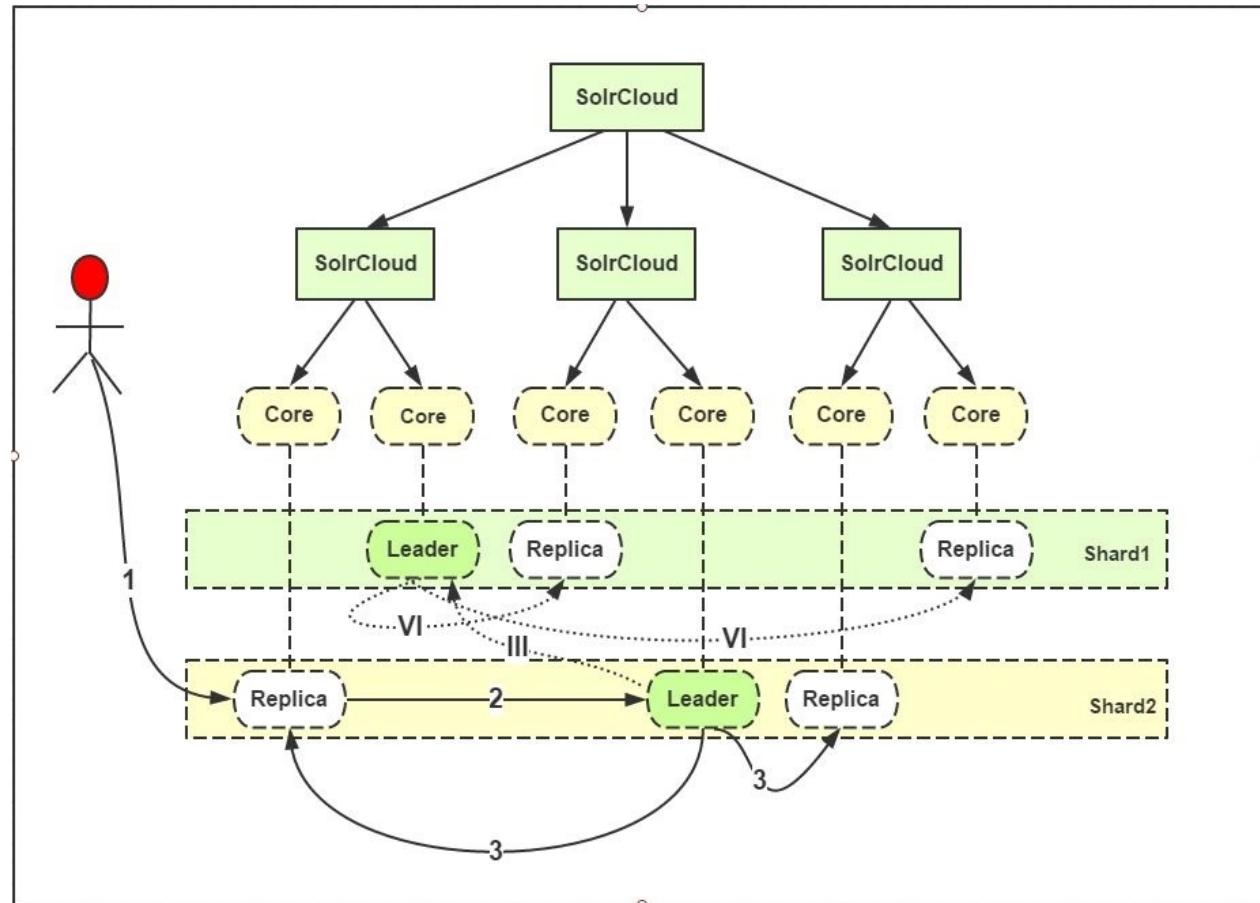
SolrCloud的工作模式



解释：

- 上图的下半部分就是Collection逻辑图，上半部分是SolrCloud的物理结构，每个solr实例中有两个core，每个core对应一个Shard的一个replica。
- 当Solr Client通过Collection访问Solr集群的时候，便可通过Shard分片找到对应的Replica即Solr Core，从而就可以访问索引文档了。

SolrCloud创建索引和更新索引



解释：

1. 用户可以把新建文档提交给任意一个Replica（Solr Core）
 1. 如果它不是leader，它会把请求转给和自己同Shard的Leader。
 2. Leader把文档路由给本Shard的每个Replica。 III. 如果文档基于路由规则(如取hash值)并不属于当前的Shard，leader会把它转交给对应Shard的Leader。 VI. 对应Leader会把文档路由给本Shard的每个Replica。

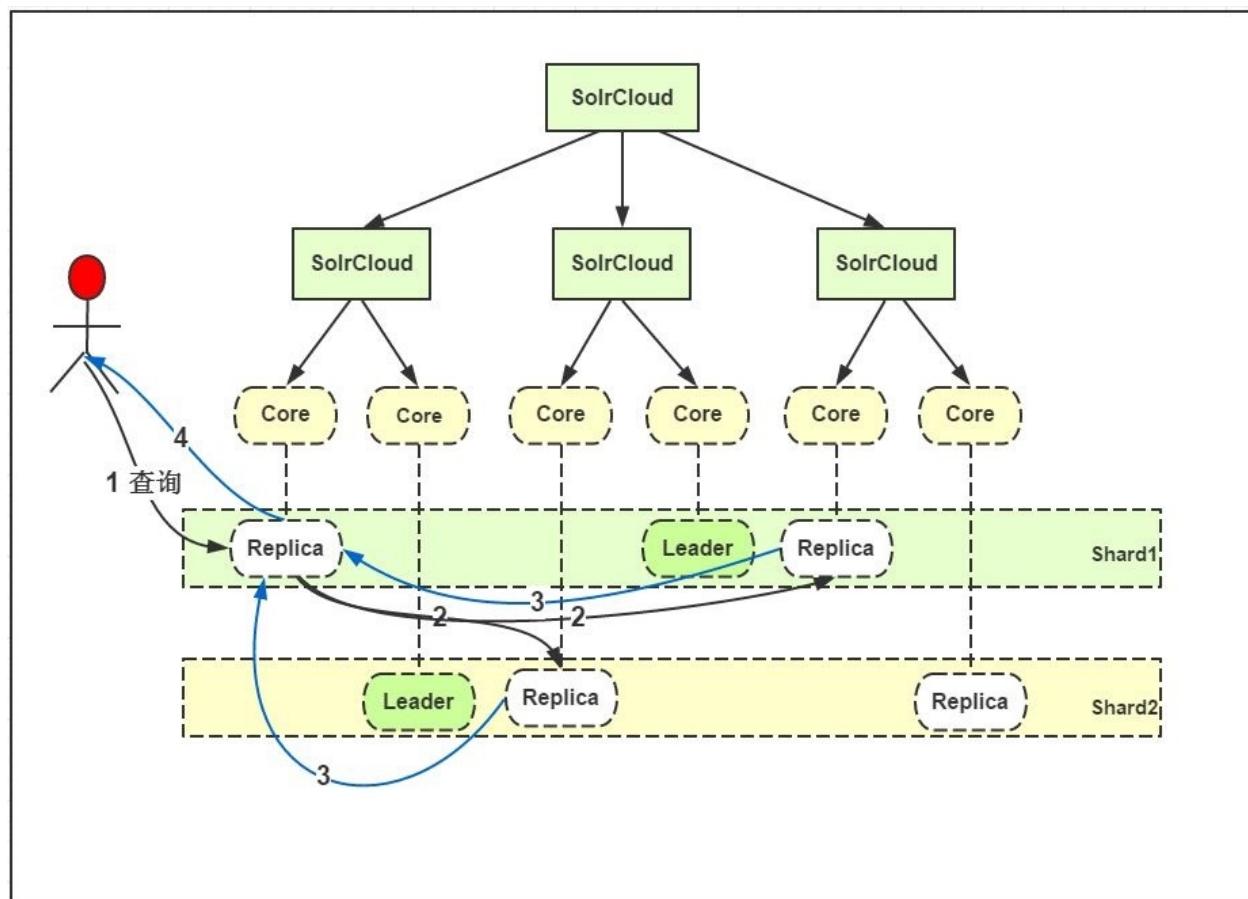
更新索引：

1. Leader接受到update请求后，先将update信息存放到本地的update log，同时Leader还会给document分配新的version，对于已存在的document，如果新的版本高就会抛弃旧版本，最后发送至replica。
2. 一旦document经过验证以及加入version后，就会并行的被转发至所有上线的replica。SolrCloud并不会关注那些已经下线的replica，因为当他们上线时候会有recovery进程对他们进行恢复。如果转发的replica处于recovering状态，那么这个replica就会把update放入update transaction 日志。
3. 当leader接受到所有的replica的反馈成功后，它才会反馈客户端成功。只要shard中有一个replica是active的，Solr就会继续接受update请求。这一策略其实是牺牲了一致性换取了写入的有效性。

近实时搜索：

SolrCloud支持近实时搜索（near real time），所谓的近实时搜索即在较短的时间内使得新添加的document可见可查，这主要基于softcommit机制。软提交（softcommit）指的是仅把数据提交到内存，index可见，此时没有写入到磁盘索引文件中。

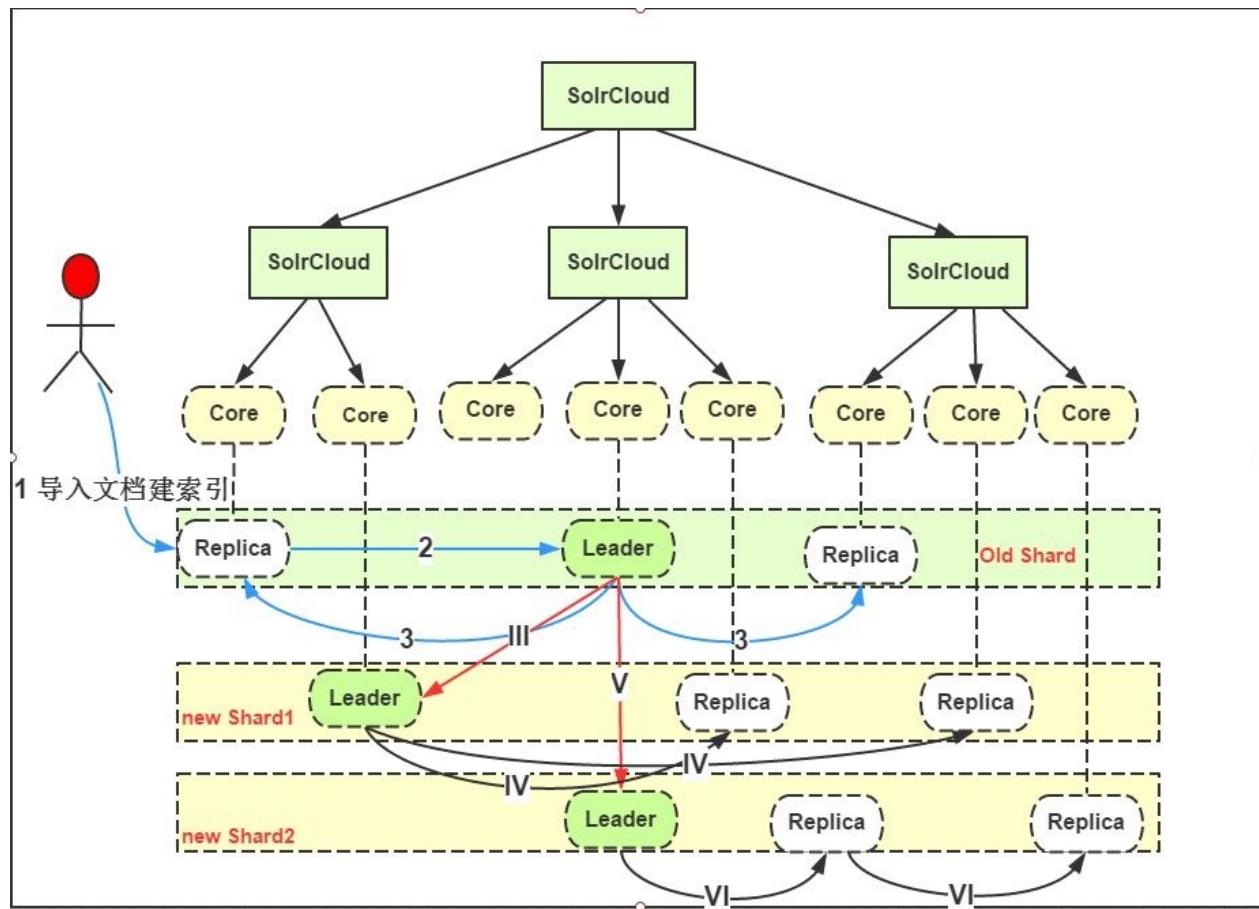
SolrCloud 索引的检索



解释：

1. 用户的一个查询，可以发送到含有该Collection的任意Solr的Server，Solr内部处理的逻辑会转到一个Replica。
2. 此Replica会基于查询索引的方式，启动分布式查询，基于索引的Shard的个数，把查询转为多个子查询，并把每个子查询定位到对应Shard的任意一个Replica。
3. 每个子查询返回查询结果。
4. 最初的Replica合并子查询，并把最终结果返回给用户。

Solr Shard Splitting 的具体过程



解释：

- 一般情况下，增加Shard和Replica的数量能提升SolrCloud的查询性能和容灾能力，但是我们仍然得根据实际的document的数量，document的大小，以及建索引的并发，查询复杂度，以及索引的增长率来统筹考虑Shard和Replica的数量。
- 多个shard的情况下需要对文档进行分片，这就是这节要讲的。

Shard分割的具体过程（old shard split为newShard1和newShard2）可以描述为：

- 在一个Shard的文档到达阈值，或者接收到用户的API命令，Solr将启动Shard的分裂过程。
- 此时，原有的Shard仍然会提供服务，Solr将会提取原有Shard并按路由规则，转到新的Shard做索引。

同时，新加入的文档：

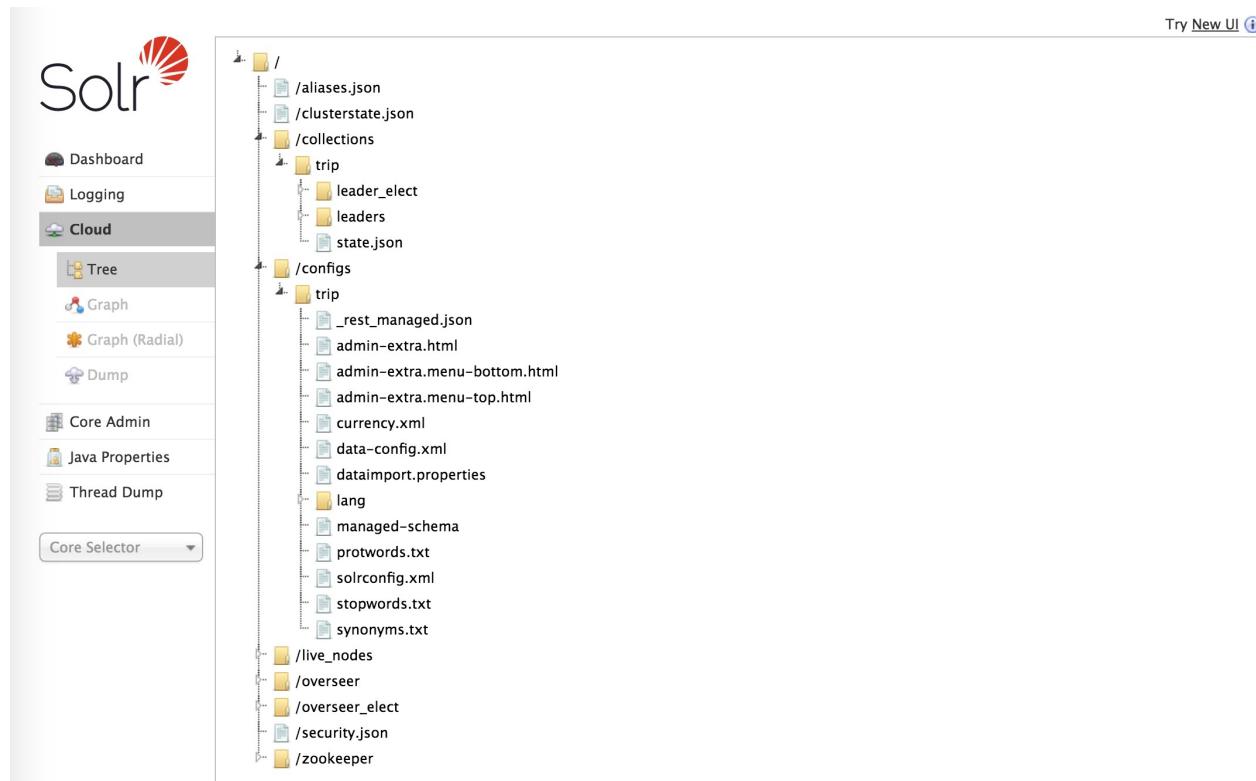
- 用户可以把文档提交给任意一个Replica
- Replica将文档转交给Leader。
- Leader把文档路由给原有Shard的每个Replica，各自做索引。

III.V. 同时，会把文档路由给新的Shard的Leader

IV.VI. 新Shard的Leader会路由文档到自己的Replica，各自做索引，在原有文档重新索引完成，系统会把分发文档路由切到对应的新的Leader上，原有Shard关闭。Shard只是一个逻辑概念，所以Shard的Splitting只是将原有Shard的Replica均匀的分不到更多的Shard的更多的

Solr节点上去。

Zookeeper



以上为本项目中的Zookeeper的文件结构，可以看到上传的Solr配置文件。

zookeeper的主要作用有：

- 集中配置存储以及管理。
- 集群状态改变时进行监控以及通知。
- shard leader的选举。

在本项目中，为了更好的模拟真实的生产环境，对于SolrCloud技术不采用伪集群方式，而是真正实现一个搜索集群，当然由于笔者在做该项目时仍是学生，没有资金租用多台服务器来搭建集群，因此借助于Docker的容器技术在一台服务器上虚拟出逻辑上的六台主机。

```

1. root@iZ2880wp05zZ: ~ (ssh)
Welcome to Ubuntu 14.04.2 LTS (GNU/Linux 3.13.0-32-generic x86_64)

 * Documentation: https://help.ubuntu.com/

Welcome to aliyun Elastic Compute Service!

Last login: [REDACTED]
root@iZ2880wp05zZ:~# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
b49582d15da4      xuping/ubuntu_java   "/bin/bash /exe/solrR"   2 days ago        Up 2 days          solr3
cdd6bd6ce7d9      xuping/ubuntu_java   "/bin/bash /exe/solrR"   2 days ago        Up 2 days          solr2
50cc933803b3      xuping/ubuntu_java   "/bin/bash /exe/solrR"   2 days ago        Up 2 days          solr1
b634b37e00d2      xuping/ubuntu_java   "/bin/bash /exe/zooRu"  2 days ago        Up 2 days          zoo3
7544e30419bd      xuping/ubuntu_java   "/bin/bash /exe/zooRu"  2 days ago        Up 2 days          zoo2
504dcba4de307     xuping/ubuntu_java   "/bin/bash /exe/zooRu"  2 days ago        Up 2 days          zoo1
root@iZ2880wp05zZ:~#

```

在阅读本节之前，需要对Docker技术和SolrCloud的基础知识有一定的了解，可以参考一下链接。

- [Docker —— 从入门到实践](#)
- [SolrCloud基础](#)

java环境镜像制作

本项目中的java环境基于ubuntu镜像，除了jdk还安装了必要的工具，例如vim，ifconfig,ssh等。

已经做好的镜像可以从docker hub或者上拉取

```
docker pull xuping/ubuntu_java
```

运行镜像

1. 在运行镜像之前，要先建立容器要挂载的目录。在本项目中，我在系统根目录新建data目录，并在该目录包含如下目录，其中 `solr*` 为三个solr实例的程序执行目录，其中 `zookeeper*` 为三个zookeeper实例的执行目录。

```
root@iZ2880wp05zZ:/data# tree -L 1
.
|-- conf
|-- data
|-- exe
|-- jetty
|-- solr1
|-- solr2
|-- solr3
|-- zookeeper1
|-- zookeeper2
`-- zookeeper3
```

solr* 目录中放置Solr的可运行程序，其实就是solr安装目录，需要做的修改是将solr安装目录下bin/solr.in.sh，将ZK_HOSTS的值改为zookeeper的地址，在本项目中就是ZK_HOSTS="zoo1:2181,zoo2:2181,zoo3:2181"，去掉注释使之生效。

zookeeper* 目录中放置Zookeeper的可运行程序，这里需要修改的是zookeeper安装目录下conf/zoo.cfg(新建)，以及在data目录下新建myid文件，并写入当前服务的id,例如 server.1 就在myid中写入1。

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/data/data
clientPort=2181

server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

2. 运行镜像，启动zookeeper实例：

```
docker run -d --name zoo1 -h zoo1 -v /data/zookeeper1:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/zooRun.sh

docker run -d --name zoo2 -h zoo2 -v /data/zookeeper2:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/zooRun.sh

docker run -d --name zoo3 -h zoo3 -v /data/zookeeper3:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/zooRun.sh
```

解释：

- run 运行镜像
- -d 保持后台运行

- --name zoo2 -h zoo2 镜像的运行实例，即container的名字和主机名都为zoo2
- -v /data/zookeeper2:/data 挂载宿主机的/data/zookeeper2文件夹到container中的/data
- xuping/ubuntu_java 要运行的镜像
- /bin/bash /exe/zooRun.sh 在镜像启动时，要运行的命令。

3. 运行镜像，启动Solr实例：

```
docker run -d --name solr1 -h solr1 -v /data/solr1:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/solrRun.sh

docker run -d --name solr2 -h solr2 -v /data/solr2:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/solrRun.sh

docker run -d --name solr3 -h solr3 -v /data/solr3:/data -v /data/exe:/exe xuping/ubuntu_java /bin/bash /exe/solrRun.sh
```

4. 通过 docker ps 查看运行的容器，如果运行成功即会看到本文开始的那张图片。

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b49582d15da4	xuping/ubuntu_java	"./bin/bash /exe/solrR"	2 days ago	Up 2 days		solr3
cdd6bdce7d9	xuping/ubuntu_java	"./bin/bash /exe/solrR"	2 days ago	Up 2 days		solr2
50ca933803b3	xuping/ubuntu_java	"./bin/bash /exe/solrR"	2 days ago	Up 2 days		solr1
b634b37e00d2	xuping/ubuntu_java	"./bin/bash /exe/zooRu"	2 days ago	Up 2 days		zoo3
7544e30419bd	xuping/ubuntu_java	"./bin/bash /exe/zooRu"	2 days ago	Up 2 days		zoo2
504dcba4de307	xuping/ubuntu_java	"./bin/bash /exe/zooRu"	2 days ago	Up 2 days		zoo1

5. 由于我们没有定制docker容器的网络，虽然同一台机器上的容器默认在同一网络中，但其网络是自动分配，需要将各自的ip进行同步。利用docker的挂载功能，将同一目录挂载到所有容器，这样即可用脚本来同步所有容器的网络信息，其脚本如下所示，作用不仅是同步网络，另外也是保持容器后台持续运行。

```

service ssh start    #启动ssh服务
/data/bin/zkServer.sh start  #启动zookeeper服务

local_ip=`/sbin/ifconfig -a|grep inet|grep -v 127.0.0.1|grep -v inet6|awk '{print $2}' |tr -d 'addr:'` 
host_item=$local_ip` `hostname` 
echo $host_item >> /exe/hosts

input="/exe/hosts"
while true;
do echo update hosts;

    echo '' > /etc/hosts
    while IFS= read -r var
    do
        echo "$var" >> /etc/hosts
    done < "$input"
sleep 10;
done

```

同样的，在Solr实例中也有类似的脚本。

创建collection

通过ssh连接上任意一个solr实例，转到solr安装目录。

1. 上传配置文件到Zookeeper中

```

server/scripts/cloud-scripts/zkcli.sh -zkhost zoo1:2181,zoo2:2181,zoo3:2181 -cmd upconfig -confname trip -confdir ./conf

```

解释：

- -cmd upconfig 表示操作是上传配置
- -confname 表示在Zookeeper中的配置文件所放的路径
- -confdir 表示本地要上传的配置文件的路径

2. 通过Url API来创建collection

```

wget http://solr1:8983/solr/admin/collections?action=CREATE&name=trip&numShards=3&replicationFactor=3&maxShardsPerNode=5&collection.configName=trip

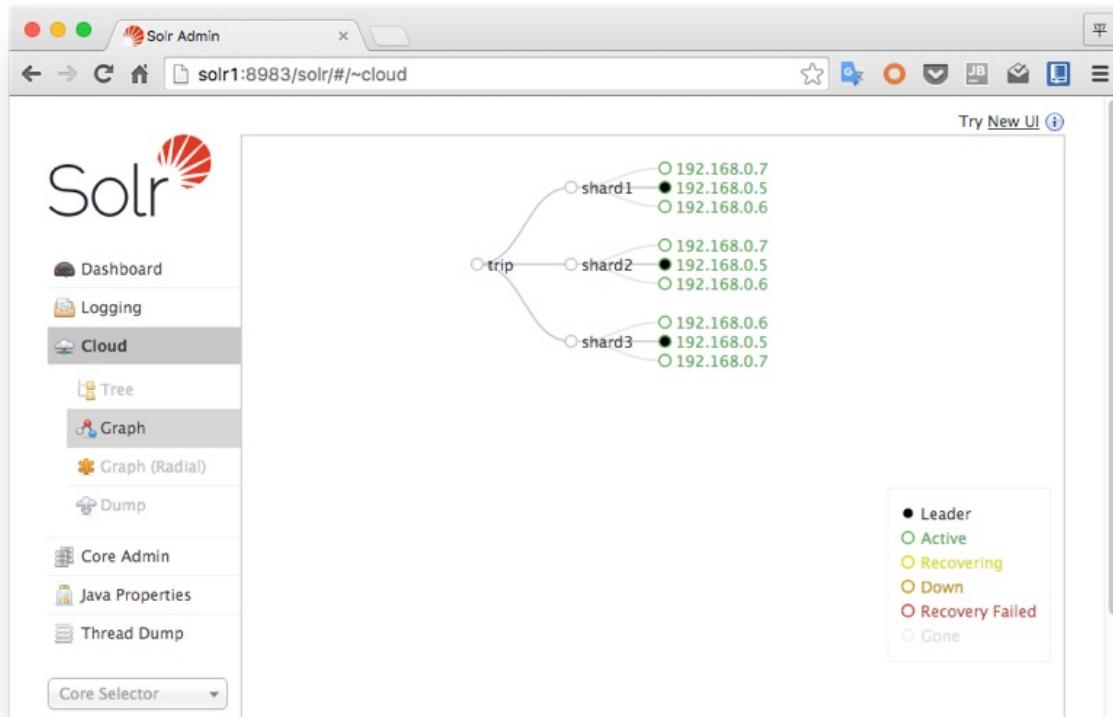
```

解释：

- action=CREATE 动作为创建collections

- name=trip collection的名字为trip
- numShards=3 分片数量为3
- replicationFactor=3 复件个数为3
- maxShardsPerNode=5 每节点最大分片个数为5
- collection.configName=trip 要创建的collection在zookeeper中的配置文件的路径

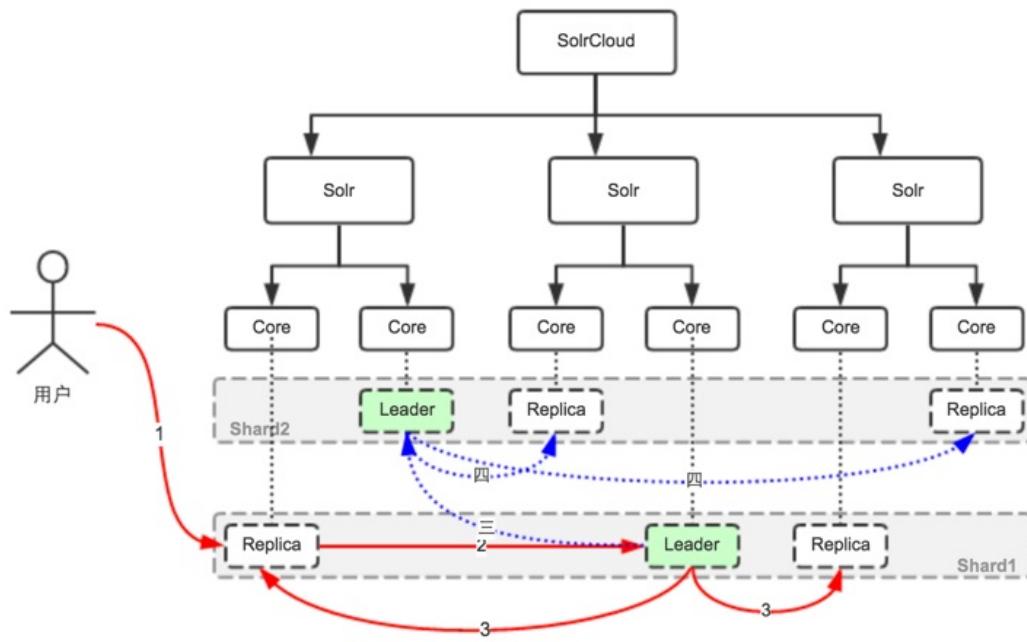
成功之后，访问任一solr节点的管理界面，会看到如下所示的样子。（当然前提是你的浏览器要和这些节点在同一网络内，无论是代理，还是VPN都能实现）



建立索引

还是选择从mysql中导入数据，和单机情况下的导入一样，配置文件不需要改变，直接像单机情况下使用即可。

当然后台的一些情况还是不同于单机版的，其实流程也比较清晰，我们暂时不用管怎么实现这些的，只管了解即可。



查询

只用修改连接的实例即可，其它部分和单机情况下一样。

```
String urlString = "zoo1:2181,zoo2:2181,zoo3:2181";
SolrClient solrClient = new CloudSolrClient(urlString);
```