**基于Lucene的分析与应用**

**需求规格说明书**

Version 2.0

小组成员：

刘宏宇

滕延林

顾泽鹏

杨帆

周晓懿

**版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 变更时间 | 修改人 | 审核人 | 备注 |
| 1.0 | 2016/03/16 | 滕延林 | 刘宏宇 顾泽鹏 杨帆 | 初稿 |
| 1.01 | 2016/03/20 | 滕延林 | 刘宏宇 顾泽鹏 杨帆 | 针对老师提出的问题进行修改 |
| 1.02 | 2016/04/03 | 刘宏宇 | 滕延林 顾泽鹏 杨帆 | 针对老师提出的问题以及网评结果进行修改 |
| 1.03 | 2016/04/09 | 顾泽鹏 | 滕延林 刘宏宇 杨帆 周晓懿 | 根据组间互评结果以及老师课堂的点评进行了修改 |
| 2.0 | 2016/05/04 | 杨帆  滕延林 | 刘宏宇 顾泽鹏 周晓懿 | 根据问题列表的所有问题，4.4数据需求部分进行修改、添加场景需求说明；添加第6章 实现说明部分 |
| 2.1 | 2016/05/09 | 滕延林 杨帆 | 刘宏宇 顾泽鹏 周晓懿 | 在版本变更记录中的备注中添加更详细的记录方案；第6章修改为实现需求和实现方案，并添加部分实现需求方面的内容 |
|  |  |  |  |  |

目录

[图片目录 5](#_Toc450574582)

[1 前言 6](#_Toc450574583)

[1.1 目的 6](#_Toc450574584)

[1.2 系统概述 6](#_Toc450574585)

[1.3 文档概述 7](#_Toc450574586)

[1.4 术语和缩略语 7](#_Toc450574587)

[2 引用文档 9](#_Toc450574588)

[3 功能需求 9](#_Toc450574589)

[3.1 软件功能分析 9](#_Toc450574590)

[3.2 需求识别 10](#_Toc450574591)

[3.2.1 用户与系统交互 10](#_Toc450574592)

[3.2.2 系统生成索引 10](#_Toc450574593)

[3.2.3 合并索引 10](#_Toc450574594)

[3.2.4 执行检索 11](#_Toc450574595)

[3.2.5 分词 11](#_Toc450574596)

[3.3 用例图 12](#_Toc450574597)

[3.4 RUCM模型 12](#_Toc450574598)

[3.4.1 更新文件 13](#_Toc450574599)

[3.4.2 添加文件 13](#_Toc450574600)

[3.4.3 修改文件 14](#_Toc450574601)

[3.4.4 删除文件 14](#_Toc450574602)

[3.4.5 更新索引 15](#_Toc450574603)

[3.4.6 检索文本文件 16](#_Toc450574604)

[3.4.7 检索图片文件 17](#_Toc450574605)

[3.4.8 分解查询请求 17](#_Toc450574606)

[3.4.9 计算相似度 18](#_Toc450574607)

[4 非功能性需求分析 18](#_Toc450574608)

[4.1 兼容性 18](#_Toc450574609)

[4.2 可修改性 19](#_Toc450574610)

[4.3 高效性 19](#_Toc450574611)

[4.4 数据需求 19](#_Toc450574612)

[4.4.1 数据格式 19](#_Toc450574613)

[4.4.2 数据来源 19](#_Toc450574614)

[4.4.3 数据规模 20](#_Toc450574615)

[4.4.4 数据特点 20](#_Toc450574616)

[4.5 场景需求 20](#_Toc450574617)

[4.5.1 基于Lucene的检索引擎的应用场景 20](#_Toc450574618)

[4.5.2 场景的存在的问题 21](#_Toc450574619)

[5 运行要求 21](#_Toc450574620)

[5.1 硬件要求 21](#_Toc450574621)

[5.2 软件要求 21](#_Toc450574622)

[5.3 用户界面需求 21](#_Toc450574623)

[6 实现需求和实现说明 22](#_Toc450574624)

[6.1中文分词 22](#_Toc450574625)

[6.2图像特征选择 23](#_Toc450574626)

[6.2.1 图像特征分析 23](#_Toc450574627)

[6.2.2 图像特征实验对比 23](#_Toc450574628)

[6.3图像特征提取 27](#_Toc450574629)

[6.4数据预处理 27](#_Toc450574630)

[6.5距离选择 27](#_Toc450574631)

[6.6网站搭建 28](#_Toc450574632)

[6.7 Hash排序 28](#_Toc450574633)

图片目录

图1- 1搜索应用程序和 Lucene 之间的关系 7

[图3- 1用例图 12](#_Toc450132076)

[图3- 2更新文件 13](#_Toc450132077)

[图3- 3添加文件 13](#_Toc450132078)

[图3- 4修改文件 14](#_Toc450132079)

[图3- 5删除文件 14](#_Toc450132080)

[图3- 6更新索引 15](#_Toc450132081)

[图3- 7检索文本文件 16](#_Toc450132082)

[图3- 8 检索图片文件 17](#_Toc450132083)

[图3- 9分解查询请求 17](#_Toc450132084)

[图3- 10计算相似度 18](#_Toc450132085)

# 前言

## 目的

以Lucene开源框架及相关资料为输入，分析软件设计需求，结合软件工程综合实验具体要求，输出软件需求规格说明书，作为设计开发的依据并指导后续的开发工作。

## 系统概述

Lucene 是一个基于 Java 的全文信息检索工具包，它不是一个完整的搜索应用程序，而是为你的应用程序提供索引和搜索功能。Lucene 目前是 Apache Jakarta 家族中的一个开源项目。也是目前最为流行的基于 Java 开源全文检索工具包。

目前已经有很多应用程序的搜索功能是基于 Lucene 的，比如 Eclipse 的帮助系统的搜索功能。Lucene 能够为文本类型的数据建立索引，所以你只要能把你要索引的数据格式转化的文本的，Lucene 就能对你的文档进行索引和搜索。比如你要对一些 HTML 文档，PDF 文档进行索引的话你就首先需要把 HTML 文档和 PDF 文档转化成文本格式的，然后将转化后的内容交给 Lucene 进行索引，然后把创建好的索引文件保存到磁盘或者内存中，最后根据用户输入的查询条件在索引文件上进行查询。不指定要索引的文档的格式也使 Lucene 能够几乎适用于所有的搜索应用程序。

图 1 表示了搜索应用程序和 Lucene 之间的关系，也反映了利用 Lucene 构建搜索应用程序的流程：

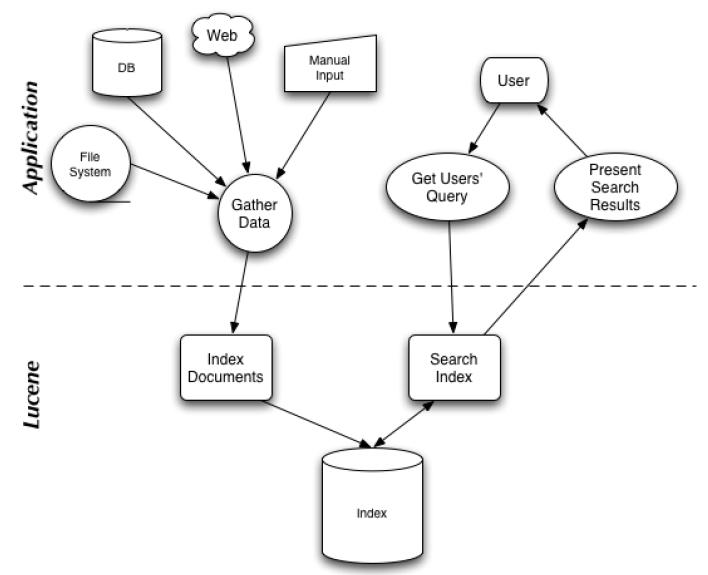


图1- 1搜索应用程序和 Lucene 之间的关系

## 文档概述

文档用途：本文档主要是介绍Lucene系统需求及规格说明。 主要内容：

⮚以用例图、状态图的形式给出 Lucene系统功能需求的分解结构，并对用例模型中的参与者和用例进行详细的描述，其中主要包括软件系统的用 例模型、系统的核心流程等；

⮚使用 RUCM 模型对功能需求进行建模；

⮚描述了与此次系统实施相关的硬件环境的一些要求；

⮚描述了与此系统实施相关的软件环境的要求；

## 术语和缩略语

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 术语 | 英文 | 说明 |
| 1 | UCM | UCM | 用例建模 |
| 2 | RUCM | RUCM | 限制性用例模型 |
| 3 | 索引 | Index | 在 Lucene 中一个索引是放在一个文件夹中 |
| 4 | 段 | Segment | 一个索引可以包含多个段，段与段之间是独立的，添加新文档可以生成新的段，不同的段可以合并。 |
| 5 | 文档 | Document | 文档是我们建索引的基本单位，不同的文档是保存在不同的段中的，一个段可以包含多篇文档。 |
| 6 | 域 | Field | 一篇文档包含不同类型的信息，可以分开索引，比如标题，时间，正文，作者等，都可以保存在不同的域里。 |
| 7 | 前缀后缀规则 | Prefix+Suffix | 所谓前缀后缀规则，即当某个词和前一个词有共同的前缀的时候，后面的词仅仅保存前缀在词中的偏移，以及除前缀以外的字符串(称为后缀)。 |
| 8 | 差值规则 | Delta | 所谓差值规则(Delta)就是先后保存两个整数的时候，后面的整数仅仅保存和前面整数的差即可。 |
| 9 | 词元 | Token | 将文档分词，并且去除标点符号和停词后，得到的一个个单独的单词。 |
| 10 | 词 | Term | 经Token经过过滤后，得到的小写、词根形式的单词。 |
| 11 | 停词 | Stop word | 一种语言中最普通的的一些单词，由于没有特殊的意义，因而大多数情况下不能成为搜索的关键词，例如“this”， “a”， “the”等。 |
| 12 | 分词组件 | Tokenizer | 将文档文本进行分词的组件。 |
| 13 | 语言处理组件 | TokenFilter | 将Token串进行过滤的组件。 |

# 引用文档

* 《Lucene原理与代码分析完整版》

# 功能需求

## 软件功能分析

Lucene 软件包的发布形式是一个 JAR 文件，下面我们分析一下这个 JAR 文件里面的主要的 JAVA 包，使读者对之有个初步的了解。

* Package: org.apache.lucene.document这个包提供了一些为封装要索引的文档所需要的类，比如 Document, Field。这样，每一个文档最终被封装成了一个 Document 对象。
* Package: org.apache.lucene.analysis这个包主要功能是对文档进行分词，因为文档在建立索引之前必须要进行分词，所以这个包的作用可以看成是为建立索引做准备工作。
* Package: org.apache.lucene.index这个包提供了一些类来协助创建索引以及对创建好的索引进行更新。这里面有两个基础的类：IndexWriter 和 IndexReader，其中 IndexWriter 是用来创建索引并添加文档到索引中的，IndexReader 是用来删除索引中的文档的。
* Package: org.apache.lucene.search这个包提供了对在建立好的索引上进行搜索所需要的类。比如 IndexSearcher 和 Hits, IndexSearcher 定义了在指定的索引上进行搜索的方法，Hits 用来保存搜索得到的结果。

## 需求识别

### 用户与系统交互

用户对于文件系统有读/写操作，读操作包括对于文件的读取与查询。读文件时需要知道文件的路径，查找时，可以输入关键字，时间，作者，标题等信息进行查找。写操作包括文件的添加，删除与修改。系统可以进行文件更新操作，当用户对于文件系统进行写操作时，系统会自动更新文件，同时修改相应的索引。但用户查询文件时，系统会进行相似度计算，从文件系统中检索，给出合理的排序，反馈给用户。缓存管理的任务贯穿系统运行的始终。

### 系统生成索引

系统的主要任务是索引更新，文件系统更新，缓存管理好相似度计算。在索引更新中，还有涉及到字段合并的操作。文件系统更新对于与用户对于文件系统的修改，包括文件的添加、修改和删除。缓存管理是为了提升系统的效率并且使系统的高负载情况下不会崩溃，增强系统鲁棒性。对于不同类型的数据有不同的策略，主要包括char,byte,int三种。相似度计算用于系统检索文件。当用户提出检索请求时，系统进行该操作。相似度计算首先要进行分词，并维护一个倒排词表来比较文件的相似度，给出一个合理的排序，越符合用户需求的文件会排在越前面。

### 合并索引

索引段的合并模块(merge\_module)

从索引段队列中选择待合并的段，根据具体情况确定合并策略，并合理地调度任务。

合并过程最重要的是三部分：

1. 一是选择哪些段应该参与合并，这一步由Segment selection 来决定

2. 二是选择什么样的合并策略能够取得最优性能，这一步由MergePolicy 来决

3. 三是调度合并段的过程，这一步由MergeScheduler 来执行

### 执行检索

检索过程主要分为以下几个步骤：

1. 用户输入要检索的词语，比如 “apple not iphone”；
2. 打开索引文件，并将索引文件的相关信息读入并对查询分析器进行初始化；
3. 根据用户输入的词语，分析其中的逻辑；该例子就是要检索出不包括iphone的apple；
4. 根据分析结果，进行检索。
5. 对检索的结果进行打分排序；
6. 返回最终的检索结果。

### 分词

被索引的文档在建立索引之前，首先要经过分词器的处理。其目的是，将文档分解为一系列的单词，从而更方便的建立索引。分词器主要分为两个部分，即分词组件和语言处理组件。其用例图如下所示。

分词组件的主要功能是将输入的文本分割为一个个词元组，即一系列Token。在这个过程中，同时还要去除标点符号和停词。停词是一种语言中最普通的的一些单词，由于没有特殊的意义，因而大多数情况下不能成为搜索的关键词，例如“this”， “a”， “the”等。在经过分词组件的处理后，文本中大部分的没有意义的信息被去除了。

## 用例图

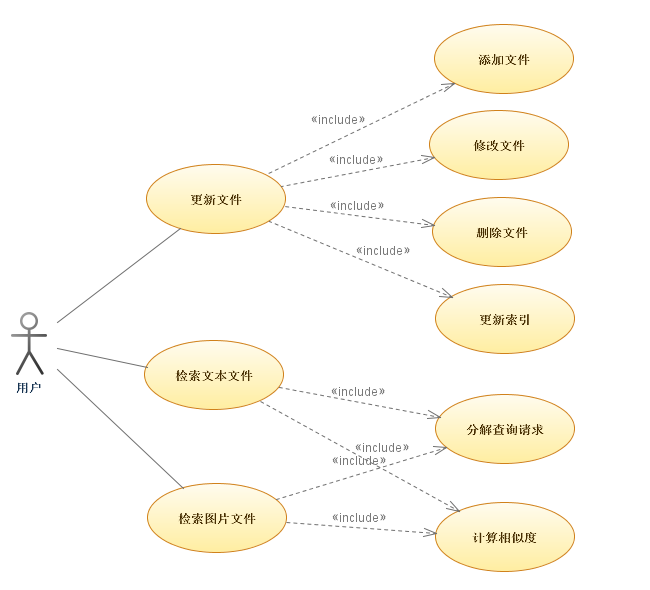


图3- 1用例图

## RUCM模型

RUCM 即限制性用例建模。 它的目标是：

* 使 UCMs 更加可理解并且更精确。
* 从 UCMs 自动生成分析模型。

RUCM 有以下两部分组成：

* 一个用于系统组织 UCSs 的用例模板。
* 限制用户写 UCSs 的一系列规则。

通过 RUCM 模型能够对用例进行规范的描述，接下来将使用 RUCM 模型描述图 3-1中的用例。

### 更新文件



图3- 2更新文件

### 添加文件



图3- 3添加文件

### 修改文件



图3- 4修改文件

### 删除文件



图3- 5删除文件

### 更新索引



图3- 6更新索引

### 检索文本文件



图3- 7检索文本文件

### 检索图片文件



图3- 8 检索图片文件

### 分解查询请求



图3- 9分解查询请求

### 计算相似度



图3- 10计算相似度

# 非功能性需求分析

## 兼容性

作为一个跨平台的全文搜索引擎，系统应该具有强大的兼容性：

1. 操作系统兼容性

理想的软件应该具有与平台无关性，因此基于Lucene开发的搜索程序应该具有运行于不同操作系统的能力，需要定义独立于平台的索引格式，还要考虑前端和后端操作系统的可选择性。

1. 异构数据库兼容性

搜索引擎索引的数据需要数据库系统的支持，因此程序需要考虑其对不同数据库平台的支持能力，兼容异构数据库，使得不同数据库能够共享建立的索引文件。

1. 新旧数据转换

对于软件升级后可能定义的新的数据格式或者文件格式，提供新旧数据转换的功能。提供对原来格式的支持及更新，使得原来的用户记录能够被集成，在新的格式下依然可用。这还涉及了转换过程中数据的完整性与正确性的验证问题。

## 可修改性

作为一种检索系统框架，Lucene并不直接提供系统的实现，而仅仅是系统框架而已。为了进行高效的开发。要求程序要具备一种简明、方便的构架与函数接口来方便用户的使用。此外，为了使运行更加高效，需要不断学习，引入新的技术、算法或辅助措施，因此程序必须具备很好的可修改性（ modifiability）。

## 高效性

本程序作为检索系统，需要支持对大规模数据的索引及搜索，因此需要具有很高的效率。这主要体现在两个方面：

1. 高效实现大规模数据实时索引入库

作为一个全文检索系统，如果入库数据占据了大量时间，那么必然影响了检索系统的实时性。因此程序需要高效完成大规模数据的索引入库过程。

1. 迅速响应查询条件并返回结果

衡量搜索引擎性能的一个重要指标就是响应时间，因此程序需要能够在很短的时间内响应查询请求并返回检索结果

## 数据需求

### 数据格式

文本数据为TXT格式文本。

图片数据为JPG格式。

### 数据来源

文字检索功能的样例数据来自于互联网上下载的金庸全集。小说的语言环境丰富，包括了各种短语，成语，句式，作为一个检索源来说是一个丰富的语料库。越大的文本库越能够检验系统的性能，包括准确率，排序效果，时间等；已有的文本是检索的目标，根据已有的文本进行优化，可以构造更加适用于该文本的检索系统。

图片检索的样例数据爬取自新浪微博。

### 数据规模

文字检索实验数据大小为34M，共1050万字左右。

图片检索实验数据大小为7.4G，共50000张。

### 数据特点

1. Lucene默认分词器均为英文分词；而本实验中文本检索的实验数据均为中文。
2. Lucene默认功能中不具有图片检索的能力；图片检索的实验数据需要进行特征提取。
3. 图片实验数据的索引与默认索引可能有一定区别。

## 场景需求

### 基于Lucene的检索引擎的应用场景

Lucene是一个高效的，基于Java的全文检索库。

全文检索是将存储于数据库中整个文本库、整本书、整篇文章中的任意内容信息查找出来的检索。它可以根据需要获得全文中有关章、节、段、句、词等信息，也就是说类似于给整本书的每个字词添加一个标签，也可以进行各种统计和分析。例如，它可以很快的找到“《红楼梦》一书中“林黛玉”在哪里出现？”的问题。

本实验的场景为用户针对样例（金庸全集与图片库）进行全文检索。特别的，在中文的全文检索中，还有分词的需求。在英文检索中，以空格或标点分隔开的每一个词都是一个单独的词元结构。然而，由于中文的语法结构比较复杂多变，有词语，短语，成语，俗语等各种组成，这使得分词需要考虑多种情况才能适应用户的搜索需求，给分词带来了更多的要求。

图像检索是基于图像的内容，在库中检索出与该张图片相近的图片，并按照相似度进行排序、展示。

### 场景的存在的问题

1. 保证全文检索的查全率；（查全率=被检出相关信息量/相关信息总量（%））
2. 保证全文检索的查准率；（查准率=被检出相关信息量/被检出信息总量（%））
3. 全文检索的效率；由于可能有大量检索的要求，需要保证全文检索的效率；如果Lucene的结构对效率产生了限制，则需要对Lucene进行性能优化。
4. 系统如何确定两张图像是否相似
5. 图像检索时，对库内所有图像进行比对时，效率问题

# 运行要求

## 硬件要求

* CPU:E7500以上
* 内存：4G内存及以上
* 硬盘：100G硬盘及以上

## 软件要求

* 操作系统：Windows7版本及以上
* 编译环境：JRE6.0版本及以上。

## 用户界面需求

简单实用，容易操作

# 实现需求和实现说明

基于lucence实现了两个上层应用，分别是中文检索系统与图像检索系统。中文检索系统与原有lucence的不同是中文分词系统中包含中文分析器。而图像检索系统使用lucence进行存储特征，支持对多种格式的图像进行检索。

为了方便展示，中文检索系统与图像检索系统都利用B/S系统。前台采用jsp技术，后台采用servlet技术对前台的query进行处理，调用检索系统，并将结果发送给前台浏览器页面进行展示。

检索系统分为两部分。第一部分是对待检数据进行入库处理。比如中文检索系统。首先要准备待检数据。目前该系统中准备的数据是金庸的武侠小说。对这些武侠小说进行入库，生成lucence的索引。而图像检索系统的待检数据是从微博上爬取的50000张图像数据。图像的入库过程比较复杂，首先要对所有图像进行特征的提取，然后将所有特征进行hash处理，生成hash码，将图像的特征和hash码都存储到lucence中。第二部分是检索过程。中文检索过程是将query与索引中的信息进行匹配，利用倒排索引，检索出query在哪个文档中，并将检索出的结果进行打分、排序。图像检索过程是对待检图像进行特征提取，然后对其进行hash处理，并在lucence库中取出与该hash码前24位相同的图像的特征，然后对根据特征之间的距离进行重排序。

为实现中文检索系统与图像检索系统，主要采用的技术与方法如下。

## 6.1中文分词

原始的lucence只能支持英文检索，如果想对中文进行处理，那就要需要支持对中文的分词。

因为中文分词是一个非常复杂的问题，在短短的时间内无法取得比较好的结果，我们决定使用已有的方案。

中文分词技术是采用中文分词器mmseg4j。mmseg4j 用 Chih-Hao Tsai 的 MMSeg算法(<http://technology.chtsai.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。

MMSeg 算法有两种分词方法：Simple和Complex，都是基于正向最大匹配。Complex 加了四个规则过虑。官方说：词语的正确识别率达到了 98.41%。mmseg4j 已经实现了这两种分词算法。目前 complex 1200kb/s左右, simple 1900kb/s左右, 但内存开销了50M左右。

## 6.2图像特征选择

在图像检索系统中，为了取得比较好的检索效果，图像的特征选择是非常关键的技术。图像特征对算法计算结果的好坏有巨大的影响。因此，要选择一种合适的图像特征。需要对多种特征进行对比分析。下面对FCTH、CEDD、Gabor三种特征进行分析，并通过实验对这三种特征进行选择。

### 6.2.1 图像特征分析

（1）FCTH特征。FCTH是192位整数组成的向量，它是通过3个模糊单元的组合求得的结果，使用了小波变换等方法，主要对图像的颜色和纹理进行提取。

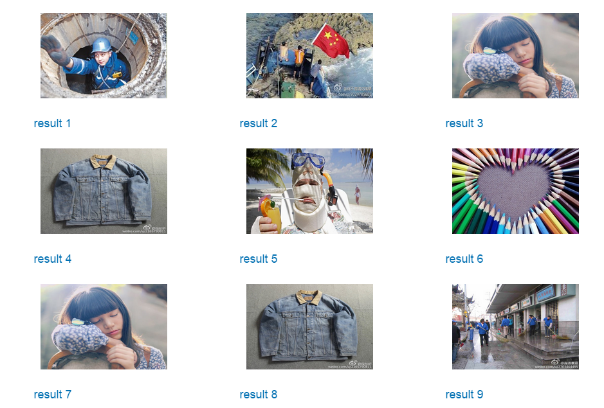
（2）CEDD特征。CEDD特征是144位整数组成的向量，如同FCTH特征一样，对图像的颜色和纹理进行提取，不过提取方式不同。CEDD特征是通过色彩空间转换、模糊过滤器等方式进行提取。

（3）Gabor特征。Gabor特征是利用Gabor小波变换进行提取的特征。该特征对图像的边缘比较敏感，但是对光照变化不敏感。该特征对于一些特征识别等有着巨大的优势。

### 6.2.2 图像特征实验对比

为了对比特征的好坏，对多种特征进行了对比试验。由于存在语义鸿沟，无法使用定量的方式进行统计，只能通过肉眼的直观感受对图像的特征进行判断。

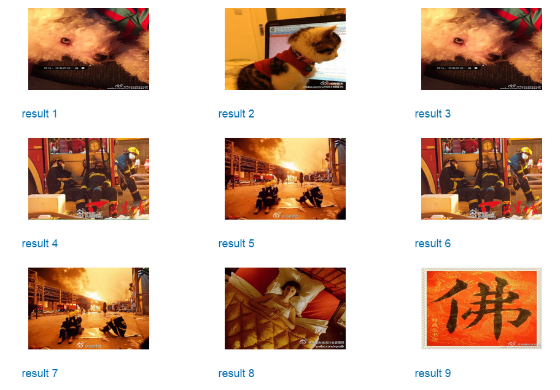
(1) FCTH特征检索结果如图6.1所示。从a-f六张图像，可以看出，FCTH检索效果一般。在b图中，第3张和第7张是相似的，d图中，第3张和第7张是相似的，e图中，第3、4、5、8张图像是相似的。而f图中，所有图像的色调都是相同的。FCTH特征，效果能够接受。

(a) (b)

(c) (d)

(e) (f)

图6. 1 FCTH特征检索部分示意图

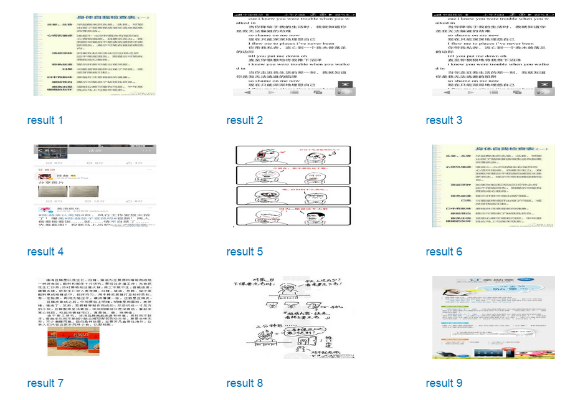
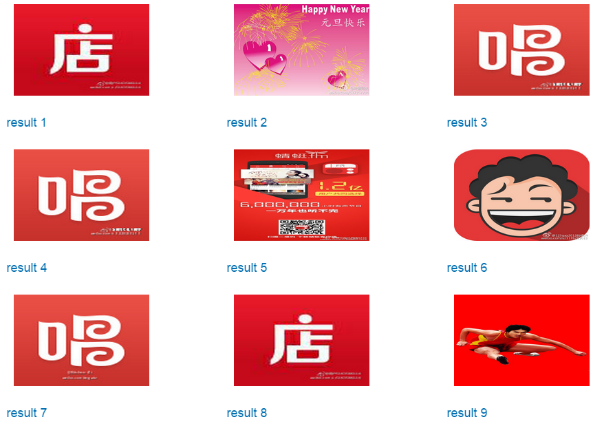
(2)CEDD特征检索结果图6.2所示。从a-f六张图我们可以看出，CEDD特征的检索效果比较好。c图中所有图像都是相似的，b、e、d、f中，所有色调都是一致的，而a图中，只有一张色调不一致，其他看起来都相似。

(a) (b)

 ****

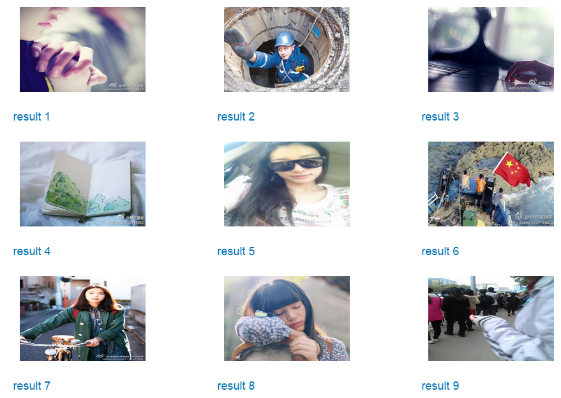
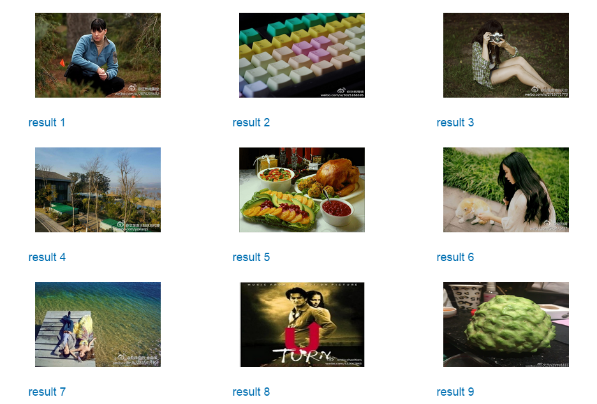
(c) (d)

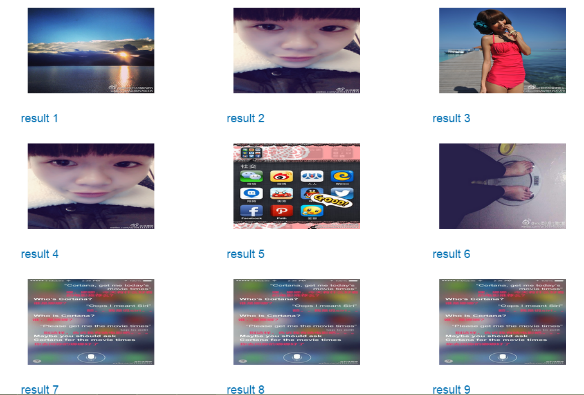
(e) (f)

图6.2 CEDD特征检索部分示意图

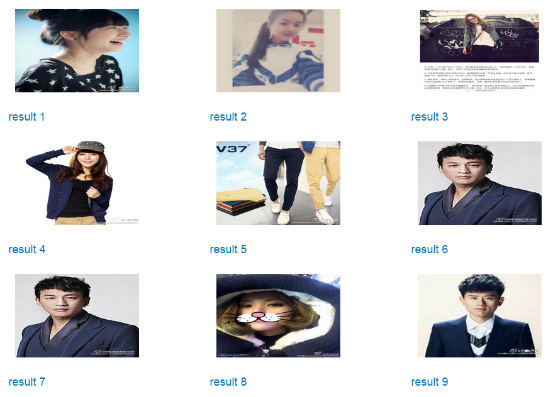
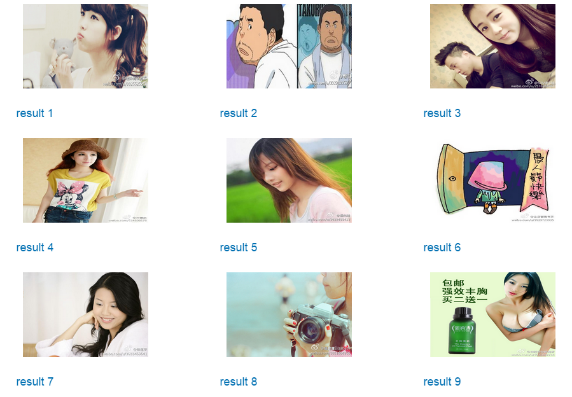
(3)Gabor特征检索结果如图6.3所示。从检索结果来看，c图中有三张相似的图像，但是与其他图像不相似。d、e图中图像看起来也不是很一致。a图从颜色来看，还是有点一致的，但是从纹理上看，就不一致了。整体来看，效果并不是很好。



(a) (b)



(c) (d)



(e) (f)

图6.3 Gabor特征检索部分示意图

由于篇幅限制，每种特征的只是列举出了6个query的检索结果，每个结果截取了9张图。从上面的结果很明显可以看出， Gabor特征检索效果最差，而CEDD特征检索效果最好。另外，CEDD特征是144位的整数，而FCTH特征为192位的整数，从计算复杂度上来说，CEDD特征要简单。因此，确定CEDD特征为最终选择。

## 6.3图像特征提取

图像特征的提取也是一个复杂的问题。我们决定采用已有的全局特征的提取方法。特征提取技术采用lire中提取特征的方法。

## 6.4数据预处理

在中文检索系统中，需要对待检数据进行处理。由于lucence系统只能定位到哪一篇文档，并不能准确查找到特定的句子甚至词语。与此同时，为了提高检索的准确度，需要待检数据进行处理。我们从网上获取的数据是一个txt文件。比如金庸先生的《射雕英雄传》，存储于一个txt文件中。由于mmseg4j中文分词器只能处理UTF-8数据，我们首先对所有文件进行转码处理。然后将其按照章节进行划分为多个txt文件。之后，为了提高检索效果，又对每个章节进行分片。

## 6.5距离选择

在图像检索中，比较两张图片是否相似，我们采用的方式是比较两张图片的特征之间的距离。然而，距离有多种选择。而测度两个图像特征之间的常用的距离有欧几里得距离、余弦距离、Tanimoto距离等等。下面将对常用的这几种距离进行简要介绍。

欧几里得距离又称为欧氏距离，它的的计算公式为,相当于高维空间内表示的点与点之间的距离。这是一种非常容易理解的算法，就是把二维或三维空间中点之间的距离扩充到N维空间，而缺点就是没有考虑各个分量之间的相关性，完全将各个分量独立开来。

余弦距离的计算公式为, 就是两个向量之间的夹角的余弦值。与欧式距离相比，余弦距离对方向上比较敏感，而不能表达绝对距离的差异，余弦距离更多地应用在区别相似度与差异上。

Tanimoto距离又被称为广义Jaccard距离，它的计算公式为 **。**优点是比较适用于高维数据，缺点是计算比较复杂。

为了确定哪种距离比较合适，应该在图像上分别使用这三种距离进行测度，然后根据结果进行比较。经过多组的实验表明，欧式距离的效果最差，但是花费的时间最少。余弦距离的效果优于欧式距离而差于Tanimoto距离，花费的时间比欧式距离略多。Tanimoto距离是效果最好的，但是计算Tanimoto距离消耗的时间最多。经过权衡，最终选择Tanimoto距离来衡量点之间的距离。

## 6.6网站搭建

为了方便展示，检索系统都利用B/S系统。前台采用jsp技术，后台采用servlet技术对前台的query进行处理，调用检索系统，并将结果发送给前台浏览器页面进行展示。

为了提高前台的美观性，采用bootstrap的css和js库。目前网站已经上线。图像检索系统的网址是<http://buaatyl.cn/lucenceWeb/imageSearch.jsp>，中文检索系统的网址是<http://buaatyl.cn/lucenceWeb/textSearch.jsp>。由于服务器的内存是1G，带宽是1M。图片的下载速度比较慢。

## 6.7 Hash排序

中文检索系统中有自带的倒排索引对检索进行加速，而图像检索中并不适合这种方式。如果在检索过程中，将待检图像与库中所有的图像进行对比，那么算法复杂度是O（n）。这并不是我们能够接受的。为了提高检索速度，我们采用了hash的方式。

在图像的入库过程，首先要对所有图像进行特征的提取，然后将所有特征进行hash处理，生成hash码，根据hash进行排序，将图像的特征和hash码都存储到lucence中。图像检索过程是对待检图像进行特征提取，然后对其进行hash处理，并在lucence库中取出与该hash码前24位相同的图像的特征，然后对根据特征之间的距离进行重排序。

通过hash的方式，能够大大提高检索的速度。