基于 Lucene 的分析与应用

项目计划

Version 1.0

小组成员：

刘宏宇

滕延林

顾泽鹏

杨帆

版本变更记录

版本	变更时间	修改人	审核人	备注
1.0	20160316	滕延林	刘宏宇 顾泽鹏 杨帆	初稿
1.01	20160320	滕延林	刘宏宇 顾泽鹏 杨帆	针对老师提出的问题进行修改

目录

1 前言

1.1 目的

为了便于协调组内成员进行后期的工作，对项目进行跟踪和监控，对任务的进度进行安排与调控，故对后期工作进行计划。提出需求，指导后续工作。

1.2 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：

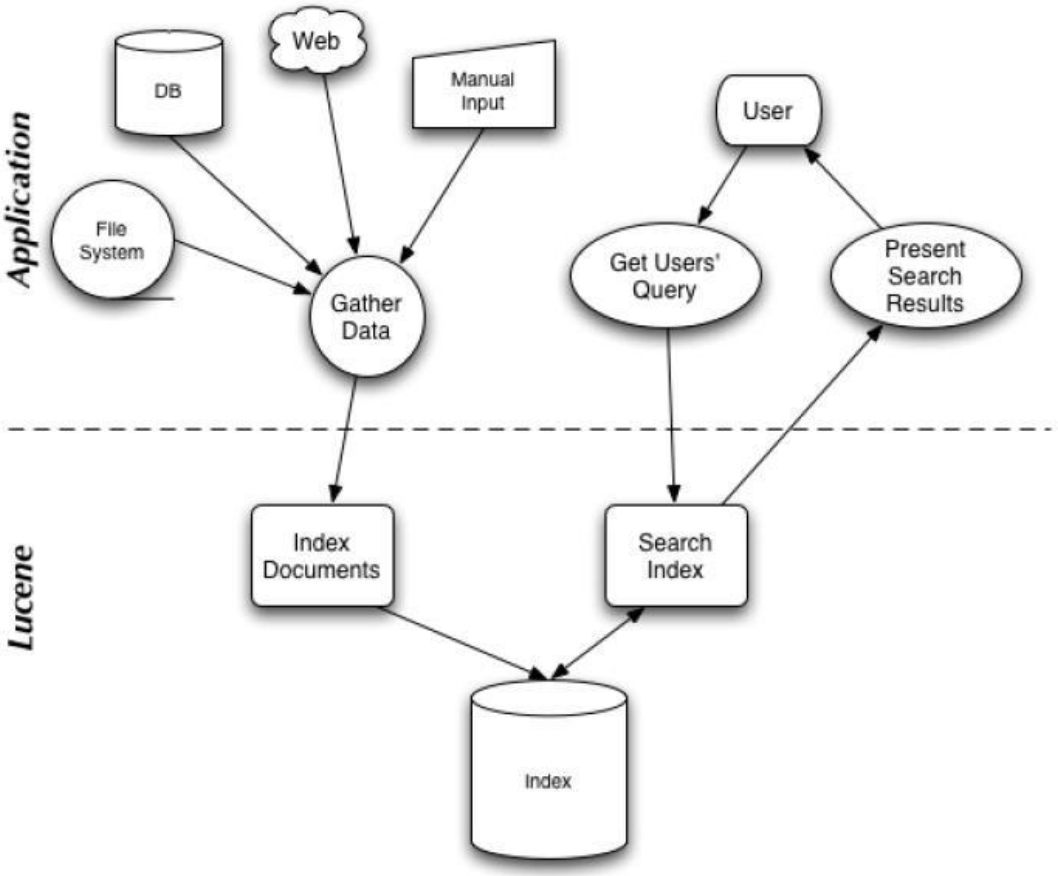


图 1 搜索应用程序和 Lucene 之间的关系

1.3 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；

1.4 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。

2 引用文档

Lucene 原理与代码分析完整版

3 需求分析

3.1 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。

3.2 非功能性需求分析

3.2.1 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。

2. 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。

3. 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。

3.2.2 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）。

3.2.3 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。

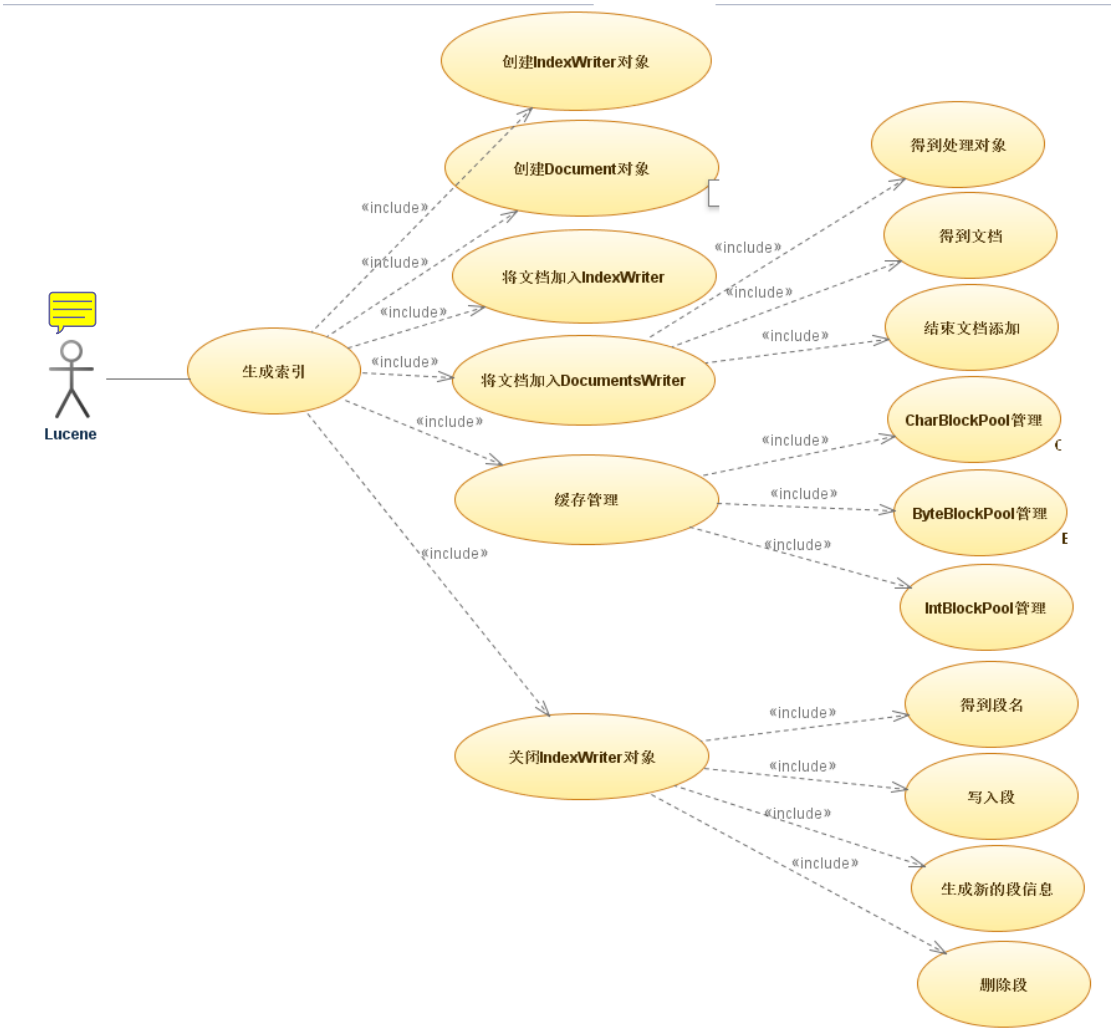
2. 迅速响应查询条件并返回结果

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

3.3 需求识别

将业务需求分解为功能性需求，同时考虑到非功能性需求，最终得到的 Lucene 的用例图如下图所示。

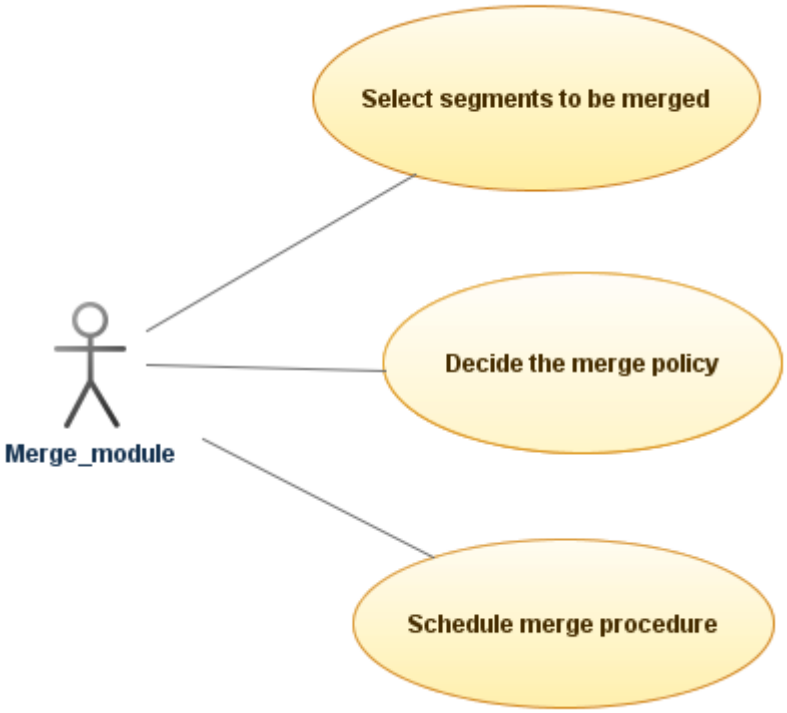
3.3.1 索引生成



3.3.2 索引段的合并

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

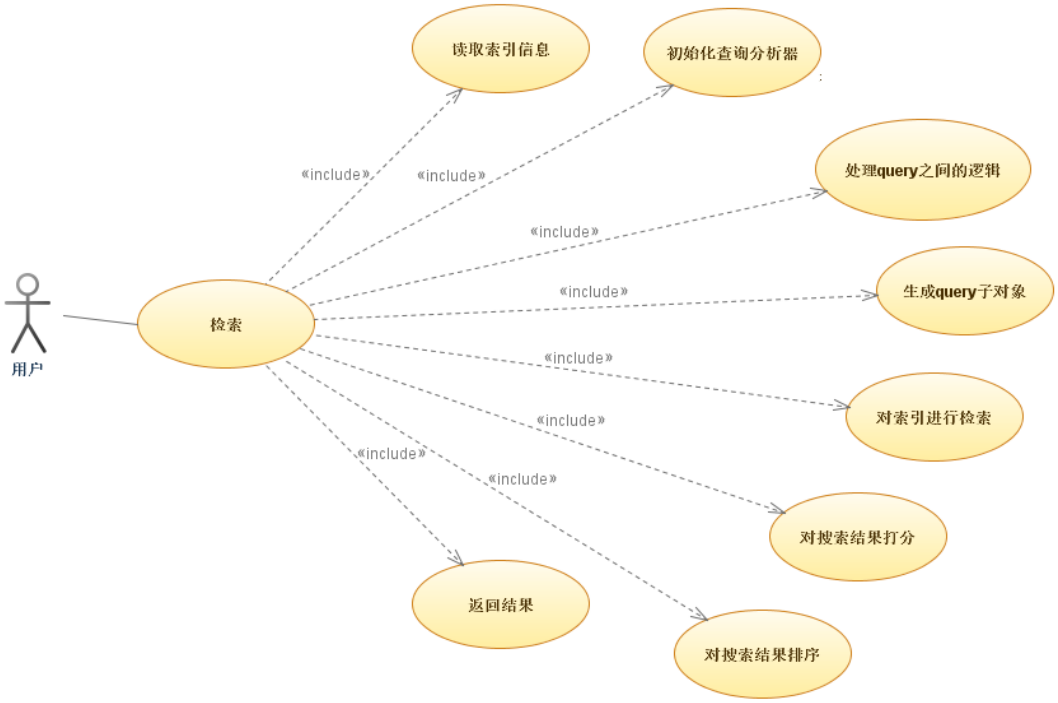


3.3.3 检索过程

检索过程：

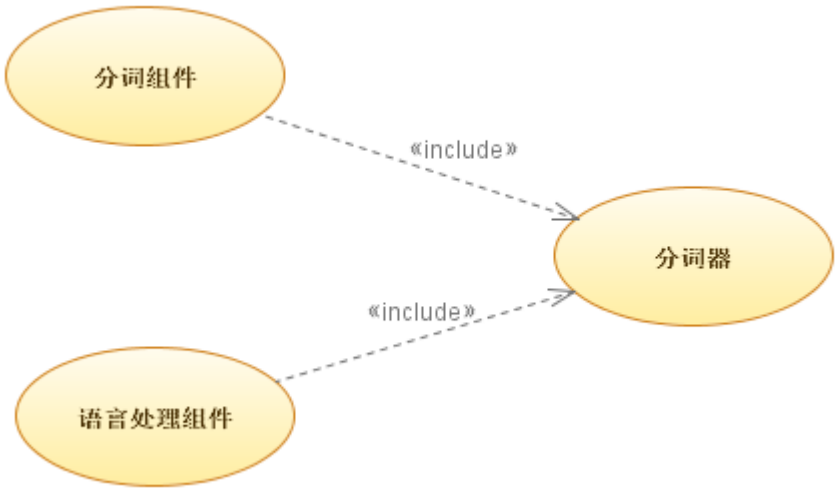
索引的过程主要分为以下几个步骤

首先用户输入要检索的词语，比如“apple not iphone”，之后打开索引文件，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。然后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。根据分析结果，进行检索。之后对检索的结果进行打分排序，返回最终的检索结果。



3.3.4 分词器 Analyzer

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。其 RUCM 模型如下图所

3.4RUCM 模型

RUCM 即限制性用例建模。它的目标是：

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

RUCM 有以下两部分组成：

1. 一个用于系统组织 UCSs 的用例模板。
2. 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 1 中的用例。

3.4.1 生成索引

Use Case Specification

Use Case Name	生成索引
Brief Description	生成文本类型文件的索引，以便于管理
Precondition	系统初始化完成
Primary Actor	Lucene
Secondary Actors	None
Dependency	**INCLUDE USE CASE** 创建IndexWriter对象, **INCLUDE USE CASE** 创建Document对象, **INCLUDE USE CASE** 将文档加入IndexWriter, **INCLUDE USE CASE** 将文档加入DocumentsWriter, **INCLUDE USE CASE** 缓存管理, **INCLUDE USE CASE** 关闭IndexWriter对象
Generalization	None

Basic Flow (Untitled) ▼	**Steps**	
	1	文件系统发生修改
	2	**DO**
	3	**INCLUDE USE CASE** 创建IndexWriter对象
	4	**INCLUDE USE CASE** 创建Dcument对象
	5	**INCLUDE USE CASE** 将文档加入IndexWriter
	6	**INCLUDE USE CASE** 将文档加入DocumentsWriter
	7	**INCLUDE USE CASE** 关闭IndexWriter对象
	8	**UNTIL**新索引与当前文件系统匹配
	9	索引更新完成
	Postcondition	None

3.4.2 创建 IndexWriter 对象

Use Case Specification	
Use Case Name	创建IndexWriter对象
Brief Description	用于索引文档，合并段，保证索引完整性
Precondition	文件修改完毕，系统发出索引更新请求
Primary Actor	Lucene
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	获取需要索引的目录
	2	开启分词器
	3	申请IndexWriter的空间
	4	设置段合并因子
	Postcondition	可以创建Document

3.4.3 创建 Document 对象

Use Case Specification	
Use Case Name	创建Document对象
Brief Description	将不同格式的的文本文件统一封装成Document格式
Precondition	IndexWriter对象创建完成
Primary Actor	Lucene
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	获取文件目录
	2	将文件格式统一
	3	获取系统时间
	4	将文件内容与时间写入Document
	5	将Document加入Field
	Postcondition	可以将文档加入IndexWriter

3.4.4 将文档写入 IndexWriter

Use Case Specification	
Use Case Name	将文档加入IndexWriter
Brief Description	索引的更新过程
Precondition	Document创建完成
Primary Actor	Lucene
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	调用DocumentWriter
	2	将文档加入IndexWriter
	3	清空缓存
	Postcondition	可以将文档加入DocumentWriter

3.4.5 将文档加入 DocumentWriter

Use Case Specification	
Use Case Name	将文档加入DocumentsWriter
Brief Description	文件系统的更新
Precondition	文档已经加入IndexWriter
Primary Actor	Lucene
Secondary Actors	None
Dependency	INCLUDE USE CASE 得到处理对象, INCLUDE USE CASE 得到文档, INCLUDE USE CASE 结束文档添加
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	读取索引链
	2	INCLUDE USE CASE 得到处理对象
	3	INCLUDE USE CASE 得到文档
	4	INCLUDE USE CASE 结束文档添加
	Postcondition	IndexWriter可以关闭

3.4.6 关闭 IndexWriter 对象

Use Case Specification	
Use Case Name	关闭IndexWriter对象
Brief Description	索引更新完成，释放系统空间
Precondition	文档索引更新完成
Primary Actor	Lucene
Secondary Actors	None
Dependency	INCLUDE USE CASE 得到段名, INCLUDE USE CASE 写入段, INCLUDE USE CASE 生成新的段信息, INCLUDE USE CASE 删除段
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	INCLUDE USE CASE 得到段名
	2	INCLUDE USE CASE 写入段
	3	INCLUDE USE CASE 生成新的段信息
	4	INCLUDE USE CASE 删除段
	Postcondition	None

3.4.7 缓存管理

Use Case Specification	
Use Case Name	缓存管理
Brief Description	对于缓存的合理分配利用
Precondition	系统初始化完成
Primary Actor	Lucene
Secondary Actors	None
Dependency	INCLUDE USE CASE CharBlockPool管理, INCLUDE USE CASE ByteBlockPool管理, INCLUDE USE CASE IntBlockPool管理
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	DO
	2	INCLUDE USE CASE CharBlockPool管理
	3	INCLUDE USE CASE ByteBlockPool管理
	4	INCLUDE USE CASE IntBlockPool管理
	5	UNTIL 缓存分配满足需求
	Postcondition	None

3.4.8 索引段的合并模块

Use Case Specification	
Use Case Name	Select segments to be merged
Brief Description	Select segments from the queue and merge them
Precondition	The merge module is running
Primary Actor	Merge_module
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	The merge module scans the queue of segments to be merged.
	2	**IF** the queue is not empty **THEN**
	3	**DO** select the appropriate segments **UNTIL** the queue is empty.
	4	**ELSE**
	5	Terminate the procedure.
	6	**ENDIF**
	Postcondition	The segments are selecte.

Use Case Specification	
Use Case Name	Schedule merge procedure
Brief Description	Schedule merge procedure
Precondition	The merge module is running
Primary Actor	Merge_module
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow	Steps	
(Untitled) ▼	1	Specify the cost.
	2	Determine the priority of the procedure
	Postcondition	None

3.4.9 检索过程

Use Case Specification	
Use Case Name	检索
Brief Description	用户输入要检索的词，系统对该词进行检索
Precondition	索引构建完毕
Primary Actor	用户
Secondary Actors	None
Dependency	INCLUDE USE CASE 读取索引信息, INCLUDE USE CASE 初始化查询分析器, INCLUDE USE CASE 处理query之间的逻辑, INCLUDE USE CASE 生成query子对象, INCLUDE USE CASE 对索引进行检索, INCLUDE USE CASE 对搜索结果打分, INCLUDE USE CASE 对搜索结果排序, INCLUDE USE CASE 返回结果
Generalization	None

Basic Flow "main" ▼	Steps	
	1	读取索引信息
	2	初始化查询分析器
	3	处理query之间的逻辑
	4	生成query子对象
	5	对索引进行检索
	6	对搜索结果打分
	7	对搜索结果排序
	Postcondition	获得搜索结果

3.4.10 分词器

Use Case Specification	
Use Case Name	分词组件
Brief Description	对字符串进行分词
Precondition	待检索的文档已经输入
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	Steps	
	1	将文档分成一个个单独的单词
	2	去除标点符号
	3	去除停词
	4	将得到的词元组成为词元串
	Postcondition	过滤词元串

Global Alternative Flow (Untitled) ▼	分词出现异常	
	1	分词出现异常
	2	ABORT
	Postcondition	停止运行。

经过分词处理后的词元串流入语言处理组件中。语言处理组件对得到的词元做一些同语言相关的处理。对于英语，语言处理组件一般进行两项处理：将词元变为小写；将单词缩减或转变为词根形式。例如，将”cars”缩减为”car”，将”drove”转化为”drive”。语言处理组件的 RUCM 模型如下图所示。

Use Case Specification

Use Case Name	语言处理组件
Brief Description	对词元串进行过滤
Precondition	对文档进行分词
Primary Actor	None
Secondary Actors	None
Dependency	None
Generalization	None

Basic Flow (Untitled) ▼	**Steps**
	1 将词元变为小写
	2 将单词缩减为词根形式
	3 得到词
	Postcondition 创建词典

Global Alternative Flow (Untitled) ▼	Guard Condition
	1 分词出现异常
	2 **ABORT.**
	Postcondition 停止运行。

分词器整体的 RUCM 模型如下图所示。

Use Case Specification

Use Case Name	分词器
Brief Description	对文档进行分词
Precondition	被检索的文档已经输入
Primary Actor	None
Secondary Actors	None
Dependency	**INCLUDE USE CASE** 分词组件, **INCLUDE USE CASE** 语言处理组件
Generalization	None

Basic Flow (Untitled) ▼	**Steps**
	1 **INCLUDE USE CASE** Tokenizer
	2 对文档进行分词，得到Token串
	3 **INCLUDE USE CASE** TokenFilter
	4 对Token串进行过滤，得到Term
	Postcondition 输出TokenStream

Global Alternative Flow (Untitled) ▼	分词出现异常
	1 分词出现异常
	2 **ABORT**
	Postcondition 停止运行。

以上分词器的工作过程为通用的对于英文文本的处理过程。对于中文则需要更复杂的处理方式，例如，需要一定的算法先进行句子和词组的划分，再对词组和词语进行缩减与转化，处理流程因算法而异。