LUCENE 技术文档

说明：本文档在我们最大努力范围之内确保其正确性、实效性和可观性，但并不代表所有的观点都是正确的，而仅代表个人看法。如发现不当之处，请多指教，谢谢！

Lucene 技术说明

联系方式：XXXXXXXXXXX(直接敲本人就好了嘛)

1、技术概述

Lucene 是 apache 软件基金会 4 jakarta 项目组的一个子项目，是一个开放源代码的全文检索引擎工具包，但它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言）。

Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。Lucene 提供了一个简单却强大的应用程式接口，能够做全文索引和搜寻。在 Java 开发环境里 Lucene 是一个成熟的免费开源工具。就其本身而言，Lucene 是当前以及最近几年最受欢迎的免费 Java 信息检索程序库。人们经常提到信息检索程序库，虽然与搜索引擎有关，但不应该将信息检索程序库与搜索引擎相混淆。

Lucene 的使用是建立在庞大的索引库的基础上的，通常来说，我们在查找时并不直接到数据库去查找，而是直接从索引库中查找，因为索引库是是二进制格式的，所以检索速度是很快的。所以我们使用 Lucene 进行检索时，要先创建或者获取一个索引库。

2、使用优缺点

2.1 优点：

（1）索引文件格式独立于应用平台。Lucene 定义了一套以 8 位字节为基础的索引文件格式，使得兼容系统或者不同平台的应用能够共享建立的索引文

件。

（2）在传统全文检索引擎的倒排索引的基础上，实现了分块索引，能够针对新的文件建立小文件索引，提升索引速度。然后通过与原有索引的合并，达到优化的目的。

（3）优秀的面向对象的系统架构，使得对于 Lucene 扩展的学习难度降低，方便扩充新功能。

（4）设计了独立于语言和文件格式的文本分析接口，索引器通过接受 Token 流完成索引文件的创立，用户扩展新的语言和文件格式，只需要实现文本分析的接口。

（5）已经默认实现了一套强大的查询引擎，用户无需自己编写代码即可使系统可获得强大的查询能力，Lucene 的查询实现中默认实现了布尔操作、模糊查询（Fuzzy Search[11]）、分组查询等等。

面对已经存在的商业全文检索引擎，Lucene 也具有相当的优势。

首先，它的开发源代码发行方式（遵守 Apache Software License[12]），在此基础上程序员不仅仅可以充分的利用 Lucene 所提供的强大功能，而且可以深入细致的学习到全文检索引擎制作技术和面向对象编程的实践，进而在此基础上根据应用的实际情况编写出更好的更适合当前应用的全文检索引擎。在这一点上，商业软件的灵活性远远不及 Lucene。

其次，Lucene 秉承了开放源代码一贯的架构优良的优势，设计了一个合理而极具扩充能力的面向对象架构，程序员可以在 Lucene 的基础上扩充各种功能，比如扩充中文处理能力，从文本扩充到 HTML、PDF[13]等等文本格式的处理，编写这些扩展的功能不仅仅不复杂，而且由于 Lucene 恰当合理的对系统设备做了程序上的抽象，扩展的功能也能轻易的达到跨平台的能力。

最后，转移到 apache 软件基金会后，借助于 apache 软件基金会的网络平台，程序员可以方便的和开发者、其它程序员交流，促成资源的共享，甚至直接获得已经编写完备的扩充功能。最后，虽然 Lucene 使用 Java 语言写成，但是开放源代码社区的程序员正在不懈的将之使用各种传统语言实现（例如.net framework[14]），在遵守 Lucene 索引文件格式的基础上，使得 Lucene 能够运行在各种各样的平台上，系统管理员可以根据当前的平台适合的语言来合理的

选择。

2.2 缺点

1. Lucene 搜索算法不适合网格计算

Lucene 被写出来的时候硬件还没有很大的内存，多处理器也不存在。因此，索引结构是被设计成使用线性的内存开销很小的方式。即使我们花很长的时间来重写跨度查询算法，并使用多线程内容（使用双核处理器），但是基于迭代器的目录读取算法几乎不能实现。在一些罕见的场合你能做一些优化并能迭代一个索引通过并行方式，但是大多数场合这是不可能的。我们遇到的情况是，当我们有一个复杂的，超过 50+的内嵌跨度查询，CPU 还在空闲但 I/O 却一直忙碌，甚至在使用了 RAMDirectory.

2.一个关闭的 API 使得继承 Lucene 成为痛苦

在 Lucene 的世界中,它被称之为特性。当某些用户需要得到某些细节，方针是开放类。这导致了大多数的类都是包保护级别的，这意味着你不能够继承他们（除非在你创建的类似在同一个包下，这样做会污染客户代码）或者你不得不复制和重写代码。更重要的是，如同上面一点提到的，这个严重缺乏 OO 设计的结构，一些类应该被设为内部类却没有，匿名类被用作复杂的计算当你需要重写他们的行为。关闭 API 的理由是让代码在发布前变得整洁并且稳定。虽然想法很光荣，但它再一次让人感到痛苦。因为如果你有一些代码和 Lucene 的主要思路并不吻合，你不得不经常回归 Lucene 的改进到你自己的版本直到你的补丁被接受。

3.Lucene 并非良好设计

Lucene 有一个非常糟糕的 OO 设计。虽然有包，有类的设计，但是它几乎没有任何设计模式。这让我想起一个由 C(++)开发者的行为，并且他把坏习惯带到了 java 中。这造成了，当你需要自定义 Lucene 来满足你的需求(你将来必定会遇到这样的需求)，你必须面对这样的问题。例如：

- 几乎没有使用接口。查询类（例如 BooleanQuery,SpanQuery,TermQuery…）都是一个抽象类的子类。如果你要添加其中的一个细节，你会首先想到写一个接口来描述你扩展的契约，但是抽象的 Query 类并没有实现接口，你必须经常的变化自

己的查询对象到 Query 中并在本地 Lucene 中调用。成堆的例子如(HitCollecor,…)这对使用 AOP 和自动代理来说也是一个问题.

- 别扭的迭代实现.没有 hasNext()方法,next()方法返回布尔类型并刷新对象内容.这对你想要保持对迭代的元素跟踪来说非常的痛苦.我假定这是故意用来节省内存但是它又一次导致了算法上的杂乱和复杂.

4.积分不能被插件化

Lucene 有自己对积分算法的实现,当条件增加时使用 Similarity 类。但很快它显示出局限性当你想要表示复杂的积分,例如基于实际匹配和元数据的查询。如果你这样做，你不得不继承 Lucene 的查询类。因为 Lucene 使用类似 tf/idf 的积分算法，然而在我们遇到的场合，在语意上的积分上 Lucene 的积分机制并不合适。我们被迫重写每一个 Lucene 的查询类使得它支持我们自定义的积分。这是一个问题。

5.跨度查询太慢

这对 Lingway 公司来说可能是个特殊的问题。我们对跨度查询有很强要求,Lucene 检索结构已经开始添加这一细节，但它们当初可没这么想。最基础的实现导致了复杂的算法并且运行缓慢，尤其是当某些短语在一份文档中重复了许多次出现。这是为什么我倾向说 Lucene 是一个高性能的划词检索引擎当你仅仅使用基本的布尔查询时。

6. 没有对集群的内置支持。

如果你创建集群，你可以写出自己对 Directory 的实现，或是使用 Solr 或者使用 Nutch+Hadoop。Solr 和 Nutch 都支持 Lucene，但不是直接的替代。Lucene 是可嵌入的，而你必须支持 Solr 和 Nutch..我认为 Hadoop 从 Lucene 团队中产生并不惊讶:Lucene 并不是通用的。它的内在性决定了对大多数场合来说它是非常快速的，但是对大型文档集合时,你不得不排除 Lucene。因为它在内核级别上并没有实现集群，你必须把 Lucene 转换到别的搜索引擎，这样做并不直接。转换到 Solr 或者 Nutch 上的问题会让你遇到许多不必要的麻烦:Nutch 中的集成 crawling 和 Solr 中的检索服务。

3、使用步骤

3.1 前期准备

导包

Lucene 需要的基本包有 5 个

lucene-core，lucene-analyzers-common，lucene-highlighter，lucene-memory，lucene-queryparse

这几个包即可满足我们在创建索引或者搜索时的基本需求。

注意：不同版本的 lucene 包在操作时调用的方法也可能会不同，我们下面的例子中使用的版本为 6.0.0，其运行环境需要 1.8 以上的 JDK。

3.2 创建索引

lucene 的“索引库”中是存放 Document 的。Document 中存放的是该 Document 的 field ，即 List<Field>,而我们要用 Java 的面向对象编程，不能直接把 java 的对象直接存放在 Lucene 的索引库中，所以需要转换下。

创建索引的步骤：

1、创建一个 IndexWriter 对象

2、将实体对象转换为 Document 对象

3、将 Document 对象存入索引库中

4、关闭资源

下面是一个创建索引的例子：

```
package com.lucene.test;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
```

```
import org.apache.lucene.document.FieldType;
import org.apache.lucene.document.TextField;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.index.IndexWriterConfig;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.baidu.entity.Item;
import com.baidu.service.ItemService;

public class CreateIndex {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("spring.xml");
        ItemService service = context.getBean(ItemService.class);
        //前面这部分是查找所有要添加到索引的数据
        List<Item> list = service.findAllItems();

        try {
            //索引库位置（这个位置可以自定义的）
            Directory directory = FSDirectory.open(new File("E:/dicindex").toPath());
            //分词器
            Analyzer analyzer = new StandardAnalyzer();
            IndexWriterConfig config = new IndexWriterConfig(analyzer);
            //1、创建一个 IndexWriter 对象
            IndexWriter writer = new IndexWriter(directory,config);

            //创建一个集合存储 Document 对象
            List<Document> dlist = new ArrayList<Document>();

            //遍历集合 list 将 item 挨个存入索引
            for (Item item : list) {
                    /*
                     * TYPE_STORED        索引，分词，存储
                     * TYPE_NOT_STORED    索引，分词，不存储
                     */
                    Document document = new Document();
                    FieldType type = new FieldType(TextField.TYPE_STORED);
                    /*type.setStored(false);//是否存储
                    type.setTokenized(false);//是否分类*/
                    //2、将 Article 对象转成 document 对象
                    //第一个参数是搜索时的关键字，可以随意取名
```

```
                    Field idfield = new Field("id",item.getId().toString(),type);
                    Field titlefield = new Field("title",item.getTitle(),type);
                    Field contentfield = new
Field("content",item.getSellPoint(),type);

                    document.add(idfield);
                    document.add(titlefield);
                    document.add(contentfield);
                    dlist.add(document);

            }
            //3、把 document 对象放入索引库中
            writer.addDocuments(dlist);
            //4、关闭资源
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("索引添加完成");
    }

}
```

3.3 查询索引

查询索引的步骤：

1、创建一个 IndexSearch 对象

2、创建一个 Query

3、将检索出的数据由 Document 转换为实体对象存入集合中

4、关闭资源

下面是一个从索引库中检索的例子：

```
package com.lucene.test;

import java.io.File;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.List;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.DirectoryReader;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.queryparser.classic.MultiFieldQueryParser;
```

```
import org.apache.lucene.queryparser.classic.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.ScoreDoc;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.Directory;
import org.apache.lucene.store.FSDirectory;
import com.baidu.entity.Item;

public class FindIndex {

    public static void main(String[] args) {
        try {
            Directory directory = FSDirectory.open(new
File("E:/dicindex").toPath());
            IndexReader reader = DirectoryReader.open(directory);
            IndexSearcher search = new IndexSearcher(reader);
            //分词器
            Analyzer analyzer = new StandardAnalyzer();
            //从单个字段中检索
//          QueryParser parser = new QueryParser("content", analyzer);
            //从多个字段中检索
            QueryParser parser = new MultiFieldQueryParser(new
String[]{"title","content"}, analyzer);

            Query query = parser.parse("手机");
            //第二个参数是指检索的条数，我们做分页的时候可以把这个数据定义大一点，因
        为 lucene 分页时也需要将全部数据都检索出来再从中选出需要的数据
            TopDocs topDocs = search.search(query, 20);
            int total = topDocs.totalHits;//检索出来的记录数
            System.out.println("符合查询关键字(手机)的结果有："+total);
            //检索出来的记录
            ScoreDoc[] scoreDocs = topDocs.scoreDocs;
            List<Item> list = new ArrayList<Item>();
            for (ScoreDoc scoreDoc : scoreDocs) {
                //关键词的索引
                int index = scoreDoc.doc;
                //根据关键词的索引查到对应的 document 对象
                Document doc = search.doc(index);
                //将 Document 对象转为实体对象
                Item a = new Item();
                a.setId(new BigInteger(doc.get("id")));
                a.setTitle(doc.get("title"));
                a.setSellPoint(doc.get("content"));
```

```
            list.add(a);
        }
        //输出查询记录
        for (Item article : list) {
            System.out.println(article.getTitle());
        }
        //关闭资源
```