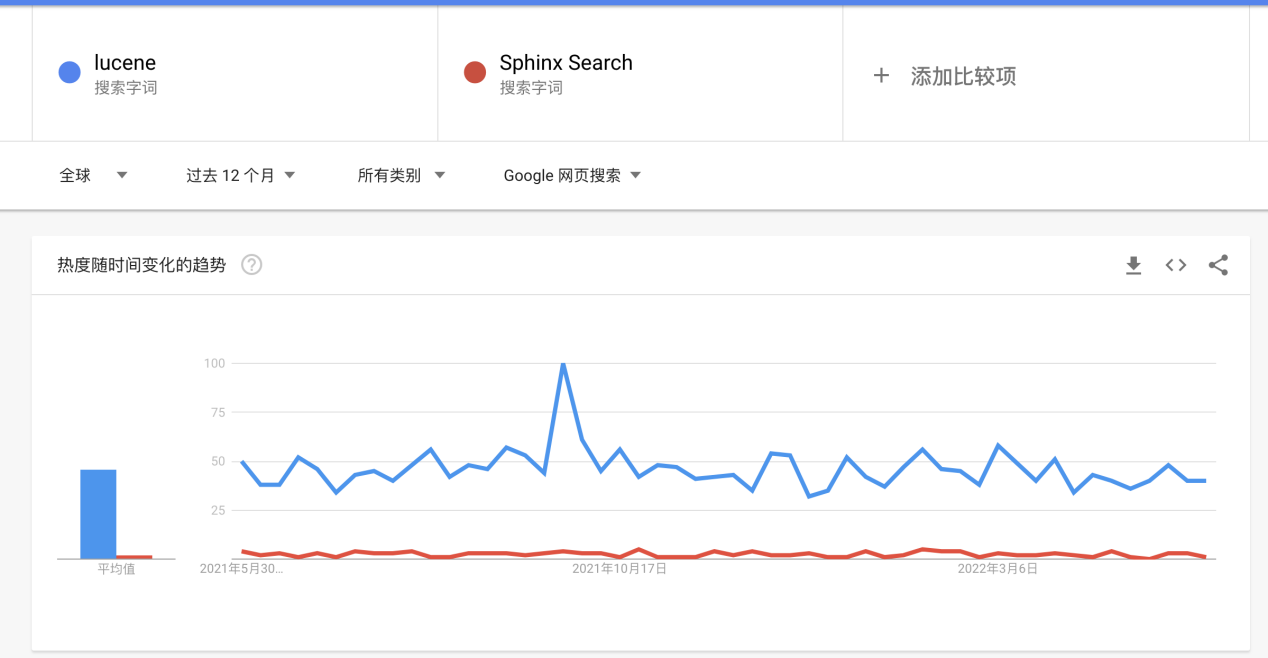
BTC查询数据缓存方案研究

1. 备选优化方案简介
   1. 方案一：Lucene
      1. 官网：https://lucene.apache.org/
      2. 开发语言：JAVA
      3. 开源协议：Apache License, Version 2.0
      4. 使用算法：倒排索引，跳跃表，FST（Finite StateTransducers）
   2. 方案二：sphinx
      1. 官网：http://sphinxsearch.com/
      2. 开发语言：C++
      3. 开源协议：GNU General Public License (GPL), version 2.
      4. 使用算法：权重，反向索引
2. 方案对比，基于JAVA语言

|  |  |  |
| --- | --- | --- |
|  | **Lucene** | **Sphinx** |
| **使用** | Java lib包 | 系统应用需要安装 |
| **建立索引** | 官方方法，任意来源 | 配置数据库，自动生成索引文件  支持mysql, pgsql, mssql, xmlpipe, xmlpipe2, odbc  其他方式需自行实现 |
| **增加索引** | 官方方法 | 需要自行实现（类似mysql方式） |
| **修改索引** | 无法直接修改 | 官方方法 |
| **删除索引** | 官方方法 | 需要自行实现（类似mysql方式） |
| **查询索引** | 官方方法 | 官方方法 |
| **分布式** | Solr或ElasticSearch基于Lucene | 自身支持 |
| **项目活跃度** | 9.2.0 - 2022.05.23 | 3.4.1 - 2021.07 |
| **项目热度（Google）** | ≈ 50 | ≈ 4 |

项目热度对比



1. 方案性能对比，基于33,000,000条数据量

|  |  |  |
| --- | --- | --- |
|  | **Lucene** | **Sphinx** |
| 启动内存 | 82 ~ 286MB | 3.03GB |
| 磁盘使用 | 4,853,371,259 字节（磁盘上的4.89 GB），共57个项目 | 7,269,660,301 字节（磁盘上的7.3 GB），共576个项目 |
|  |  |  |
| 总数统计 | 0.017ms | 336ms |
| 最大记录数 | 0.002ms | 813ms |
| 普通字段搜索 | 28.86 ~ 48ms | 616ms |
| 关键字段搜索 | 1.2 ~ 6.6ms | 3.5ms |
|  |  |  |
| 非文本字段搜索 | 422 ~ 578ms | 7.6 ~ 15.4ms |
| 引擎id搜索 | 引擎无自带ID | 10 ~ 0.4ms |
| 索引字段搜索 | 1.2 ~ 6.6ms  （无特别索引，参考关键字段） | 0.6 ~ 2.3ms |
|  |  |  |
| 100w数据写入 | 8.1s ~ 10.5s | 240 ~ 270s |
| 100w数据写入内存使用 | 2.7 ~ 3.5GB | 3.03GB |
|  |  |  |
| 100w数据读取 | 9 ~ 13s | 23 ~ 61s |
| 100w数据读取内存使用 | 1.12 ~ 1.38GB | 3.03GB |

1. 示例代码
   1. Lucene
      1. 使用java工具包

<dependency>

<groupId>org.apache.lucene</groupId>

<artifactId>lucene-core</artifactId>

<version>9.1.0</version>

</dependency>

* + 1. 基础设置，Lucene同一索引可用同一配置，更新索引后需要重新建立reader

public class LuceneManager {

private static volatile Directory gDirectory;

public static synchronized Directory directory() throws Exception {

if (gDirectory == null) {

gDirectory = FSDirectory.open(Paths.get("/Volumes/Base/lcache"));

// gDirectory = FSDirectory.open(Paths.get("/Users/chriswang/tmp/lcache"));

}

return gDirectory;

}

private static volatile IndexWriter gIndexWriter;

public static synchronized IndexWriter indexWriter() throws Exception {

if (gIndexWriter == null) {

Analyzer analyzer = new StandardAnalyzer();

IndexWriterConfig config = new IndexWriterConfig(analyzer);

config.setIndexCreatedVersionMajor(Version.LUCENE\_9\_1\_0.major);

config.setOpenMode(IndexWriterConfig.OpenMode.CREATE\_OR\_APPEND);

gIndexWriter = new IndexWriter(LuceneManager.directory(), config);

}

return gIndexWriter;

}

private static volatile IndexReader gIndexReader;

public static synchronized IndexReader indexReader() throws Exception {

if (gIndexReader == null) {

gIndexReader = DirectoryReader.open(directory());

}

return gIndexReader;

}

private static volatile IndexSearcher gIndexSearcher;

public static synchronized IndexSearcher indexSearcher() throws Exception {

if (gIndexSearcher == null) {

gIndexReader = indexReader();

gIndexSearcher = new IndexSearcher(gIndexReader);

}

return gIndexSearcher;

}

}

* + 1. 创建索引
       1. 模型转换，非储存字段与索引字段要分别设置，转为Lucene专用对象

Document lcDoc = new Document();

lcDoc.add(new TextField("\_id", doc.\_id, Store.YES)); // 不进行分析

lcDoc.add(new StringField("address", doc.address, Store.YES));

lcDoc.add(new StringField("txid", doc.txid, Store.YES));

final long id = currentIndex.addAndGet(1l);

// field

lcDoc.add(new StoredField("id", id));

lcDoc.add(new StoredField("count", doc.\_count));

lcDoc.add(new StoredField("txtime", doc.txtime));

lcDoc.add(new StoredField("block", doc.block));

lcDoc.add(new StoredField("side", doc.side != null && doc.side ? 1 : 0));

// search

// lcDoc.add(new DoubleDocValuesField("count", doc.\_count));

lcDoc.add(new SortedNumericDocValuesField("id", id)); // id for search

lcDoc.add(new NumericDocValuesField("txtime", doc.txtime));

lcDoc.add(new NumericDocValuesField("block", doc.block));

// lcDoc.add(new NumericDocValuesField("side", doc.side != null && doc.side ? 1 : 0));

* + - 1. 保存

LuceneManager.indexWriter().addDocuments(lcDocs);

* + 1. 搜索索引
       1. 搜索

// 文本

query = new TermQuery(new Term("address", "1NkBNT6pJ3Xfq5v39FJ9m7UM3rS8gV67C5"));

topDocs = LuceneManager.indexSearcher().search(query, 10);

// 数字，SortedNumericDocValuesField或者NumericDocValuesField，看储存类型

query = SortedNumericDocValuesField.newSlowRangeQuery("id", 101l, 1000100l);

topDocs = LuceneManager.indexSearcher().search(query, 1000000);

* + - 1. 模型转换

for (ScoreDoc sdoc : docRes)

Document doc = indexSearcher.doc(sdoc.doc);

Map<String ,Object> res = new HashMap<>();

res.put("id", ObjectUtils.getLong(doc.get("id")));

res.put("\_id", doc.get("\_id"));

res.put("address", doc.get("address"));

res.put("count", ObjectUtils.getDoubleValue(doc.get("count")));

res.put("txtime", ObjectUtils.getLongValue(doc.get("txtime")));

res.put("block", ObjectUtils.getLongValue(doc.get("block")));

res.put("txid", ObjectUtils.getLongValue(doc.get("txid")));

res.put("side", ObjectUtils.getIntValue(doc.get("side")));

* 1. Sphinx
     1. 使用java工具包

自实现类似JDBC操作

* + 1. 基础设置

id需要自己维护

* + 1. 创建索引
       1. 模型转换，转为SphinxBaseEntity的子类对象即可

Transactions lcDoc = new Transactions();

lcDoc.\_id = ObjectUtils.getString(doc.get("\_id"));

lcDoc.address = ObjectUtils.getString(doc.get("address"));

lcDoc.\_count = ObjectUtils.getDoubleValue(doc.get("count"));

lcDoc.txtime = ObjectUtils.getLongValue(doc.get("txtime"));

lcDoc.block = ObjectUtils.getLongValue(doc.get("block"));

lcDoc.txid = ObjectUtils.getString(doc.get("txid"));

String side = ObjectUtils.getString(doc.get("side"));

lcDoc.side = side != null && side.equals("out") ? true : false;

* + - 1. 保存

类似数据库insert操作

* + 1. 搜索索引
       1. 搜索

类似数据库search操作

* + - 1. 模型转换

gson由map转为entity