Поисковая библиотека Lucene

Курс «Базы данных»

Андреев Герман Андреевич

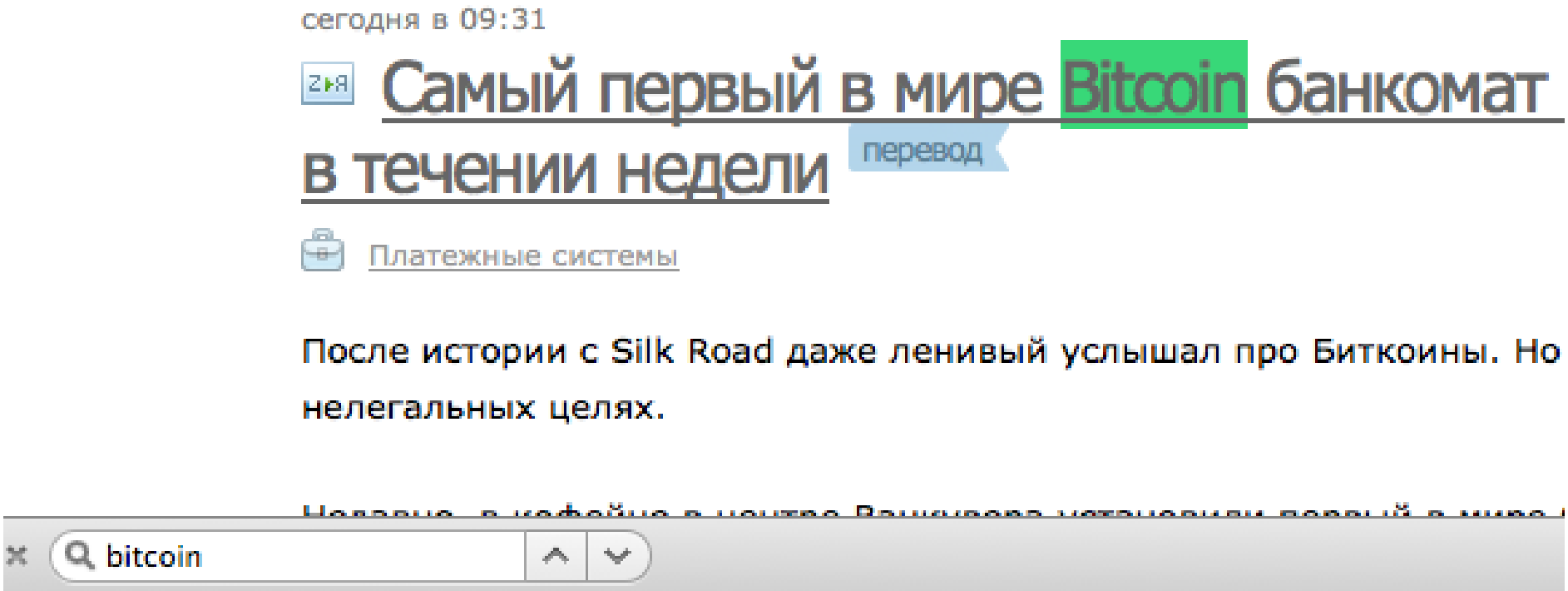
Computer Science Center

18 ноября 2013 г.

Содержание

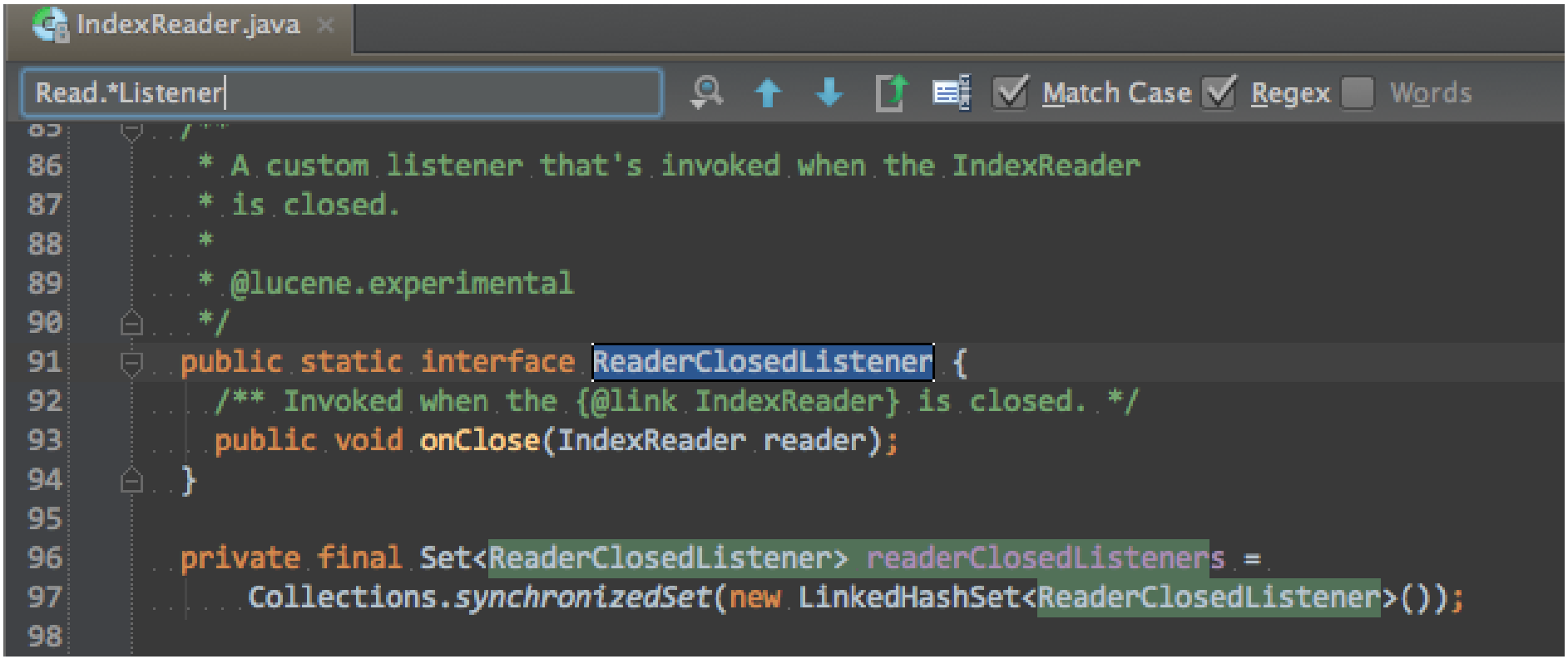
Примеры

Пример: поиск по документу



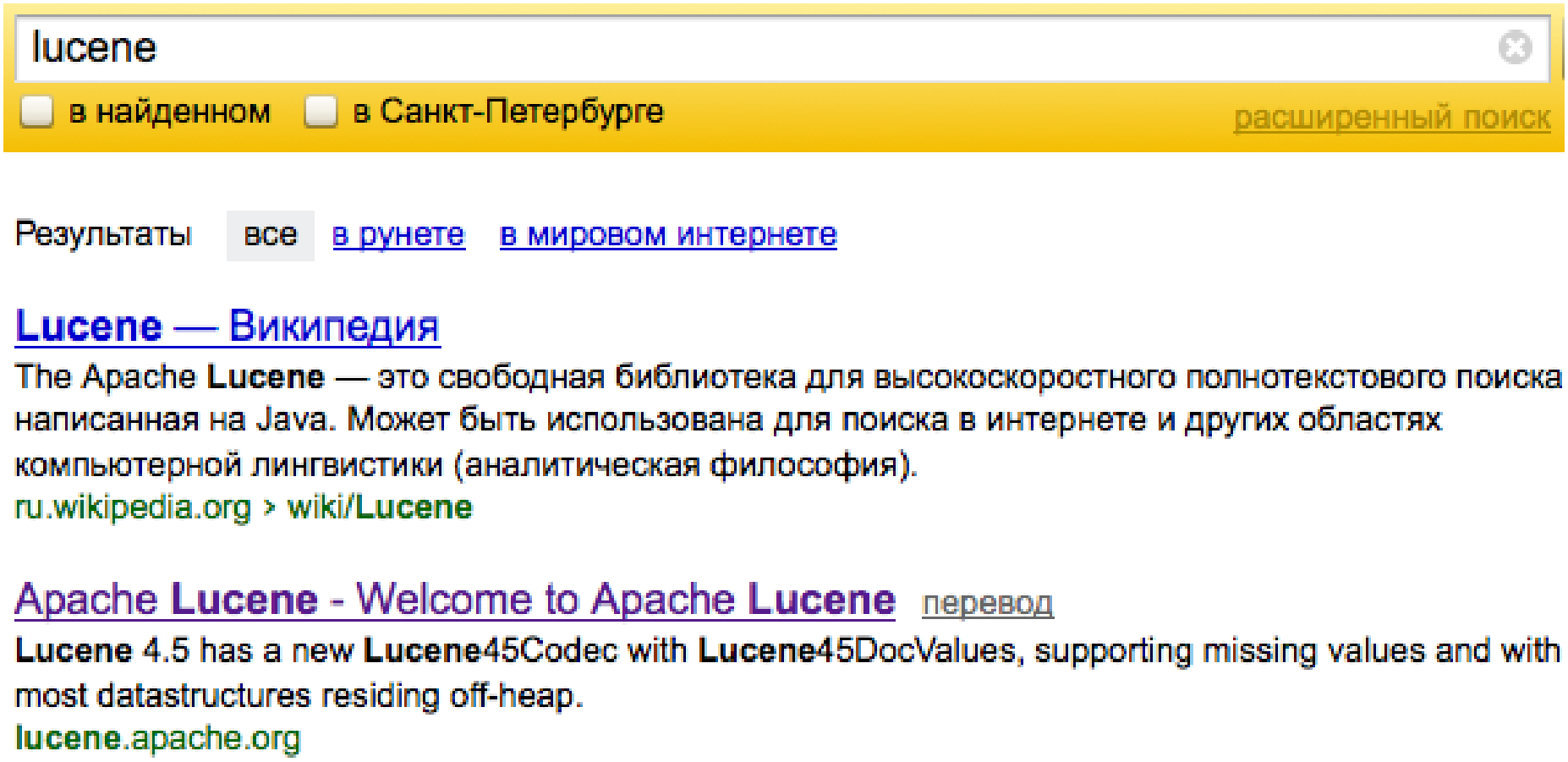
Примеры

Пример: поиск по документу



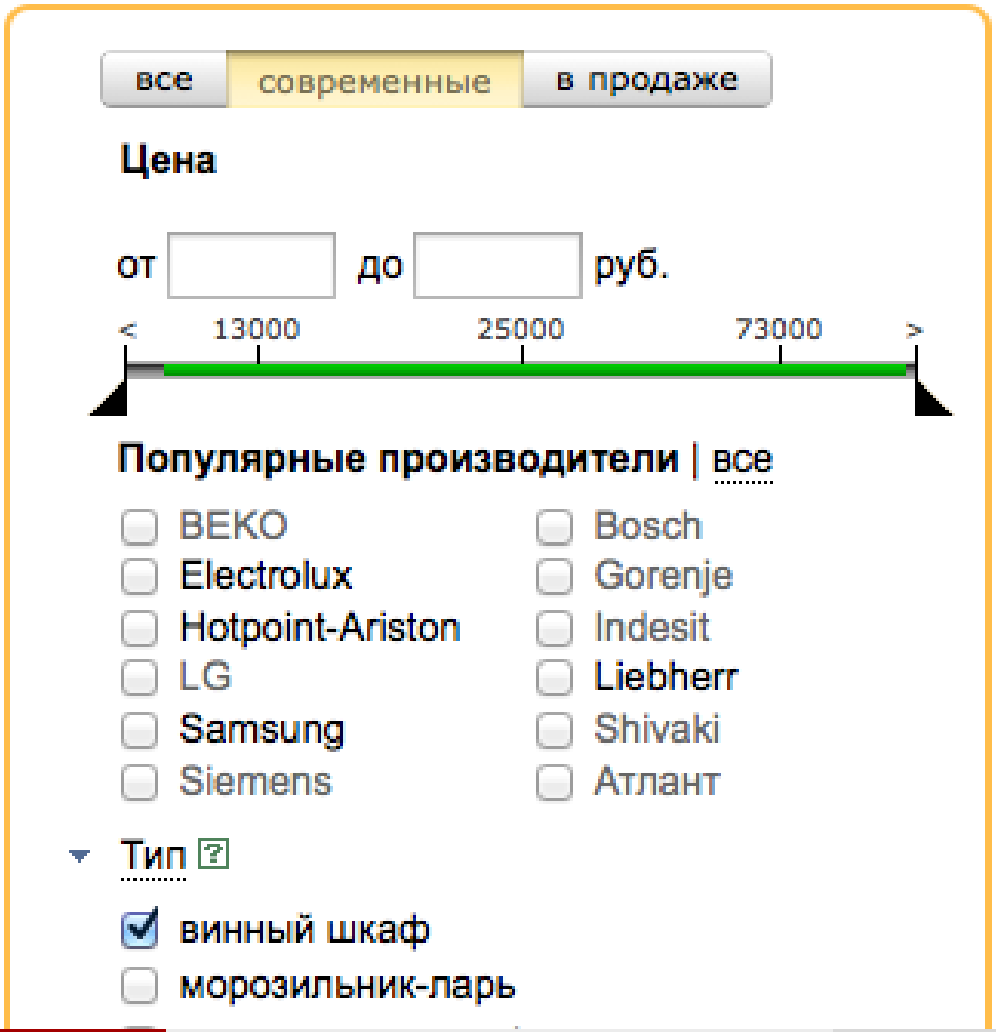
Примеры

Пример: полнотекстовый поиск



Примеры

Пример: параметрический поиск



Примеры

Поиск по документу

- Хорошо изучен
- Множество алгоритмов
- Готовые библиотеки
- Можно реализовать самому

Примеры

Поиск регулярных выражений:

- Хорошо изучен
- Множество алгоритмов
- Готовые библиотеки
- Сложно реализовать самому

Примеры

Полнотекстовый поиск:

- Очень много теории
- Огромное количество алгоритмов
- Некоторые доступны в виде библиотек

Примеры

Различия между примерами:

- Количество документов
- Функция поиска
- Ранжирование

Библиотека Lucene

Особенности:

- Реализована на Java
- Достаточно компактна: lucene-core - 2.1 Mb
- Богатая функциональность
- Высокая скорость работы
- Высокая скорость индексации

Библиотека Lucene

Преимущества:

- Открытый исходный код
- Удобная лицензия: Apache License
- Много коммиттеров
- Широко известна
- Есть порт под .NET

Библиотека Lucene

История развития:

- Появилась в 1999: жила на SourceForge
- В 2001 переместилась в Apache
- В 2005 стала top-level проектом в рамках Apache
- Претерпела 4 мажорных релиза
- Активно развивается: последняя версия 4.5.1 выпущена 24 октября
- 8 релизов только за этот год

Модель данных

Модель данных:

- Основные объекты: `Document`, `Field`

Модель данных

`Field`:

- Пара ключ-значение
- Ключ: произвольная ненулевая строка
- Значение: произвольная ненулевая строка
- В 4.0.0 версии появилась возможность хранить типизированные значения

Модель данных

`Field`:

- У каждого поля есть специальные атрибуты
- `Indexed`
- `Stored`
- `Tockenized`

Модель данных

`Indexed Field`:

- Будет ли построен индекс по этому полю
- По этим полям осуществляется поиск
- По этим полям можно сортировать

Модель данных

`Stored Field`:

- Попадет ли поле в хранильще документов
- Нужно ли это значение при поиске

Модель данных

`Tockenized Field`:

- Требуется ли дополнительная обработка значения

Модель данных

`Document`:

- Содержит список `Field`
- Нет ограничения на повторяемость имени
- При построении индекса каждому документу назначается docId

Построение индекса

`Directory`:

- Абстракция над файловой системой
- Есть несколько реализаций
- `SimpleFSDirectory`: простая реализация
- `NIOFSDirectory`: реализация с использованием NIO
- `MMapDirectory`: реализация на основе memory mapped файлов
- `RAMDirectory`: все содержимое хранится в памяти
- Важно! Не используйте `RAMDirectory` для крупных индексов

Построение индекса

Индекс состоит из:

- Инвертированного индекса
- Хранилища документов

Построение индекса

Построение индекса:

- `IndexWriterConfig`: настройки индексации
- `IndexWriter`: управляет созданием индекса
- `indexWriter.addDocument(...)`: добавление документа в индекс
- Одновременно записывать может только один `IndexWriter`

Построение индекса

Хранилище документов:

- Попадают все `Stored` поля
- Часть исходного документа в результатах поиска

Построение индекса

Токенизация:

- Каждый индексируемое поле превращается в последовательность токенов

Построение индекса

Инвертированный индекс:

- Попадают все `Indexed` поля
- На нем основан механизм поиска

Построение индекса

Пример

Построение документа

```
1    Document doc = new Document();
2    Field name = new StringField("name", country.getName(),
         Field.Store.YES);
3    doc.add(name);
4    Field population = new LongField("population",
         country.getPopulation(), Field.Store.YES);
5    doc.add(population);
```

Построение индекса

Пример

Неиндексируемое поле

```
1    FieldType TYPE_NOT_INDEXED = new FieldType();
2    TYPE_NOT_INDEXED.setIndexed(false);
3    TYPE_NOT_INDEXED.setOmitNorms(true);
4    TYPE_NOT_INDEXED.setIndexOptions(IndexOptions.DOCS_ONLY);
5    TYPE_NOT_INDEXED.setTokenized(false);
6    TYPE_NOT_INDEXED.freeze();
7    ...
8    Field currency = new Field("currency",
         country.getCurrency(), TYPE_NOT_INDEXED);
9    doc.add(currency);
```

Построение индекса

Пример

Создание индекса

```
1    Directory dir = FSDirectory.open(new File(indexPath));
2    Analyzer analyzer = new
         StandardAnalyzer(Version.LUCENE_40);
3    IndexWriterConfig iwc = new
         IndexWriterConfig(Version.LUCENE_40, analyzer);
4    iwc.setOpenMode(OpenMode.CREATE);
5    IndexWriter writer = new IndexWriter(dir, iwc)
```

Построение индекса

Пример

Добавление документа

```
1     ...
2     IndexWriter writer = new IndexWriter(dir, iwc);
3     for (Country country: countries) {
4        Document doc = new Document();
5        ...
6        writer.addDocument(doc);
7     }
8     writer.close();
```

Поиск по индексу

Поиск по индексу:

- За чтение отвечает IndexReader
- За поиск - IndexSearcher
- Lucene позволяет читать из индекса одновременно с записью

Поиск по индексу

Поиск по индексу:

- Для поиска `indexSearcher.search(query, ...)`
- Все запросы - наследники `Query`
- В простейшем случае `indexSearcher.search(query, n)`

Поиск по индексу

Поиск по индексу:

- `TopDocs`: результат поиска
- `totalHits`: всего найденных документов
- `scoreDocs`: список `docId` для найденных документов

Поиск по индексу

Запросы:

- Запросы бывают разных типов
- `TermQuery`: наличие токена в документе
- `BooleanQuery`: условие на основе других `Query`
- `TermRangeQuery`: диапазон текстовых токенов
- `NumericRangeQuery`: числовой диапозон
- Несколько других типов

Поиск по индексу

Результаты поиска:

- За результаты отвечает `Collector`
- Чаще всего используется `TopScoreDocCollector`
- Можно реализовать свой `Collector`

Поиск по индексу

Сортировка результатов:

- В `indexSearch.search` можно передать `Sort`
- Содержит список полей для сортировки

Поиск по индексу

Ранжирование:

- Альтернатива сортировке
- Задается через класс `Similarity`
- Должно совпадать при индексировании и поиске
- Нужно для полнотекстового поиска
- `Vector Space Model` - на основе `tf-idf`
- `BM25`

Поиск по индексу

Пример

Чтение индекса

```
1    Directory dir = FSDirectory.open(new File(indexPath));
2    Analyzer analyzer = new
         StandardAnalyzer(Version.LUCENE_40);
3    DirectoryReader reader = DirectoryReader.open(dir);
4    IndexSearcher searcher = new IndexSearcher(reader);
```

Поиск по индексу

Пример

Построение запроса

```
1    BooleanQuery bq = new BooleanQuery();
2    TermQuery tq = new TermQuery(new Term("name", "Russia"));
3    NumericRangeQuery<Long> nrq =
         NumericRangeQuery.newLongRange(
4      "population", 140_000_000L, Long.MAX_VALUE, true, false);
5
6    bq.add(tq, BooleanClause.Occur.MUST_NOT);
7    bq.add(nrq, BooleanClause.Occur.MUST);
```

Поиск по индексу

Пример

Поиск документов

```
1    TopDocs topDocs = indexSearcher.search(bq, 10);
2    for (ScoreDoc scoreDoc: topDocs.scoreDocs) {
3        Document foundDoc = indexSearcher.doc(scoreDoc.doc);
4    }
```

Поиск по индексу

Поиск документов:

- `foundDoc` отличается от `doc`

Индекс на диске

- Индекс разбит на сегменты
- После создания сегмент не меняется
- Содержит диапазон документов
- Периодически выполняется merge

Индекс на диске

- `DocId` внутри сегмента уникальны
- При удалении могут появится пропуски
- Merge удаляет пропуски
- Merge может изменить `docId`

Индекс на диске

Файлы индекса:

- `write.lock`: лок на запись
- `segments.gen`, `segments_N`: commit point
- Файлы сегмента начинаются на `_N`
- `.fnm`: методанные полей
- `.fdx`, `.fdt`: Stored поля документа

Индекс на диске

Файлы индекса:

- `.tim`, `.tip`: словарь `Term`
- `.doc`: список документов, содержащих `Term`
- `.tvx`, `.tvd`, `.tvf`: `Term Vector`
- `.nvd`, `.nvm`, `.dvd`, `.dvm`: нормы и данные для ранжирования
- `.del`: удаленные документы

Индекс на диске

Файлы индекса:

- Возможно включить "склеивание"файлоф
- `.cfs`, .cfe: составные файлы индекса

Индекс на диске

Новый сегмент создается в момент операции `flush`
Можно вызвать программно, обычно вызывается автоматически
Настройки:

- `IndexWriterConfig.setMaxBufferedDocs`:
- `IndexWriterConfig.setRAMBufferSizeMB`

Устройство поиска

- `IndexReader`: содержит списко сегментов
- Поиск по каждому сегменту независим
- Выполняется последовательно по списку сегментов

Устройство поиска

Инвертированный индекс состоит из:

- Списка `Term` для каждого поля
- Упорядоченного списка `docId` для каждого `Term`

Устройство поиска

`TermQuery` порождают объект, умеющий:

- Проверяет есть ли нужный `Term` для поля
- Находит соответствующему списку `docId`

Устройство поиска

`Query` порождают объект, отвечающий за:

- Хранение текущего найденного `docId`
- Поиск следующего подходящего `docId`

Устройство поиска

`BooleanQuery`

- Композиция из `Query`
- Опрашивает дочернии `Query` по очереди
- Документ найден, если все дочерние `Query` передвинулись на один и тотже `docId`

Устройство поиска

Большинство объектов - потоко-безопасно:

- `IndexReader`
- `IndexSearcher`
- `Query`: после создания можно искать из нескольких потоков

Пример использования

- Размер документа: 14 полей
- Одно из полей мультизначное: до 10 значений
- Все поля `Indexed`, некоторые `Stored`
- Порядка 100000 объектов

Пример использования

- Размер индекса: несколько десятков мегабайт
- Время построения: быстро

Пример использования

- Жесткие требования по скорость работы
- Нагрузка порядка 25 RPS
- Несколько тысяч найденных объектов

Пример использования

Первая попытка

- Оказалась неудовлетворительной
- Получение документа - дорогая операция

Пример использования

Вторая попытка

- Избавится от `Stored` полей
- Все документы хранить в памяти
- `Lucene` - только для поиска

Пример использования

Вторая попытка

- Успех!
- Максимальная нагрузка - сотни RPS
- 98% запросов - 5 ms
- 99.5% запросов - 25 ms