УДК 004

Работа с репозиториями Spring Data Solr

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье будут рассмотрены возможности создания репозитория с использованием Spring Data Solr. Работа будет происходить в среде разработки IntelijIdea.
Ключевые слова: Spring Data Solr, Spring, Spring boot

Working with Spring Data Solr repositories

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article explores the possibilities of creating a repository using Spring Data solr. The work will take place in the IntelijIdea development environment.
Keywords: Spring Data Solr, Spring, Spring boot

Spring Data — это фреймворк, к которому можно обратиться при попытке получить доступ к базе данных в приложении Spring. Наряду с реляционными базами данных он также обеспечивает поддержку широкого спектра баз данных noSQL, включая базы данных на основе документов, такие как Apache Solr.

Цель работы – создать репозитории с использованием Spring Data Solr и проверить их работоспособность.

Исследованиями в данной теме занимались следующие авторы. А.К.Борисов разработал web-приложение, позволяющее командированному сотруднику получить унифицированный доступ к приложениям IT-инфраструктурам [1]. Д.А.Викулина, С.Н.Макаров рассмотрели в своей

работе передовые средства и технологии для разработки настольных приложений, провели их анализ и сравнительную характеристику [2]. Т.И.Тимофеев, В.В.Козлов произвели сравнительный обзор фреймворков для настольных приложений по ряду критериев [3]. Е.Д.Зайцев, Д.М.Зайцев раскрыли вопросы эффективности автоматизации тестирования мобильных приложений и web. [4].

Чтобы создать новый проект загрузки Spring с помощью Spring Data Solr, необходимо добавить зависимость spring-boot-starter-data-solr (рис.1).

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-solr</artifactId>
</dependency>
```

Рисунок 1 – Добавление зависимости

Для того чтобы написать собственные запросы, сначала необходимо убедиться, что есть настройка модели (рис.2).

```
@Data
@SolrDocument(solrCoreName = MarkdownDocument.MARKDOWN_CORE)
public class MarkdownDocument {
    public static final String MARKDOWN_CORE = "markdown";
    public static final String FILE_ID_FIELD = "file.id";
    public static final String CONTENT_FIELD = "content";
    public static final String LAST_MODIFIED_FIELD = "last_modified";
    @Id
    @Indexed(FILE_ID_FIELD)
    private String id;
    @Indexed(CONTENT_FIELD)
    private String content;
    @Indexed(LAST_MODIFIED_FIELD)
    private LocalDateTime lastModified;
}
```

Рисунок 2 – Создание модели

Так же можно использовать многоядерную настройку, указав solrCoreName при добавлении @SolrDocument аннотации.

Теперь, когда есть модель, можно начать писать репозиторий (рис.3).

```
public interface MarkdownDocumentRepository extends SolrCrudRepository<MarkdownDocument, String> {

}
```

Рисунок 3 – Создание репозитория

Далее добавляем методы, используя аннотацию @Query, для предоставления запроса в Solr (рис.4).

```
public interface MarkdownDocumentRepository extends SolrCrudRepository<MarkdownDocument, String> {
    @Query("file.id:?0 OR content:?0")
    List<MarkdownDocument> findAll(String searchTerm);
}
```

Рисунок 4 – Добавление метода

Данный пример будет искать все документы, соответствующие заданному термину, который может находиться либо в file.id поле, либо в content поле.

По умолчанию Solr может усилить поиск документов (рис.5).

```
List<MarkdownDocument> findByIdOrContent(String id, String content);
```

Рисунок 5 – Запрос

В этом случае документы, соответствующие как заданному, так id и content полю, получат более высокий балл, чем документы, соответствующие любому полю. Это можно легко увидеть, если добавить поле оценки в модель и аннотируя его @Score (рис.6).

```
@Score
private float score;
```

Рисунок 6 – Добавление счетчика

Если запустить приложение сейчас, то увидим, что документы соответствуют обоим, id и их количество content будет вдвое больше, чем у других документов. Количество повторов также меняет счет.

Работа с разбивкой на страницы работает одинаково во всех реализациях Spring Data. Просто добавляем Pageable в метод репозитория и возвращаем, «Page<MarkdownDocument>», а не «List<MarkdownDocument>» (рис.7).

```
Page<MarkdownDocument> findByIdOrContent(@Boost(2) String id, String content, Pageable pageable);
```

Рисунок 7 – Разбивка страниц

Теперь можно начать разбиение на страницы, вызвав этот метод с помощью PageRequest класса или реализовав собственный Pageable класс (рис.8).

```
public class OffsetPageRequest implements Pageable {
    private long offset;
    private int limit;
    private Sort sort;

    public OffsetPageRequest(long offset, int limit) {
        this(offset, limit, null);
    }

    public OffsetPageRequest(long offset, int limit, Sort sort) {
        this.offset = offset;
        this.limit = limit;
        this.sort = sort;
    }

    @Override
    public int getPageNumber() {
        return 0;
    }

    @Override
    public int getPageSize() {
        return limit;
    }

    @Override
    public long getOffset() {
        return offset;
    }

    @Override
    public Sort getSort() {
        return sort;
    }

    @Override
    public Pageable next() {
        return new OffsetPageRequest(getOffset() + getPageSize(), getPageSize(), getSort());
    }

    @Override
    public Pageable previousOrFirst() {
        return new OffsetPageRequest(Math.max(0, getOffset() - getPageSize()), getPageSize(), getSort(
    }

    @Override
    public Pageable first() {
        return new OffsetPageRequest(0, getPageSize(), getSort());
    }

    @Override
    public boolean hasPrevious() {
        return getOffset() > 0;
    }
}
```

Рисунок 8 – Добавление страниц

Эта реализация позволяет работать со смещениями и ограничениями, а не со страницами и размерами.

При работе с репозиториями иногда требуется больше контроля над запросами, которые собираемся выполнить, путем их программного определения. Для этого можно использовать API критериев. Прежде чем начать, нужно определить SolrTemplatebean-компонент (рис.9).

```
@Bean
public SolrTemplate solrTemplate(SolrClient solrClient) {
    return new SolrTemplate(solrClient);
}
```

Рисунок 9 – Добавление компонента

После определения этого bean-компонента можно расширить существующий репозиторий, создав новый интерфейс. Далее добавим это расширение в репозиторий и после этого создаем свою реализацию (рис.10).

```
@AllArgsConstructor
public class MarkdownDocumentRepositoryImpl implements CustomMarkdownDocumentRepository {
    private SolrTemplate solrTemplate;

    @Override
    public HighlightPage findDocuments(String searchTerm, Pageable page) {
        Criteria fileIdCriteria = new Criteria(MarkdownDocument.FILE_ID_FIELD).boost(2).is(searchTerm)
        Criteria contentCriteria = new Criteria(MarkdownDocument.CONTENT_FIELD).fuzzy(searchTerm);
        SimpleHighlightQuery query = new SimpleHighlightQuery(fileIdCriteria.or(contentCriteria), page
        query.setHighlightOptions(new HighlightOptions()
            .setSimplePrefix("<strong>")
            .setSimplePostfix("</strong>")
            .addField(MarkdownDocument.FILE_ID_FIELD, MarkdownDocument.CONTENT_FIELD));
        return solrTemplate.queryForHighlightPage(MarkdownDocument.MARKDOWN_CORE, query, MarkdownDocum
    }
}
```

Рисунок 10 – Создание реализации

В этом примере строим два критерия:

- Критерий, который проверяет, соответствует ли file.id поле заданному поисковому запросу, и, если это так, повышает оценку в два раза.
- Другой критерий, который проверяет, соответствует ли нечеткое поле содержимого заданному поисковому запросу.

После создания этих критериев можно использовать метод Criteria.and() или, Criteria.or() чтобы присоединиться к ним и создать из них запрос. Существуют различные реализации запросов, такие как SimpleQuery, SimpleHighlightQuery. В зависимости от типа результата или страницы, которую необходимо получить.

В данном случае используем, SimpleHighlightQuery и предоставляем префикс и постфикс, как было показано ранее, передавая HighlightOptions

объект. Доступ к выделенным частям можно выполнить с помощью page.getHighlights(solrDocument) метода.

В данной статье были рассмотрены примеры создания репозиторием и их работы с помощью Spring Data Solr.

Библиографический список

1. Борисов А.К. Sun secure global desktop все ваши приложения в окне браузера // Системный администратор. 2009. №9(82). С. 48-52.
2. Викулина Д.А., Макаров С.Н. Современные технологии создания desktop-приложений // Наука и современность 2012. №18. С. 180-186.
3. Тимофеев Т.И., Козлов В.В. Обзор современных средств создания интерфейса пользователя для desktop приложений // Традиции и инновации в строительстве и архитектуре. строительные технологии. 2017. №1. С.585-588.
4. Зайцев Е.Д., Зайцев Д.М. К вопросу об эффективности автоматизации тестирования web-, desktop- и мобильных приложений // Проблемы инфокоммуникаций 2018. №2(8). С. 56-63.