УДК 004

Использование EhCache 3 с загрузкой Spring

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 вопросы, касающиеся кэширования с помощью EHCache и Spring. Так же рассмотрены примеры использования кэширования.
Ключевые слова: EHCache, Spring, Spring boot

Using EhCache 3 with Spring boot

Erovleva Regina Viktorovna
Sholom-Aleichem Priamursky State University
student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article will cover issues related to caching with EHCache and Spring. Examples of using caching are also considered.
Keywords: EHCache, Spring, Spring boot

Абстракция кеша применяет кеширование к методам Java, тем самым уменьшая количество выполнений на основе информации, доступной в кэше. То есть каждый раз, когда вызывается целевой метод, абстракция применяет поведение кэширования, которое проверяет, был ли метод уже вызван для данных аргументов. Если он был вызван, кешированный результат возвращается без необходимости вызывать фактический метод. Если метод не был вызван, он вызывается, а результат кэшируется и возвращается пользователю, чтобы при следующем вызове метода возвращался кешированный результат. Таким образом, дорогостоящие методы (связанные с процессором или вводом-выводом) могут быть вызваны только один раз для заданного набора параметров, а результат будет повторно использован без необходимости повторного вызова метода.

Прохоров П.В., Разговоров Н.В. рассмотрели в своей работе современные подходы и технологии в разработке серверных приложений на примере онлайн-магазина с использованием Spring [1]. В своей статье Егунова А.И., Аббакумов А.А., Воропаева М.А., Вечканова Ю.С. рассматривают проблемы повышения эффективности образовательной деятельности вуза и использование научной интеллектуальной собственности в преподавательской деятельности. Так же предложили реализацию информационного хранилища и поисковой системы в виде J2EE-приложения с использованием фреймворка Spring [2]. В своей статье Жданова В.С. описала подход в реализации серверной части клиент-серверного мобильного приложения для просмотра расписания [3]. Так же Нарижный А.Д., Губенко Н.Е. провели сравнительный анализ технологий, которые имеют схожую функциональность и которые предназначены для одних и тех же задач. Это технологии стеков Spring и JavaEE, которые предназначены для разработки Enterprise-приложений [4]. Волушкова В.Л. рассмотрела в своей работе технологии программирования на примере языка Java [5].

Кэширование - обычная операция при разработке приложений. Spring предоставляет абстракцию поверх всех различных библиотек кеширования, чтобы сделать это еще проще (рис.1).

Начнем с открытия Spring Initializr и добавим следующие зависимости:

1. Spring Web: эта зависимость позволяет создавать REST API.
2. Spring cache abstraction: эта библиотека содержит абстракцию кеша.
3. Lombok: Эта библиотека упростит написание классов, поскольку она будет генерировать геттеры, сеттеры, конструкторы и так далее.

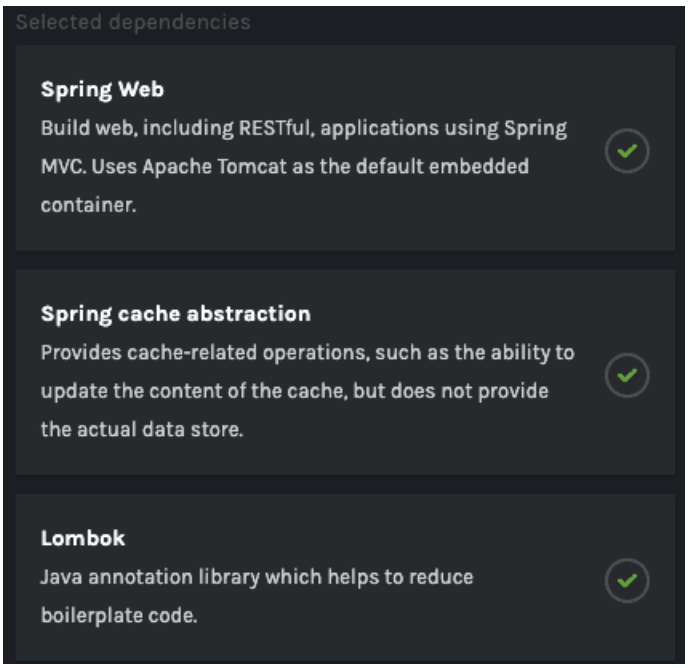

Рисунок 1 – добавление зависимостей

Чтобы иметь возможность настроить кеширование, сначала нужны данные для кеширования. В этом примере создадим API задачи, используя следующий DTO (рис.2)

```
@Getter
@RequiredArgsConstructor
public class TaskDTO {
    private final long id;
    private final String task;
    private final boolean completed;
}
```

Рисунок 2 – Создание класса DTO

Кроме того, создадим следующий класс (рис.3).

```
@Slf4j
@Service
public class TaskFacade {

    public List<TaskDTO> findAll() {
        log.info("Retrieving tasks");
        return List.of(
            new TaskDTO(1L, "My first task", true),
            new TaskDTO(2L, "My second task", false));
    }
}
```

Рисунок 3 – Зависимый класс

Так же создадим контроллер для получения задач (рис.4).

```
@RestController
@RequestMapping("/api/tasks")
public class TaskController {
    private final TaskFacade taskFacade;

    @GetMapping
    public List<TaskDTO> findAll() {
        return taskFacade.findAll(noCache);
    }
}
```

Рисунок 4 – Создания контроллера

Если перейдем по адресу «http://localhost:8080/api/tasks», то увидим, что исходит ответ JSON, содержащий задачи, которые были определены (рис.5).

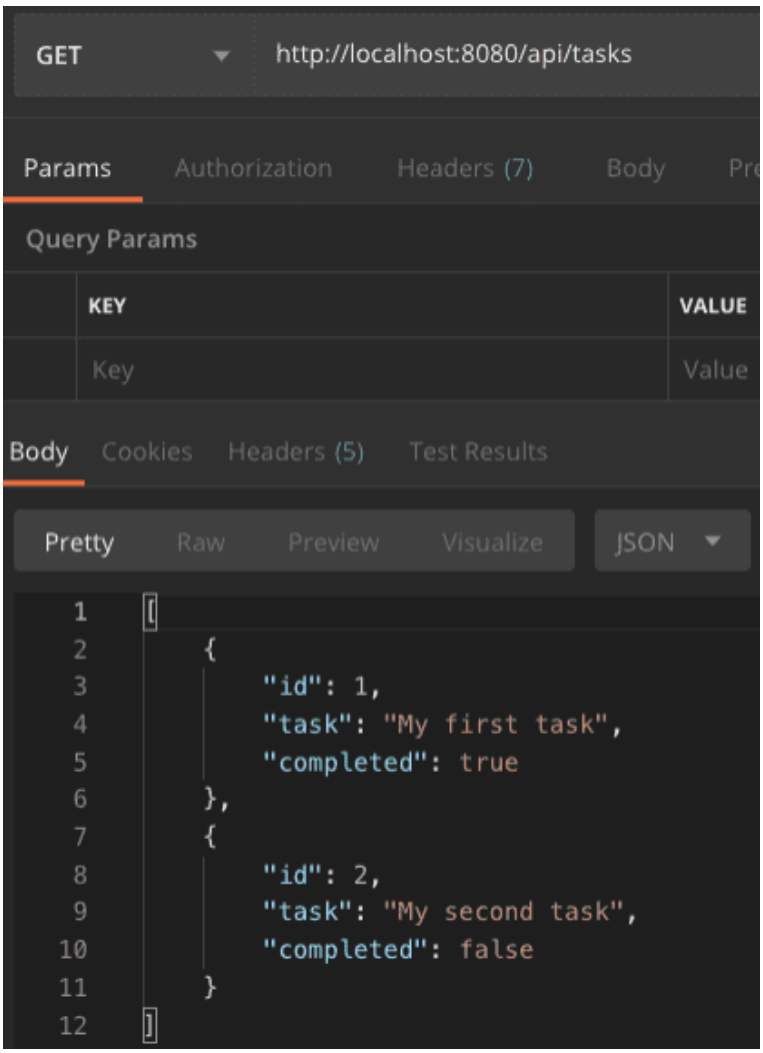

Рисунок 5 – Ответ JSON

Однако здесь еще нет кэшируемости. Если обновить страницу несколько раз, то увидите, что сообщение «Получение задач» продолжает появляться в журналах.

Чтобы включить кеш, сначала нужно добавить Ehcache3 в качестве зависимости (рис.6).

```
<dependency>
    <groupId>org.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
</dependency>
```

Рисунок 7 – Добавление зависимости

Чтобы сообщить Spring, что идет использование Ehcache3, нужно добавить «cache-api» библиотеку, которая позволяет использовать библиотеки кеширования, реализующие JSR-107, такие как «Ehcache».

Идентификатор группы должен быть «org.ehcache», убедимся, что используем ее, для «ehcache» библиотеки.

Следующим шагом будет создание файла конфигурации с именем «ehcache.xmlвнутри src/main/resources» (рис.8).

```
<config xmlns='http://www.ehcache.org/v3'>
    <cache alias="tasks">
        <key-type>org.springframework.cache.interceptor.SimpleKey</key-type>
        <value-type>java.util.List</value-type>
        <expiry>
            <ttl unit="minutes">5</ttl>
        </expiry>
        <resources>
            <heap unit="kB">10</heap>
        </resources>
    </cache>
</config>
```

Рисунок 8 – Добавление зависимостей

В этом файле конфигурации создаем кеш, называемый задачами, который сможет хранить «List», любой тип, и будет хранить его в кэше в течение пяти минут. Ehcache сохранит этот список в куче с максимальным размером 10 КБ.

Чтобы убедиться, что загрузка Spring загружает этот файл конфигурации, можно добавить «spring.cache.jcache.config» свойство (рис.9).

```
spring.cache.jcache.config=classpath:ehcache.xml
```

Рисунок 9 – Добавление свойства

Поскольку кеширование является необязательным, также необходимо явно включить кеширование с помощью «@EnableCaching» аннотации. Можем поместить эту аннотацию поверх основного класса (рис.10).

```
@EnableCaching // Add this
@SpringBootApplication
public class SpringBootEhcacheApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootEhcacheApplication.class, args);
    }
}
```

Рисунок 10 – Добавление аннотации

Теперь, когда кеширование настроено, можно начать использовать его в коде. Прежде всего, убедимся, что результат «TaskFacade.findAll()» кешируется. Для этого используем «@Cacheable» аннотацию (рис.11).

```
@Cacheable("tasks")
public List<TaskDTO> findAll() {
    log.info("Retrieving tasks");
    return List.of(
        new TaskDTO(1L, "My first task", true),
        new TaskDTO(2L, "My second task", false));
}
```

Рисунок 11 – Использование кеширования

Если перезапустить приложение и выполнить по адресу «http://localhost:8080/api/tasks» несколько операций, то можно увидить, что сообщение «Получение задач» появляется только один раз. Если подождать пять минут и попытаетесь снова, срок действия кеша истечет, и сообщение появится снова.

Следующий пример отлично работает, иногда все же нужно обойти кеш и получить фактическое значение в реальном времени. В этом случае используем «condition» аргумент @Cacheable (рис.12).

```
@Cacheable(value = "tasks", condition = "!#noCache")
public List<TaskDTO> findAll(boolean noCache) {
    log.info("Retrieving tasks");
    return List.of(
        new TaskDTO(1L, "My first task", true),
        new TaskDTO(2L, "My second task", false));
}
```

Рисунок 11 – Пример обхода кэша

В этом примере используем кешированное значение, где «noCache» имеет значение «false». Если дать значение «true», кеш будет пропущен, и фактическая реализация будет вызвана снова.

Поскольку загрузка Spring использует параметры для определения ключа кэшированного значения, необходимо изменить тип ключа в файле «ehcache.xml» с «org.springframework.cache.interceptor.SimpleKey» на «java.lang.Boolean».

Spring будет кэшировать результат метода на основе переданных параметров. В первом примере не было никакого параметра, и в этом случае он использовал «SimpleKey».

Во втором примере был добавили «noCache» параметр, и, таким образом, результаты кэшируются на основе значения этого параметра.

В данном случае использовать «noCache» параметр в качестве значения ключа бесполезно (рис.12).

```
@Cacheable(value = "tasks", condition = "!#noCache", key = "'ALL'")
public List<TaskDTO> findAll(boolean noCache) {
    log.info("Retrieving tasks");
    return List.of(
        new TaskDTO(1L, "My first task", true),
        new TaskDTO(2L, "My second task", false));
}
```

Рисунок 12 – Установление ключа

Также необходимо снова изменить тип ключа, на этот раз с «java.lang.Boolean» на «java.lang.String».

Однако не обязательно использовать жестко запрограммированное значение. Как и «condition» параметр, можно использовать язык выражений Spring для определения ключа. Например, можно использовать «!#noCache» для инвертирования логического значения и так далее.

Таким образом, было рассмотрели несколько вопросов, касающихся кеширования с помощью EhCache и Spring.

Библиографический список

1. Прохоров П.В., Разговоров Н.В. Современные подходы в backend разработке на примере онлайн-магазина // Прикладная математика и фундаментальная информатика. 2020. №2. С. 23-28.
2. Егунова А.И., Аббакумов А.А., Воропаева М.А., Вечканова Ю.С. Система управления хранилищем электронных образовательных ресурсов // Образовательные технологии и общество. 2019. №3. С. 145-154.
3. Жданова В.С. Серверный модуль системы уведомления об изменениях в расписании занятий // Молодость. Интеллект. Инициатива. 2017. С. 21-22.
4. Нарижный А.Д., Губенко Н.Е. Сравнительный анализ стеков технологий spring и javaee (jakartaee) для разработки enterprise приложений // Информатика, управляющие системы, математическое и компьютерное моделирование 2020. №3. С. 549-462.
5. Волушкова В.Л. Архитектурные решения java для доступа к данным // Теоретические основы программирования. Учебное пособие 2019. 137с.