УДК 004

Управление транзакциями в Spring Boot

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлева Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
Данная статья содержит материал для понимания принципа работы и управлением транзакций в Spring Boot. Программа написана на языке программирования Java, с использованием фреймворка Spring Boot. Результатом исследования станет готовый алгоритм для управления транзакциями.
Ключевые слова: Java, Spring Boot, транзакции.

Transaction Management in Spring Boot

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Erovleva Regina Victorovna
Sholom-Aleichem Priamursky State University
Student

Abstract
This article contains material for understanding the principle of operation and transaction management in Spring Boot. The program is written in the Java programming language using the Spring Boot framework. The result of the study will be a ready-made algorithm for managing transactions.
Keywords: Java, Spring Boot, transactions.

1 Введение

1.1 Актуальность

В компьютерном программировании под транзакцией обычно понимается последовательность обмена информацией и связанной с этим работы, который рассматривается как единица в целях удовлетворения запроса и обеспечения целостности данных. Чтобы транзакция была завершена, а изменения в базе данных стали постоянными, транзакция должна быть завершена полностью. Под транзакцией можно понимать к

примеру заказ покупателя товара, который он оформил, через звонок в компанию, а менеджер внес информацию в базу. Транзакция заказа включает в себя проверку базы данных запасов, подтверждение наличия товара, размещение заказа, подтверждение размещения заказа и ожидаемого времени отгрузки. Если мы рассматриваем это как одну транзакцию, то все шаги должны быть завершены до того, как транзакция будет успешной, и база данных будет фактически изменена, чтобы отразить новый порядок. Если что-то произойдет до того, как транзакция будет успешно завершена, то транзакция будет отклонена и данные, которые вносились раньше не будут обработаны системой. Если говорить простым языков, то транзакции позволяют вносить изменения в данные без перебоев.

1.2Обзор исследований

В.Л. Волушкова провела обзор технологий на примере языка java и привела примеры с подробным описанием для использования фреймворка SpringBoot [1]. Д.В. Козырев, Л.А. Володченкова разработали программный интерфейс серверной части для облачного хранилища данных [2]. А.С. Волков и К.А. Волкова произвели краткий обзор элементов трехуровневой архитектуры для современных Web-приложений [3]. М.А. Потовиченко, М.В. Привалов и С.В. Корнев рассмотрели разработку программного продукта, который обеспечивает учет данных посещения занятий студентов, а также защиту их работ [4]. А.Д. Нарижный и Н.Е. Губенко провели сравнительный анализ технологий, которые имеют схожую функциональность и предназначены для одинаковых задач на технологии стеков JavaEE и Spring [5].

1.3Цель исследования

Цель исследования – используя язык программирования Java и фреймворк Spring Boot реализовать алгоритмы для понимая принципа работы транзакций и их управлением.

2 Материалы и методы

Для создания программы используется фреймворк Spring Boot, и язык программирования java.

3 Результаты и обсуждения

Существует 2 типа управления транзакциями, а именно:

- декларативный;
- программный.

Декларативное управление транзакциями — это наиболее широко используемый метод, при котором разработчики указывают платформе обрабатывать транзакции. Это может быть достигнуто 2 способами, через XML-конфигурации или через аннотации

Реализуем управление через аннотации. Для этого необходимо включить 2 аннотации.

Аннотация «@EnableTransactionManagement» она находится в основном классе.

Добавление аннотации «@EnableTransactionManagement» создает «PlatformTransactionManager» — с его автоконфигурациями JDBC (рис.1).

```
@SpringBootApplication
@EnableTransactionManagement
public class Application {
  public static void main(String[] args) {
    SpringApplication.run(Application.class, args);
  }
}
```

Рисунок 1 – Включение аннотации ««@EnableTransactionManagement»

Следующая аннотация «@Transactional», включается для класса обслуживания или метода.

Использование аннотации «@Transactional» на уровне класса помечает класс как транзакционный (рис.2).

```
@Transactional
public class UserService {

    public User getUser(String id) {
      // код для получения пльзователя

    }    public User saveUser(User user) {
      // код для созранения пользователя

    }
```

Рисунок 2 – Включение аннотации «@Transactional» в классе

Использование аннотации «@Transactional» для метода также помечает этот класс как транзакционный (рис.3).

```
public class UserService {
    @Transactional
    public User getUser(String id) {
      // код для получения пльзователя

    }
}
```

Рисунок 3 – Включение аннотации «@Transactional» в методе

Когда «UserService» автоматически подключается к любому контроллеру или другим классам, прокси UserService создается и используется этими классами (рис.4).

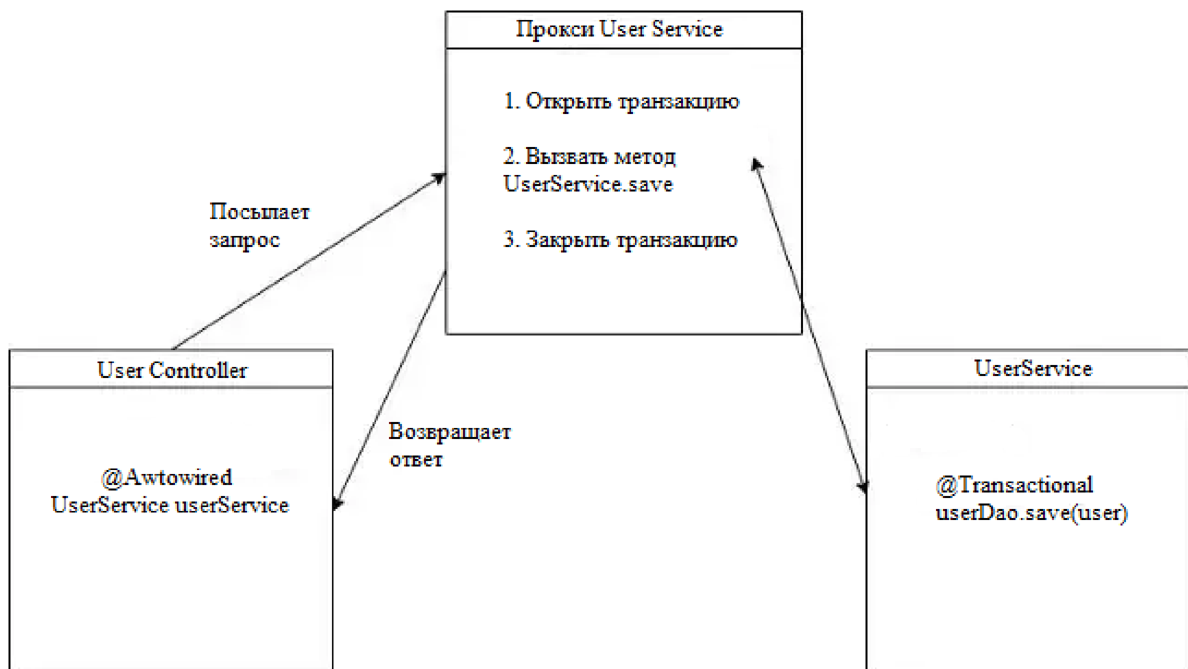

Рисунок 4 – Схема взаимодействия

1. Если Spring обнаруживает аннотацию «@Transactional» в bean-компоненте, он создает динамический прокси-сервер этого bean-компонента.
2. Прокси-сервер имеет доступ к диспетчеру транзакций (который является «PlatformTransactionManager» в Spring Boot) и будет запрашивать открытие и закрытие транзакций или соединений.
3. Сам менеджер транзакций просто использует соединение JDBC.

Теперь рассмотрим процессе отката.

Если сегмент кода внутри транзакционного метода вызывает исключение или ошибку среды выполнения, платформа автоматически откатывает транзакцию.

В приведенном ниже фрагменте кода «NullPointerException» генерируется кодом, который является исключением времени выполнения, и, следовательно, транзакция откатывается (рис.5).

```
public class UserService {@Autowired
UserRespository userRepository;@Transactional
    public User getUser(String id) throws   UserNotEligibleToDriveException{
        User user = userRepository.get(id);//Предположим, что пользовательский объект имеет значение null
//Ответ получаем NullPointerException
}
```

Рисунок 5 – Ошибка выполнения транзакции

Однако транзакция не откатывается для проверенных исключений. В следующем фрагменте кода, в случае проверенного исключения

записывается номер мобильного телефона пользователя, но транзакция не откатывается, так как код вызывает «Checked Exception» (рис.6).

```
public class UserService {@Autowired
UserRespository userRepository;@Transactional
    public User getUser(String id) throws   UserNotEligibleToDriveException{
        User user = userRepository.get(id);        user.setMobile("12345678");      //Checked Exception
        if(user.age < 18)
          throw UserNotEligibleToDriveException();
    }
}
```

Рисунок 6 – Исключение «Checked Exception»

Поскольку пользователь является постоянным объектом, «hibernate» сохраняет объект пользователя в БД, и новый номер телефона обновляется, даже если метод сохранения не выполняется. Но путь не правильный, так как нельзя сохранять какую-либо информацию и не совершать транзакцию.

Необходимо обязательно откатить транзакцию для описанного выше сценария. Это можно добиться, используя атрибут «rollbackFor» и задать ему параментр: «rollbackFor=Exception.class», чтобы сообщить Spring Boot об откате транзакции для любых проверенных исключений. Таким образом, не нужно будет указывать все проверенные исключения, генерируемые этим методом (рис.7).

```
@Transactional(rollbackFor=UserNotEligibleToDriveException.class)
    public User getUser(String id) throws   UserNotEligibleToDriveException{
    }
```

Рисунок 7 – Метод отката транзакции

Можно так же воспользоваться и другим методом - расширить «RunTimeException», и не нужно указывать атрибут «rollbackFor» (рис.).

```
public class UserNotEligibleToDriveException extends RunTimeException{}public class UserService {@Autowired
UserRespository userRepository;@Transactional
    public User getUser(String id){
        User user = userRepository.get(id);user.setMobile("12345678");//Checked Exception
        if(user.age < 18)
          throw UserNotEligibleToDriveException();
}
```

Рисунок 8 – Метод отката транзакции

Приведенный выше код будет откатан, потому что используется расширение класса «UserNotEligibleToDriveException» от класса «RunTimeException».

Но все же, как правило, безопаснее всего использовать атрибут «rollbackFor» для отката проверенных исключений.

Выше было рассмотрено декларативное управление транзакциями. Но есть сценарии, в которых это не работает, и тогда можно использовать программное управление транзакциями, где разработчики проводят границы транзакций внутри методов класса службы.

Рассмотрим следующий фрагмент кода. Сделаем несколько вызовов БД, затем вызовем внешний API, затем снова сделаем вызовы БД, а затем сделаем внутренний вызов API (рис.9).

```
@Transactional
public void completeOrder(OrderRequest request) {
    generateOrder(request); // вызов БД
    generateInvoice(request); // вызов БД
    paymentApi(request); // вызов внешнего API
    savePaymentInfo(request); // вызов БД
    createSubscription(request) // вызов БД
    sendInvoiceEmail() // внутренний вызов API
}
```

Рисунок 9 – Транзакции вызовов

Аннотация «@Transactional» создает новый «EntityManager» и запускает новую транзакцию, заимствуя одно соединение из пула соединений.

Соединение открывается и начинает генерировать ордера до тех пор, пока не будет выполнен весь фрагмент кода метода «completeOrder».

Если в платежном API есть какая-либо задержка, соединение блокируется и не возвращается в пул соединений.

Если есть одновременные пользователи, выполняющие заказы, у приложения могут закончится соединения с базой данных, что приведет к задержке приложения.

Следовательно, нельзя смешивать вызовы БД и ввода-вывода внутри транзакционного метода.

Очень хороший вариант использования программного управления транзакциями, это использовать «TransactionTemplate» (рис.10).

```
@Autowired private TransactionTemplate template;
public void completeOrder(OrderRequest request) { template.execute(
    status -> {
      generateOrder(request); // вызов БД
      generateInvoice(request); // вызов БД      получаем запрос;
    });

    paymentApi(request); // вызов внешнего API
    status -> {
      savePaymentInfo(request); // вызов БД
      createSubscription(request) // вызов БД       получаем запрос;
    });

    sendInvoiceEmail() // вызов внутреннего API
}
```

Рисунок 10 – Метод с использованием «TransactionTemplate»

В приведенном выше фрагменте не использовалась аннотация «@Transactional», но дважды использовался «TransactionTemplate» и оборачивание внутри операции БД.

В данном примере выполняются шаги:

- Шаблон запускает новую транзакцию с заимствованным соединением и выполняет «generateOrder» и «generateInvoice» в одной транзакции.
- Затем транзакция закрывается, и соединение возвращается в пул соединений.
- Создается вызов платежного API
- Затем снова шаблон запускает новую транзакцию с заимствованным соединением и выполняет «savePaymentInfo» и «createSubscription» в одной транзакции.
- Затем транзакция закрывается, и соединение возвращается в пул соединений.
- Затем выполняется внутренний вызов API, и ответ отправляется пользователю.

В данной статье были рассмотрены рабочие алгоритмы для понимания принципа работы и управлением транзакций в Spring Boot.

Библиографический список

1. Волушкова В.Л. Архитектурные решения java для доступа к данным// Тверской государственный университет. 2019. С. 137-140.
2. Володченкова Л.А., Козырев Д.В. Разработка серверной части программного приложения для удаленного хранения данных// Математические структуры и моделирование. 2020. № 1 (53). С. 108-138.
3. Волков А.С., Волкова К.А. Обзор архитектурных компонентов современного веб-приложения // Аллея науки. 2019. № 1(28). С. 958-961.
4. Потовиченко М.А., Привалов М.В., Корнев С.В. Компьютеризированная подсистема учета текущей успеваемости студента в условиях вуза // Информатика, управляющие системы, математическое и компьютерное моделирование. 2019. № 2. С. 71-75.
5. 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. № 17. С. 459-462.