Семинар №8
Spring AOP. Управление транзакциями

1. Инструментарий:

Урок
Презентация

---

2. Цели семинара №8:

- Получить обзор основных концепций Aspect-Oriented Programming (AOP).
- Закрепить практические навыки создания аспектов с использованием аннотаций и XML-конфигурации.

По итогам семинара №8 слушатель должен **знать**:

- Что такое AOP и какие проблемы он помогает решать.
- Как работает Spring AOP и как он вписывается в экосистему Spring.
- Как определить и использовать различные виды advice и срезов (pointcuts).

По итогам семинара №8 слушатель должен **уметь**:

- Разрабатывать аспекты для выполнения перекрестных задач в приложении.
- Применять различные типы советов и срезов для выполнения нужных действий в определенных точках приложения.
- Находить и устранять распространенные ошибки и проблемы в настройке и использовании Spring AOP.

---

3. План Содержание:

Этап урока	Тайминг, минуты	Формат
Введение, обзор темы	20	Модерирует преподаватель
Задание 1	40	Студенты выполняют, преподаватель помогает в решении проблем

Задание 2	40	Студенты выполняют, преподаватель помогает в решении проблем
Вопросы и обсуждение	20	Модерирует преподаватель
Длительность:	**120**	

4. Блок 1.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:

Вам необходимо создать простое Spring Boot приложение и использовать Spring AOP для логгирования времени выполнения всех методов с аннотацией `@LoggedExecution`.

1. Создайте аннотацию `@LoggedExecution`.
2. Создайте aspect, который перехватит вызовы методов с этой аннотацией и логгирует время их выполнения.
3. Создайте простой сервис с несколькими методами. Отметьте некоторые из них аннотацией `@LoggedExecution`.
4. Запустите приложение и проверьте, чтобы сообщения о времени выполнения методов действительно появлялись в логах.

Пример решения:

1. Аннотация:
```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface LoggedExecution {
}
```

2. Aspect:
```
@Component
@Aspect
public class LoggingAspect {

    @Around("@annotation(com.example.LoggedExecution)")
```

```
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws
Throwable {
        long start = System.currentTimeMillis();
        Object proceed = joinPoint.proceed();
        long executionTime = System.currentTimeMillis() - start;
        System.out.println(joinPoint.getSignature() + " executed in " +
executionTime + "ms");
        return proceed;
    }
}
```

3. Сервис:

```
@Service
public class SimpleService {

    @LoggedExecution
    public void someMethod() {
        // Some logic here
    }

    public void anotherMethod() {
        // Some logic here
    }

    @LoggedExecution
    public void yetAnotherMethod() {
        // Some logic here
    }
}
```

4. Убедитесь, что у вас подключены необходимые зависимости для Spring AOP и что настройка в `application.properties` или `application.yml` позволяет вам видеть логи.

Часто встречающиеся ошибки:

1. Забыть добавить зависимость для Spring AOP в `pom.xml` или `build.gradle`.
2. Забыть объявить класс с аспектом как компонент с аннотацией `@Component`.

3. Забыть указать корректное пространство имен для аннотации при определении среза в aspect (например, `@Around("@annotation(com.example.LoggedExecution)"`).
4. Не верно настроенный уровень логгирования, из-за чего сообщения не появляются в логах.

5. Блок 2.

Тайминг:
Объяснение правил – 10 минут
Работа в команде – 30 минут

Задание:
Задание:

Необходимо создать функционал для отслеживания исключений, возникающих в слое сервиса вашего Spring Boot приложения. Используйте Spring AOP для перехвата исключений и логгирования информации о методе, в котором произошло исключение, а также о самом исключении.

1. Создайте аннотацию `@LogException`.
2. Создайте aspect, который будет перехватывать исключения, возникающие в методах, помеченных этой аннотацией, и логгировать необходимую информацию.
3. Примените аннотацию `@LogException` к методам в слое сервиса, где вы хотите отслеживать исключения.

Пример решения:

1. Аннотация:
```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface LogException {
}
```

2. Aspect:
```
@Component
@Aspect
public class ExceptionLoggingAspect {
```

```
   @AfterThrowing(pointcut = "@annotation(com.example.LogException)",
throwing = "ex")
   public void logException(JoinPoint joinPoint, Exception ex) {
      String methodName = joinPoint.getSignature().toString();
      System.err.println("Exception in method " + methodName + ": " +
ex.getMessage());
   }
}
```

3. Сервис:

```
@Service
public class MonitoredService {

   @LogException
   public void potentiallyFailingMethod() {
      // Some logic that might throw an exception
      if (true) { // example condition
         throw new RuntimeException("Oops, something went wrong!");
      }
   }
}
```

Часто встречающиеся ошибки:

1. Ошибка в определении среза для аннотации `@LogException`. Необходимо указать полное имя класса, например, `@AfterThrowing(pointcut = "@annotation(com.example.LogException)", throwing = "ex")`.
2. Забывание добавить `throwing = "ex"` в определении `@AfterThrowing`, что приводит к тому, что исключение не может быть получено и залоггировано.
3. Применение аннотации `@LogException` к методам, которые не генерируют исключения, что делает аннотацию бессмысленной в этом контексте.
4. Не учитывание всех типов исключений, которые могут возникать в методе, что может привести к неполным или неточным логам.

6. Домашнее задание

Вам необходимо разработать механизм регистрации действий пользователя в вашем Spring Boot приложении. Используйте Spring AOP

для создания журнала действий, в котором будет сохраняться информация о том, какие методы сервиса вызывались, кем и с какими параметрами.

1. Создайте аннотацию `@TrackUserAction`.
2. Реализуйте aspect, который будет регистрировать действия пользователя, когда вызывается метод, отмеченный этой аннотацией.
3. Примените аннотацию `@TrackUserAction` к нескольким методам в слое сервиса.
4. Результаты регистрации сохраните в лог-файл.

Пример решения:

1. Аннотация:
```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface TrackUserAction {
}
```

2. Aspect:
```
@Component
@Aspect
public class UserActionTrackingAspect {

   private static final Logger logger =
LoggerFactory.getLogger(UserActionTrackingAspect.class);

   @Before("@annotation(com.example.TrackUserAction)")
   public void trackUserAction(JoinPoint joinPoint) {
      String user = "currentUser"; // Пример. На практике следует получать
пользователя из контекста безопасности или сессии.
      String methodName = joinPoint.getSignature().toString();
      Object[] args = joinPoint.getArgs();
      logger.info("User " + user + " invoked " + methodName + " with
arguments: " + Arrays.toString(args));
   }
}
```

3. Сервис:
```
@Service
```

```
public class UserService {

    @TrackUserAction
    public void createUser(String username, String password) {
        // Создание пользователя
    }

    @TrackUserAction
    public void deleteUser(String username) {
        // Удаление пользователя
    }
}
```

Рекомендации для преподавателей по оценке задания:

1. Убедитесь, что студент понимает концепции AOP и почему он выбрал определенный подход для решения задачи.
2. Проверьте, используются ли аннотации на соответствующих методах.
3. Убедитесь, что код аспекта корректно регистрирует необходимую информацию и не вызывает побочных эффектов.
4. Проверьте, корректно ли настроено логгирование, и убедитесь, что логи содержат необходимую и актуальную информацию.
5. Проверьте, обрабатываются ли потенциальные ошибки или исключения, которые могут возникнуть при работе аспекта.
6. Можно дать дополнительные баллы за продвинутые функции, такие как фильтрация логируемой информации, дополнительная настройка аспектов или использование других возможностей Spring AOP.