GeekBrains

Тестирование Backend на Java

ORM: почему JDBC иногда недостаточно

На этом уроке

1. Вспомним всё, что связано с БД.
2. Обсудим JPA.
3. Познакомимся с фреймворком MyBatis.
4. Научимся встраивать проверки в код тестов.

Оглавление

Введение

На этом уроке мы продолжим работать с проектом mini-market. Подключимся к базе данных и сделаем дополнительные проверки, используя ORM — объектную модель БД в JAVA-классах. Разбирать ORM будем на примере MyBatis: посмотрим, какие возможности она предоставляет, как использовать БД в тестах, ассертах, а также в подготовке и удалении данных.

Что такое ORM?

Если не работать исключительно с базами данных NoSQL, то, вероятно, появятся довольно много SQL-запросов. Обычно они выглядят так:

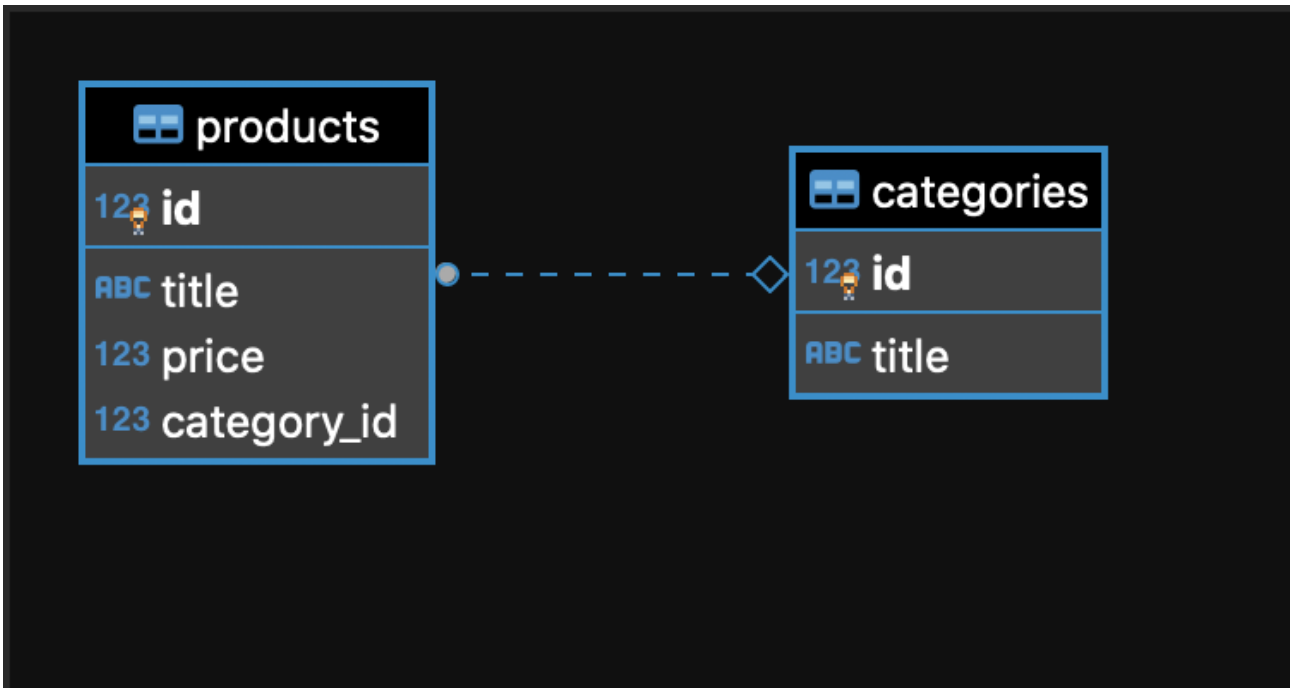
```
SELECT * FROM users WHERE email = 'test@test.com';
```

Объектно-реляционное сопоставление (маппинг) — это идея возможности писать запросы, подобные приведённому выше, а также гораздо более сложные запросы, используя объектно-ориентированную парадигму предпочтительного языка программирования. Мы пытаемся взаимодействовать с нашей базой данных, используя выбранный нами язык вместо SQL. Ниже разберём синтаксис и основные особенности нашей ORM — MyBatis.

Больше об ORM — в «Википедии» и на Habr.

О базе данных проекта

Схема нашей базы данных, а здесь используется СУБД Postgres, очень проста и состоит из двух бизнес-таблиц: categories и products. Данные для подключения есть в личном кабинете в информации к уроку.



У нас есть один foreign key в таблице products — это category_id, ссылка на поле id в таблице categories.

Такая простая схема вполне достаточна для построения бизнес-логики приложения и хранения всех данных. На её примере мы научимся делать CRUD-запросы к базе из кода. Чтобы посмотреть, как устроена БД, можно подключиться либо через pgAdmin, который включён в установщике Postgres, либо через любого другого графического клиента, например, Dbeaver CE или плагин Intellij IDEA.

Начало работы с MyBatis

Для работы потребуется установить следующие зависимости в тегах dependencies:

```
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <version>42.2.18</version>
</dependency>
<dependency>
   <groupId>org.mybatis</groupId>
   <artifactId>mybatis</artifactId>
   <version>3.5.6</version>
</dependency>
```

Первая подтягивает JDBC-драйвер для PostgreSQL, вторая добавляет сам фреймворк MyBatis. Сразу добавим плагин для автогенерации MyBatis в тегах plugins, принцип его использования будет изложен ниже.

```
  <!--   it  is used for DB objects (dao, mappers) generation    -->
   <plugin>
       <groupId>org.mybatis.generator</groupId>
       <artifactId>mybatis-generator-maven-plugin</artifactId>
       <version>1.4.0</version>
       <executions>
           <execution>
               <id>Generate MyBatis Artifacts</id>
               <goals>
                   <goal>generate</goal>
               </goals>
           </execution>
       </executions>
   </plugin>
</plugins>
</pluginManagement>
<resources>
   <resource>
       <directory>src/main/java</directory>
       <includes>
           <include>**/*.xml</include>
       </includes>
   </resource>
   <resource>
       <directory>src/main/resources</directory>
       <includes>
           <include>**/*.xml</include>
       </includes>
   </resource>
</resources>
```

Конфигурация MyBatis

Самый распространённый способ конфигурации фреймворка MyBatis — это xml-файл, основные теги которого полностью объяснены в официальной документации. Назовём его mybatis-config.xml и поместим в src/main/resources. Приведём пример для нашего проекта:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
       PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
       "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
   <environments default="qa">
       <environment id="qa">
           <transactionManager type="JDBC"/>
           <dataSource type="POOLED">
               <property name="driver" value="org.postgresql.Driver"/>
               <property name="url"
                       value="jdbc:postgresql://80.78.248.82:5432/coursedb"/>
```

```
            <property name="username" value="postgres"/>
            <property name="password" value="postgres"/>
        </dataSource>
    </environment>
  </environments>
</configuration>
```

Правильная конфигурация — в информации к уроку в личном кабинете. Разберём эти теги подробнее.

Вся конфигурация находится в корневых тегах <configuration>. Можно описать один или несколько environment. Обычно это используется для разных стендов, например, dev, qa, prod. У нашего продукта один стенд, мы условно назвали его qa, так как здесь проводится тестирование.

У transactionManager две конфигурации, JDBC и MANAGED. По умолчанию мы всегда выбираем первую, так как вторая не делает никаких транзакций или откатов транзакций самостоятельно. Она применяется, если управление транзакциями выполняет другая платформа, например, Spring или JEE Application Server.

dataSource имеет три значения параметра type: POOLED, UNPOOLED и JNDI.

1. POOLED — реализация DataSource соединения JDBC. Она используется, чтобы избежать задержек во времени для начального соединения и аутентификации. Делается это для создания нового экземпляра соединения. Это популярный подход для конкурентных веб-приложений, чтобы достичь максимально быстрого отклика.
2. UNPOOLED — реализация DataSource, которая открывает и закрывает соединение при каждом запросе. Хотя она немного медленнее, это хороший выбор для простых приложений, которым не требуется производительность быстро доступных подключений. Различные базы данных также различаются в области производительности. Поэтому для некоторых объединение в пул может быть менее важным, и эта конфигурация будет идеальной.
3. JNDI — реализация источника данных, предназначенная для использования с такими контейнерами, как EJB или серверы приложений. Последние настраивают источник данных централизованно или извне и размещают ссылку на него в контексте JNDI.

Каждая конфигурация dataSource имеет собственные параметры. Мы разберём несколько важных параметров для конфигураций POOLED и UNPOOLED, они записываются в тегах property:

- driver — здесь указывается полное название java-класса нашего JDBC-драйвера;
- url — здесь указываем JDBC URL-подключения к БД;
- username — имя пользователя для БД;

- password — пароль от пользователя БД.

Теперь подключимся к нашей БД, используя этот конфиг. Создадим класс с main-методом и напишем следующий код:

```
public class Main {
   public static void main(String[] args) throws IOException {
        SqlSessionFactory sqlSessionFactory;
        //NyBatis Configuration file
        String resource = "mybatisConfig.xml";
        InputStream is = Resources.getResourceAsStream(resource);
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(is);
}
}
```

Мы создали и инициализировали сессию подключения к БД, применяя наш конфиг.

Работа с мапперами. MyBatis generator

Особенность MyBatis и основное отличие от старого и известного ORM-фреймворка Hibernate — особенности маппинга. Если Hibernate и большинство других ORM осуществляют маппинг java-объектов на таблицы базы данных, представляя таким образом доступ к данным, то MyBatis производит маппинг объектов на результаты SQL-запросов. Ответственность за это ложится на плечи разработчиков, то есть на нас. Проще говоря, если наша база данных станет меняться, таблицы ждут значимые изменения.

Если таблицы содержат много технических полей, и/или БД не нормализована, лучше производить маппинг на собственноручно написанные запросы. Вообще маппер — это тоже xml-файл с описанием запросов. Например, select по id будет выглядеть следующим образом:

```
<select id="selectByPrimaryKey" parameterType="java.lang.Long"
resultMap="BaseResultMap">
 select
 <include refid="Base_Column_List" />
 from products
 where id = #{id,jdbcType=BIGINT}
</select>
```

Для построения базовых мапперов и моделей используется плагин-генератор, в начале урока именно его мы добавляли в plugins. Для его настройки понадобится xml-файл, который будет храниться в src/main/resources и содержать информацию о наших таблицах. В файле обычно много настроек генератора, в целом он выглядит следующим образом:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE generatorConfiguration
       PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
       "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd">

<generatorConfiguration>
   <classPathEntry

location="/Users/<username>/.m2/repository/org/postgresql/postgresql/42.2.18/pos
tgresql-42.2.18.jar"/>

   <context id="DB2Tables" targetRuntime="MyBatis3">
       <jdbcConnection driverClass="org.postgresql.Driver"

connectionURL="jdbc:postgresql://80.78.248.82:5432/coursedb"
                       userId="postgres"
                       password="postgres">
       </jdbcConnection>

       <javaModelGenerator targetPackage="ru.geekbrains.java4.lesson6.db.model"
targetProject="src/main/java">
           <property name="enableSubPackages" value="true"/>
           <property name="trimStrings" value="true"/>
       </javaModelGenerator>

       <sqlMapGenerator
targetPackage="ru.geekbrains.java4.lesson6.db.mapper.xml"
targetProject="src/main/java">
           <property name="enableSubPackages" value="true"/>
       </sqlMapGenerator>

       <javaClientGenerator type="XMLMAPPER"
targetPackage="ru.geekbrains.java4.lesson6.db.dao"
                            targetProject="src/main/java">
           <property name="enableSubPackages" value="true"/>
       </javaClientGenerator>

       <table tableName="categories">
           <property name="useActualColumnNames" value="true"/>
           <generatedKey column="id" sqlStatement="DB2" identity="true"/>
       </table>
       <table tableName="products">
           <property name="useActualColumnNames" value="true"/>
           <generatedKey column="id" sqlStatement="DB2" identity="true"/>
       </table>
   </context>
</generatorConfiguration>
```

Чтобы понять, что мы указываем в этом файле, начнём с конца. В тегах table указываются наши таблицы. У обеих указано свойство useActualColumnNames, так как нам удобно работать в этом

проекте с исходными именами столбцов. Название колонок в коде можно заменить, используя тег columnOverride, например:

```
  <columnOverride column="DATE_FIELD" property="startDate" />
```

Или даже проигнорировать:

```
<ignoreColumn column="FRED" />
```

Идём далее, выше указываем местоположение для сохранения объектов dao, мапперов, моделей в тегах javaClientGenerator. Там указывается местоположение:

- сначала пакет (любой);
- затем — положение в проекте, по умолчанию всегда ставим src/main/java.

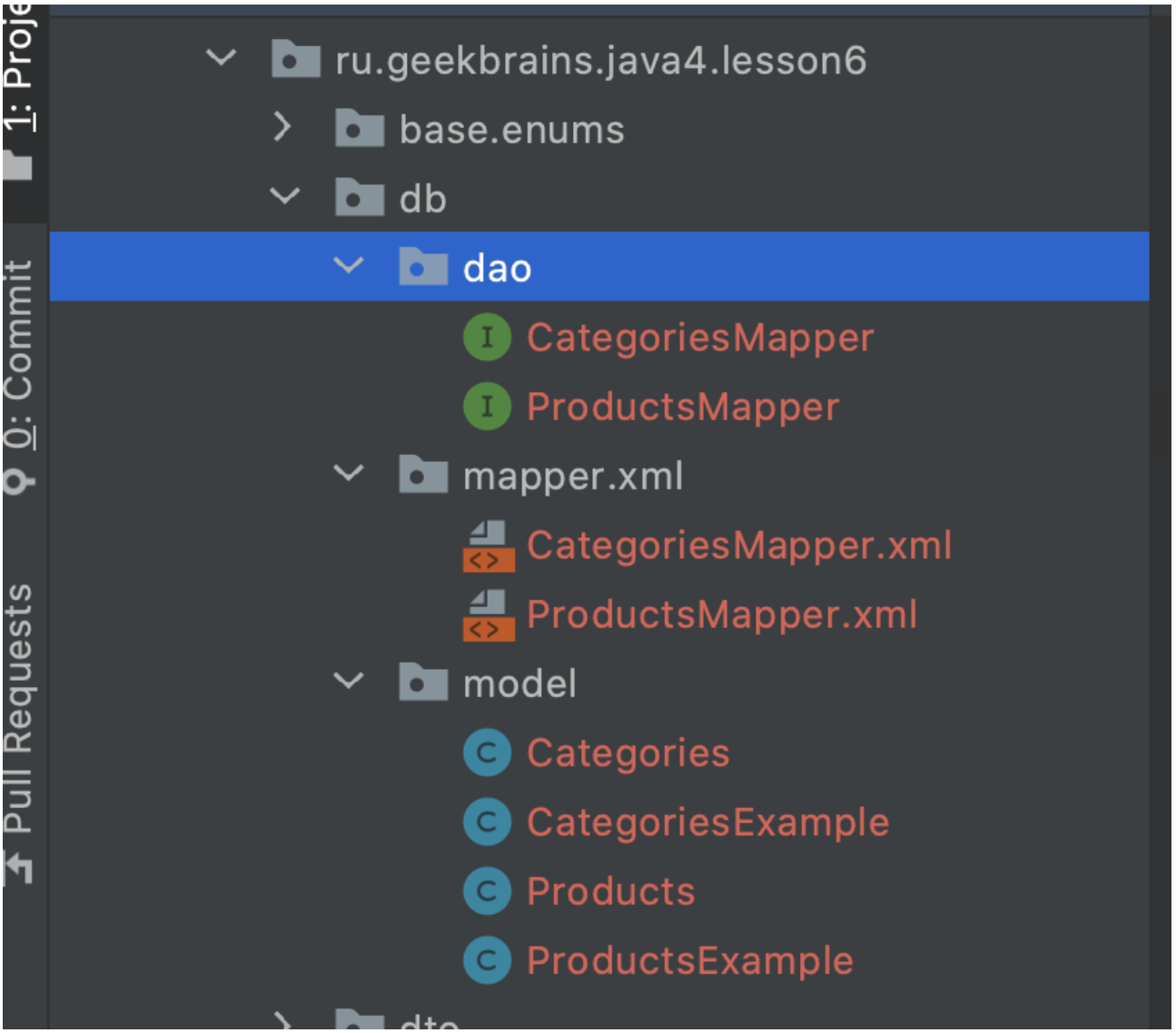
Чтобы xml-файлы читались во время исполнения программы, в настройках maven указываем, что из директории src/main/java надо читать ещё и xml-файлы:

```
    <resources>
            <resource>
                <directory>src/main/java</directory>
                <includes>
                    <include>**/*.xml</include>
                </includes>
            </resource>
            <resource>
                <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.xml</include>
                </includes>
            </resource>
        </resources>
```

Ещё выше — подробности подключения к БД, которые аналогичны тем, что мы указывали в основном конфиге. Запустить плагин можно через команду:

```
mvn mybatis-generator:generate
```

В результате у нас получится новый пакет db с таким содержимым:



Особый интерес представляет пакет model — вместо ожидаемых двух объектов Categories и Products здесь ещё есть аналогичные Example-объекты. Они используются, чтобы создать объект для фильтрации по критериям. Обычно мы это пишем в SQL-запросе блока where. Разберём на примере.

Написание запросов

Напишем простой метод подсчёта числа категорий в БД. Для этого в SQL есть функция count, а запрос выглядит следующим образом:

```
select count(*) from categories;
```

Вернёмся в наш Main-класс, напишем метод:

```
public static Integer countNumberOfAllCategories() {
  CategoriesMapper categoriesMapper = getCategoriesMapper(resource);
```

```
    CategoriesExample example = new CategoriesExample();
    return Math.toIntExact(categoriesMapper.countByExample(example));
}
```

После создания подключения надо инициализировать маппер для работы с конкретной таблицей, в нашем случае это categoriesMapper. После этого требуется создать Example-категории. Обойдёмся просто инициализацией без добавления критериев, так как блок where в исходном запросе отсутствует. Затем у объекта маппера потребуется вызвать подходящий метод (countByExample). Так как Example пустой, то посчитаются все действующие записи.

Для тестовых целей полезен также метод, получающий тот или иной маппер:

```
private static CategoriesMapper getCategoriesMapper(String resource)  {
    InputStream inputStream  = Resources.getResourceAsStream(resource);
    SqlSessionFactory sqlSessionFactory = new
            SqlSessionFactoryBuilder().build(inputStream);
    sqlSessionFactory.openSession();
    SqlSession session = sqlSessionFactory.openSession();
    return session.getMapper(CategoriesMapper.class);
}
```

Применение в тестах

1. Можно написать тесты на саму БД и её наполнение. Например, если после деплоя на базу «накатывается» эталонный дамп, который проверяется автоматически.
2. Использовать запросы к базе в фикстурах и ассертах.

Вот некоторые примеры использования:

1. Удаление тестовых данных через базу.
2. Получение данных через базу, особенно если их нельзя получить через API.
3. Проверка создания, обновления или удаления сущности.
4. Проверка полученной информации.
5. Проверка фильтрации по датам, времени или другим признакам.
6. Изменение данных через базу — подготовка для тестов или прямо в течение теста.
7. Создание данных через базу. Не рекомендуется, так как при изменении БД тесты будут падать ещё до прогона, а также велик риск создания не консистентных данных.

Практическое задание

Задача-минимум:

1. Встройте проверку данных через БД в ассерты к тестам, используя MyBatis ORM.

Задача-максимум:

1. Добавьте апдейт и удаление данных через БД в тестах и после них.

Сдайте ссылку на репозиторий и укажите ветку с кодом.

Дополнительные материалы

1. Официальная документация.
2. Статья «MyBatis как более быстрая альтернатива Hibernate».

Используемые источники

1. Официальная документация.
2. Java Persistence with MyBatis 3 by K. Siva Prasad Reddy.