Spring Boot 整合 JPA

上周回顾

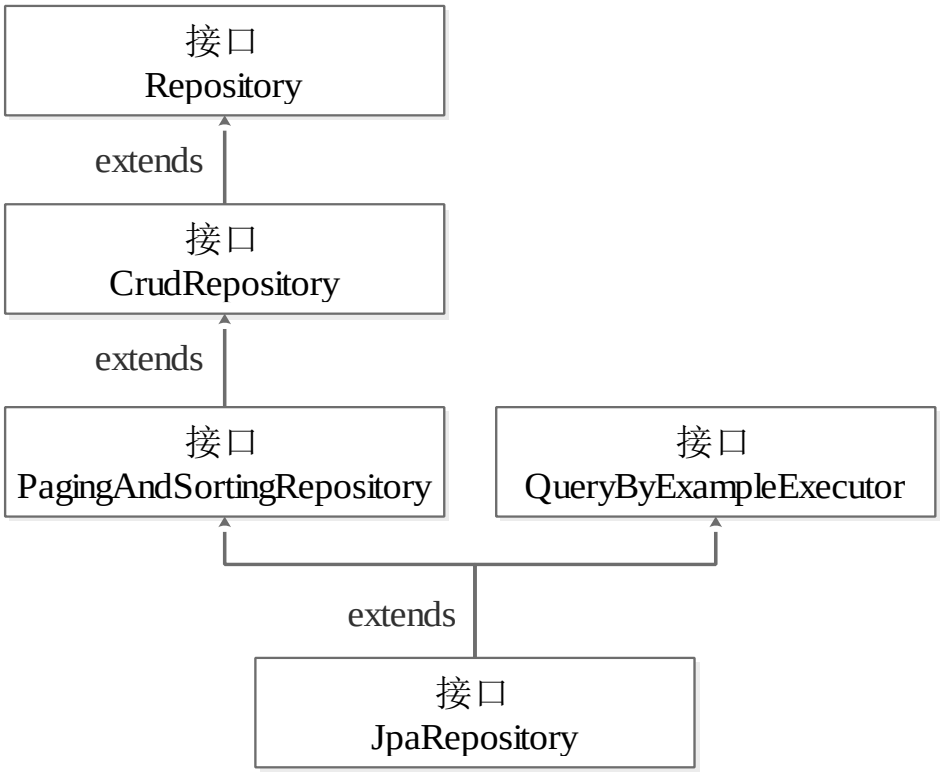
1. Spring Boot 数据访问-使用注解的方式整合 MyBatis
2. Spring Boot 数据访问-使用**配置文件方式整合 MyBatis**

一、Spring Data JPA 介绍

JPA 是 sun 公司官方提出的 Java 持久化规范，它为 java 开发人员提供一种对象/关系映射工具，管理 java 中的关系型数据。

Spring Data JPA 是 Spring 在 ORM 框架、JPA 规范的基础上封装的一套 JPA 应用框架，提供**增删改查等常用功能**，使开发者用**较少的代码**实现数据库操作，同时还易于扩展。

使用 Spring Data JPA 自定义的 Repository 接口必须继承 XXXRepository<**T,ID**>,其中 T 表示要操作的**实体类**，ID 表示**实体类主键的数据类型**。



Repository 接口是 Spring Data JPA 提供的用于自定义 Repository 接口的顶级父接口，该接口没有声明任何方法。

CrudRepository 接口是 Repository 的继承接口之一，包含一些基本的 CRUD 方法。

PagingAndSortingRepository 接口继承了 CrudRepository 接口的同时，提供分页

和排序两个方法

QueryByExampleExecutor 接口是进行条件封装查询的顶级父接口，运行通过 Example 实例执行复杂的条件查询。

JpaRepository 接口同时继承了 PagingAndSortingRepository 接口和 QueryByExampleExecutor 接口，并额外提供了一些数据操作方法，自定义 Repository 接口时，通常可以直接选择继承 JpaRepository 接口。

使用 Spring Data JPA 进行数据操作的多种实现方式：

1）如果**自定义接口**继承了 **JpaRepository 接口**，则默认包含了一些常用的 CRUD 方法。

2）自定义 Repository 接口中，可以使用**@Query** 注解配合 SQL 语句进行数据的查、改、删操作。

3）自定义 Repository 接口中，可以直接使用方法名关键字进行查询操作

下面例子中我们选择继承 JpaRepository 接口。

在自定义的 Repository 接口中，针对数据的变更操作（修改、删除），无论是否使用了@Query 注解，都必须在方法上方添加@Transactional 注解进行事务管理，否则程序执行就会出现 InvalidDataAccessApiUsageException 异常。

如果在调用 Repository 接口方法的业务层 Service 类上已经添加了@Transactional 注解进行事务管理，那么 Repository 接口文件中就可以省略@Transactional 注解。

变更操作，要配合使用@Query 与@Modify 注解

在自定义的 Repository 接口中，使用@Query 注解方式执行数据变更操作（修改、删除），除了要使用@Query 注解，还必须添加@Modifying 注解表示数据变更。

二、案例

1. 导入数据库

books 数据库创建语句第 6 周.txt

2. pom 文件

打开项目，打开 pom 文件

引入 spring-boot-starter-data-jpa 依赖

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

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

3. 创建 ORM 实体类

编写 ORM 实体类：实体类与数据表进行映射，并且配置好映射关系。

Java Persistence API 定义了一种定义，可以将常规的普通 Java 对象（有时被称作 POJO）映射到数据库， 这些映射到数据库的 Java 对象被称作 Entity Bean。

除了是用 Java Persistence 元数据将其映射到数据库外，Entity Bean 与其他 Java 类没有任何区别。

事实上，创建一个 Entity Bean 对象相当于新建一条记录，删除一个 Entity Bean 会同时从数据库中删除对应记录，修改一个 Entity Bean 时，容器会自动将 Entity Bean 的状态和数据库同步。

例如下面的 Books 类就是一个 Entity Bean

如果没有@Entity 和@Id 这两个注解的话，Entity Bean 完全就是典型的 POJO 的 java 类是，加上这两个注解后，就可以作为实体类和数据库中的表相对应。

如果类名和数据库中的表名不一样，可以在@Entity 注解中指定 name 属性值，指定对应的表格名，也可以使用@Table 注解的 name 属性指定。

@Entity 和@Table 的区别：

@Entity 说明这个 class 是实体类，并且使用默认的 orm 规则，即 class 名即数据库表中表名，class 成员变量名即表中的字段名

如果想改变这种默认的 orm 规则，就要使用@Table 来改变 class 名与数据库中表名的映射规则，@Column 来改变 class 中字段名与 db 中表的字段名的映射规则

```
@Entity(name="tb_book")
```

或

```
@Table(name="tbbook")
```

如果同时使用了上面两个注解，则@Table 注解优先。

优先级 @Table>@Entity

3.1 创建 TbComment 实体类

```
import javax.persistence.*;   //旧版
```

```
import jakarta.persistence.*;
import lombok.Data;
@Entity(name="comment")
@Data
```

```
public class TbComment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String content;
    private String c_author;
    private int bookId;
}
```

3.2 创建 Books 实体类

在 cn.edu.scnu.entity 包中创建 Books 实体类

```
package cn.edu.scnu.entity;
import jakarta.persistence.*;
import lombok.Data;
import java.util.ArrayList;
import java.util.List;
@Entity(name="tb_book")
@Data
public class Books {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="book_id")
    private Integer id;
    private String title;
    private String author;
    private String press;
    private Double price;
    @OneToMany(cascade = CascadeType.ALL,fetch = FetchType.EAGER)
    @JoinColumn(name="bookId")
    private List<TbComment> commentList = new ArrayList<>();
}
```

4. 复制静态资源

将 index2.html 复制到 resources 下

5. 控制器类

5.1 BookController 类

打开 BookController 类，添加 index2 方法

```
@GetMapping("/index2")
    public String index2(){
        return "index2";
}
```

5.2 创建 Book3Controller 类

```
package cn.edu.scnu.controller;

@Controller
@RequestMapping("/book3")
public class Book3Controller {
    @Autowired
    private Book3Service book3Service;
 @Autowired
    private Book3Service book3Service;

    @RequestMapping("/allbooks")
    @ResponseBody
    public List<Books> showbook(){
        return book3Service.showbook();
    }

    @RequestMapping("/insertBook")
    @ResponseBody
    public String insertBook(Books book){
        System.out.println(book);
        book3Service.insertTbbook(book);
        System.out.println(book);
        return "添加成功";
    }
    @RequestMapping("/updateBook")
    @ResponseBody
    public String updateBook(Books book){
        book3Service.updateBook(book);
        return "修改成功";
    }
    @RequestMapping("deleteBook")
    @ResponseBody
    public String deleteBook(Integer id){
        book3Service.deleteBook(id);
        return "删除成功";
    }
    @RequestMapping("/showComment")
    @ResponseBody
    public Books showComment(Integer id){
```

```
        return book3Service.showComment(id);
    }

}
```

6. 创建 service 类

创建 service 包，创建 BookService 类

```
package cn.edu.scnu.service;
import cn.edu.scnu.entity.Books;
import cn.edu.scnu.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Iterator;
import java.util.List;
@Service
public class BookService {
    @Autowired
    private BookRepository bookRepository;

        public List<Books> showbook() {
            return bookRepository.findAll();
        }


        public void insertTbbook(Books book) {
            bookRepository.save(book);
        }

        public void deleteBook(Integer id) {
            bookRepository.deleteById(id);
        }

        public void updateBook(Books book) {
            bookRepository.updateBook(book);
        }

        public Books showComment(Integer id) {
            return bookRepository.findByIdContain(id);
        }
}
```

7. 编写 Repository 接口-Spring Boot 整合 JPA

编写 Repository 接口：针对不同的表数据操作编写各自对应的 Repositor 接口，并根据需要编写对应的数据操作方法。

在 cn.edu.scnu.mapper 包中创建 BookRepository 接口

```
package cn.edu.scnu.repository;
```

```
import cn.edu.scnu.entity.Books;
import jakarta.transaction.Transactional;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
public interface BookRepository extends JpaRepository<Books,Integer> {
    @Transactional
    @Modifying
    @Query("update tb_book set
title=:#{#book.getTitle()},author=:#{#book.getAuthor()},press=:#{#book.getPress()},price=:#{
#book.getPrice()} where id=:#{#book.getId()}")
    void updateBook(Books book);

    @Query(value="select b.*,c.id c_id,c.content c_content,c.c_author from tb_book
b,comment c    where b.book_id=c.book_id and b.book_id=?1",nativeQuery = true)
    Books findByIdContain(Integer id);
}
```

8. 效果测试