第 5 章

Spring Boot 整合 MyBatis-Plus

MyBatis-Plus 是国内团队苞米豆在 MyBatis 的基础上开发的增强框架，扩展了一些功能，以提高效率。引入 MyBatis-Plus 不会对现有的 MyBatis 框架产生任何影响，而且 MyBatis-Plus 支持所有 MyBatis 原生的特性。

使用 MyBatis-Plus 可以无须编写 SQL 语句就能进行基本的 CRUD 操作。MyBatis-Plus 内置了 BaseMapper，提供了大量的 CRUD 方法，可以满足大部分单表的简单查询。

MyBatis-Plus 具有无侵入、损耗小等特性，支持强大的 CRUD 操作、支持 Lambda 形式调用、支持主键自动生成、支持 ActiveRecord 模式、支持自定义全局通用操作，具有内置代码生成器、内置分页插件、内置性能分析插件、内置全局拦截插件。

5.1 基本 CRUD 查询

MyBatis-Plus 封装了 BaseMapper 接口，MyBatis-Plus 启动时会自动解析实体表关系映射并转换为 MyBatis 内部对象注入容器。开发者只需创建数据访问层接口，继承 BaseMapper 就可直接使用。查看源码，可以看到 BaseMapper 提供的方法如下所示，各方法的作用见其注释。

```
//插入一条记录
int insert(T entity);
//根据 entity 条件,删除记录
int delete(@Param(Constants.WRAPPER) Wrapper<T> wrapper);
//删除(根据 ID 批量删除)
int deleteBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable>
idList);
//根据 ID 删除
int deleteById(Serializable id);
//根据 columnMap 条件,删除记录
int deleteByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
//根据 whereEntity 条件,更新记录
int update(@Param(Constants.ENTITY) T entity, @Param(Constants.WRAPPER) Wrapper<T>
updateWrapper);
//根据 ID 修改
int updateById(@Param(Constants.ENTITY) T entity);
//根据 ID 查询
```

```
T selectById(Serializable id);
//根据 entity 条件,查询一条记录
T selectOne(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);

//查询(根据 ID 批量查询)
List<T> selectBatchIds(@Param(Constants.COLLECTION) Collection<? extends Serializable>
idList);
//根据 entity 条件,查询全部记录
List<T> selectList(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
//查询(根据 columnMap 条件)
List<T> selectByMap(@Param(Constants.COLUMN_MAP) Map<String, Object> columnMap);
//根据 Wrapper 条件,查询全部记录
List<Map<String, Object>> selectMaps(@Param(Constants.WRAPPER) Wrapper<T>
queryWrapper);
//根据 Wrapper 条件,查询全部记录.注意: 只返回第 1 个字段的值
List<Object> selectObjs(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
//根据 entity 条件,查询全部记录(并翻页)
IPage<T> selectPage(IPage<T> page, @Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
//根据 Wrapper 条件,查询全部记录(并翻页)
IPage<Map<String, Object>> selectMapsPage(IPage<T> page, @Param(Constants.WRAPPER)
Wrapper<T> queryWrapper);
//根据 Wrapper 条件,查询总记录数
Integer selectCount(@Param(Constants.WRAPPER) Wrapper<T> queryWrapper);
```

在上述代码中,泛型 T 为任意实体对象,参数 Serializable 为任意类型主键,MyBatis-Plus 不推荐使用复合主键约定每张表都有自己的唯一 id 主键,对象 Wrapper 为条件构造器。

【例 5-1】 查询图书信息。

(1) 创建项目。创建 Spring Boot 项目 mybatisplus,导入 MyBatis-Plus 等依赖,pom.xml 文件中的关键代码如下:

```
//第 5 章/mybatisplus1/pom.xml
<dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>3.5.3</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>8.0.25</version>
    </dependency>
    <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
```

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

（2）创建实体类 Book，代码如下：

```
//第5章/mybatisplus1/Book.java
@Data
@AllArgsConstructor
@NoArgsConstructor
@TableName("book")
public class Book {
    @TableId(value = "id",type =  IdType.AUTO)
    private Integer id;

    @TableField("name")
    private String name;
    private double price;
    private String category;
    private int pnum;
    private String imgurl;
    private String description;
    private String author;
    private int sales;
}
```

其中注解@TableName 表示该实体类对应的数据库表名，如果实体类与表名不一致，则这一步是必要的，否则可以不添加该注解，默认表名与实体类名称相同。注解@TableId 表示实体类中对应数据库表的主键的属性，其中 type= IdType.AUTO 表示主键由数据库自增长。

（3）配置文件 application.properties。在此文件中配置的数据库连接信息内容跟 4.1 节案例相同。

此外也可全局设置表名的前缀，例如数据库中的表名有可能是带 t_开头的，但实体类一般不这样开头，所以会导致表名与实体类名称不一致，主要差别可能就是这个 t_，这时就可以在这里配置，参考代码如下：

```
mybatis-plus.global-config.db-config.table-prefix=t_
```

还可在这里全局设置主键的增长方式，参考代码如下：

```
mybatis-plus.global-config.db-config.id-type=auto
```

说明：上述配置非必要，如果实体类已经对表名进行了映射，并且主键也指定了增长策略，则无须上述两个配置。

其他常用配置的参考代码如下：

```
//第 5 章/mybatisplus1/application.yml
mybatis-plus:
    #扫描 XML 映射文件,多个目录用逗号或者分号分隔(告诉 Mapper 所对应的 XML 文件位置)
  mapper-locations: classpath:mapper/*.xml
    #实体类路径
  type-aliases-package: com.sike.entity
    #以下配置均有默认值,可以不设置
  global-config:
    db-config:
    #主键类型 AUTO:"数据库 ID 自增" INPUT:"用户输入 ID",ID_WORKER:"全局唯一 ID (数字类型
    #唯一 ID)", UUID:"全局唯一 ID UUID";
    #配置了这个实体类可以省略注解@TableId
      id-type: AUTO
    #数据库表的前缀,配置后实体类可省略注解@TableName
      table-prefix: tb_
    #字段策略 IGNORED:"忽略判断" NOT_NULL:"非 NULL 判断" NOT_EMPTY:"非空判断"
      field-strategy: NOT_EMPTY
       #数据库类型
      db-type: MYSQL
  configuration:
    #是否开启自动驼峰命名规则映射:从数据库列名到 Java 属性驼峰命名的类似映射
    map-underscore-to-camel-case: true
    #返回 map 时 true:当查询数据为空时字段返回为 null,false:不加这个查询数据为空时,字段
    #将被隐藏
   call-setters-on-nulls: true
    #这个配置会将执行的 SQL 打印出来,在开发或测试时可以用
   log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

(4) 创建数据访问层。创建 BookMapper 接口,继承 BaseMapper<Book>接口,关键代码如下:

```
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.seehope.domain.Book;
public interface BookMapper extends BaseMapper<Book> {
}
```

这里面可以不定义任何方法,因为父接口 BaseMapper 已经包含了基本的增、删、改、查方法,在业务层中调用即可。同样可以选择在这个接口上面添加@Mapper 注解或者在启动类上添加 MapperScan("接口所在包名")注解,这里选择后者。

(5) 创建业务逻辑层。创建 BookService 接口和 BookServiceImpl 实现类,其中 BookServiceImpl 的关键代码如下:

```
//第 5 章/mybatisplus1/BookServiceImpl.java
@Service
public class BookServiceImpl implements BookService {
    @Autowired
    private BookMapper bookMapper;
    @Override
    public List<Book> findAllBooks() {
        return bookMapper.selectList(null);
```

```
    }

    @Override
    public Book findBookById(int id) {
        return bookMapper.selectById(id);
    }

    @Override
    public void addBook(Book book) {
        bookMapper.insert(book);
    }
    @Override
    public void updateBook(Book book) {
        bookMapper.updateById(book);
    }
    @Override
    public void deleteBook(int id) {
        bookMapper.deleteById(id);
    }
}
```

(6) 创建控制器 BookController,代码同 4.1 节案例,具体代码参考本书配套资源。

(7) 创建视图。同 4.1 节案例。

(8) 运行代码进行测试。效果同 4.1 节案例。

总结：MyBatis-Plus 非常省事,数据访问层几乎不用写代码就可以使用。

5.2 条件查询

在 BaseMapper 接口提供的 CRUD 方法中,有些方法提供了 Wrapper 类型的参数,用于设置查询条件,Wrapper 类型的参数既可以使用子类 QueryWrapper 对象,也可以使用子类 LambdaQueryWrapper 对象。

5.2.1 使用 QueryWrapper 封装查询条件

调用 BaseMapper 与 select 有关的方法时,使用 Wrapper 子类的 QueryWrapper 对象做参数,调用 BaseMapper 的 delete 和 update 有关的方法时,使用 Wrapper 子类的 UpdateWrapper 对象做参数。

Wrapper 是一个抽象类,AbstractWrapper 是 Wrapper 的子类,QueryWrapper 和 UpdateWrapper 又是 AbstractWrapper 的子类。AbstractWrapper 提供了众多的方法,用于设置查询条件。常用的方法见表 5-1。

条件查询的步骤,先创建 QueryWrapper 对象或 UpdateWrapper 对象,然后调用表 5-1 中的方法构造查询条件,最后将 QueryWrapper 对象或 UpdateWrapper 对象放入 Dao 层接口的有关方法中作为参数即可。

表 5-1 AbstractWrapper 常用的方法

方 法	语 法	说 明	示 例
eq	eq(R column, Object val) eq(boolean condition, R column, Object val)	等于 =	eq("name", "张飞") 相当于 name = '张飞'
ne	ne(R column, Object val) ne(boolean condition, R column, Object val)	不等于 <>	ne("name", "张飞") 相当于 name<> '张飞'
gt	gt(R column, Object val) gt(boolean condition, R column, Object val)	大于>	gt("age", 18) 相当于 age>18
ge	ge(R column, Object val) ge(boolean condition, R column, Object val)	大于或等于>=	ge("age",18) 相当于 age>=18
lt	lt(R column,Object val) lt(boolean condition,R column,Object val)	小于<	lt("age",18) 相当于 age<18
le	le(R column,Object val) le(boolean condition,R column,Object val)	小于或等于 <=	le("age",18) 相当于 age<=18
between	between(R column,Object val1,Object val2) between(boolean condition,R column,Object val1,Object val2)	BETWEEN 值 1 AND 值 2	between("age",18,30) 相当于 age between 18 and 30
notBetween	notBetween(R column,Object val1,Object val2) notBetween(boolean condition,R column,Object val1,Object val2)	NOT BETWEEN 值 1 AND 值 2	notBetween("age",18,30) 相当于 age not between 18 and 30
like	like(R column,Object val) like(boolean condition,R column,Object val)	LIKE '%值%'	like("name","李") 相当于 name like '%李%'
notLike	notLike(R column,Object val) notLike(boolean condition,R column,Object val)	NOT LIKE '%值%'	notLike("name","李")相当于->name not like '%李%'
likeLeft	likeLeft(R column,Object val) likeLeft(boolean condition,R column,Object val)	LIKE '%值'	likeLeft("name","李") 相当于 name like '%李'
likeRight	likeRight(R column,Object val) likeRight(boolean condition,R column,Object val)	LIKE '值%'	likeRight("name","李") 相当于 name like '李%'

续表

方　法	语　法	说　明	示　例
isNull	isNull(R column) isNull(boolean condition,R column)	字段 IS NULL	isNull("name") 相当于 name is null
isNotNull	isNotNull(R column) isNotNull(boolean condition,R column)	字段 IS NOT NULL	isNotNull("name") 相当于 name is not null
in	in(R column,Collection<? > value) in(boolean condition, R column, Collection<? > value)	字段 IN (value.get(0), value.get(1), …)	in("age",{1,2,3}) 相当于 age in (1,2,3)
	in(R column, Object… values) in(boolean condition, R column, Object … values)	字段 IN (v0, v1, …)	in("age", 1, 2, 3) 相当于 age in (1,2,3)
notIn	notIn(R column, Collection<? > value) notIn(boolean condition, R column, Collection<? > value)	字段 NOT IN (value.get(0), value.get(1),…)	notIn("age",{1,2,3}) 相当于 age not in (1,2,3)
	notIn(R column, Object… values) notIn(boolean condition, R column, Object… values)	字段 NOT IN (v0, v1, …)	notIn("age", 1, 2, 3) 相当于 age not in (1,2,3)
groupBy	groupBy(R… columns) groupBy(boolean condition, R… columns)	分组：GROUP BY 字段, …	groupBy("id", "name") 相当于 group by id,name
orderByAsc	orderByAsc(R… columns) orderByAsc(boolean condition, R … columns)	排序：ORDER BY 字段, … ASC	orderByAsc("id", "name") 相当于 order by id ASC,name ASC
orderByDesc	orderByDesc(R… columns) orderByDesc(boolean condition, R … columns)	排序：ORDER BY 字段, … DESC	orderByDesc("id", "name") 相当于 order by id DESC, name DESC
orderBy	orderBy(boolean condition, boolean isAsc, R… columns)	排序：ORDER BY 字段, …	orderBy(true, true, "id", "name") 相当于 order by id ASC,name ASC
having	having(String sqlHaving,Object… params) having(boolean condition, String sqlHaving, Object… params)	HAVING (SQL 语句)	having("sum(age)>10")相当于 having sum(age)>10 having("sum(age)>{0}",11)相当于 having sum(age)>11
or	or() or(boolean condition)	拼接 OR 注意事项： 主动调用 or 表示紧接着下一种方法不是用 and 连接！(不调用 or 则默认为使用 and 连接)	eq("id",1).or().eq("name", "李白")相当于 id = 1 or name = '李白'

续表

方　法	语　法	说　明	示　例
and	and(Consumer<Param> consumer) and(boolean condition, Consumer<Param> consumer)	AND 嵌套	and(i -> i.eq("name", "张飞").ne("status", "活着"))相当于 and (name = '张飞' and status <> '运动')

【例 5-2】 动态查询图书信息,任意输入不同条件组合均可查询到相关的图书信息。

(1) 接着上一个案例,在项目的 BookService 接口和 BookServiceImpl 中均添加方法 List<Book> searchBooks(Book book),其中 BookServiceImpl 中的方法的代码如下:

```
//第 5 章/mybatisplus1/BookServiceImpl.java
@Override
public List<Book> searchBooks(Book book) {
    QueryWrapper<Book> queryWrapper = new QueryWrapper<>(); queryWrapper.like(book.getName()!= ""
&&book.getName()!= null,"name",book.getName()); queryWrapper.eq(book.getCategory()!= ""&&book.
getCategory()!= null,"category",book.getCategory()); queryWrapper.eq(book.getAuthor()!= ""
&&book.getAuthor()!= null,"author",book.getAuthor());
    return bookMapper.selectList(queryWrapper);
}
```

这里使用了 AbstractMapper 的 like 与 eq 方法,like 方法的第 1 个参数是条件,即当 book 对象的 name 属性不为空时才使用这条查询。上述多个 queryWrapper 语句之间默认为 AND 的关系。

(2) 控制器方法与视图都同第 4 章案例,具体代码可参考本书配套资源,运行代码进行测试,效果也相同。

5.2.2 使用 LambdaQueryWrapper 封装查询条件

上面使用 QueryWrapper 封装查询条件时各个属性是手工输入的字符串,容易出错,而使用 LambdaQueryWrapper 封装查询条件时可以使用 Lambda 表达式,可以调用各个属性,从而避免出现错误。

【例 5-3】 使用 LambdaQueryWrapper 封装上述搜索条件。

(1) 将业务层 BookServiceImpl 中的 searchBooks 原有方法代码注释掉,然后添加代码,添加的代码如下:

```
//第 5 章/mybatisplus1/BookServiceImpl.java
    public List<Book> searchBooks(Book book) {
        LambdaQueryWrapper<Book> queryWrapper = new LambdaQueryWrapper<>(); queryWrapper.
like(book.getName()!= ""&&book.getName()!= null,Book::getName,book.getName());
queryWrapper.eq(book.getCategory()!= ""&&book.getCategory()!= null,Book::getCategory,book.
getCategory());
queryWrapper.eq(book.getAuthor()!= "" &&book.getAuthor()!= null,Book::getAuthor,book.
getAuthor());
        return bookMapper.selectList(queryWrapper);
    }
```

(2) 运行代码进行测试,结果相同。

5.3 分页查询

MyBatis-Plus 提供了 Page 类封装分页信息,Page 包括以下常用分页属性,代码如下:

```
    //当前页的记录集合
  private List<T> records;
    //总记录数
  private long total;
    //每页显示的记录数
  private long size;
    //当前页码
  private long current;
```

Page 类的带参构造方法为 public Page(long current, long size),其中参数 current 表示当前页,size 表示每页显示的记录数。

BaseMapper 有一个 selectPage 方法,完整语法如下:

```
<E extends IPage<T>> E selectPage(E page, @Param("ew") Wrapper<T> queryWrapper);
```

该方法的第 1 个参数是 Page 对象,需用上述提到的构造方法进行构造,第 2 个参数是查询条件,可以为 null,如果是 null 就查询泛型 T 代表的数据库表的所有记录。返回的 IPage 对象即为封装好的分页信息。

此外,还需要做分页配置类。具体见下面的案例。

【例 5-4】 在上面案例项目的基础上分页查询图书信息。

(1) 创建分页配置类。其实就是一个拦截器,代码如下:

```
//第 5 章/mybatisplus1/MybatisPlusConfig.java
import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class MybatisPlusConfig {
    @Bean
    public MybatisPlusInterceptor paginationInterceptor() {
        MybatisPlusInterceptor mybatisPlusInterceptor = new MybatisPlusInterceptor();
        mybatisPlusInterceptor.addInnerInterceptor(new PaginationInnerInterceptor());
        return mybatisPlusInterceptor;
    }
}
```

(2) 向业务层 BookService 接口和 BookServiceImpl 实现类添加 getPage(int pageNum,int size)方法。实现类 BookServiceImpl 中的方法,代码如下:

```
//第 5 章/mybatisplus1/BookServiceImpl.java
   @Override
public IPage<Book> getPage(int pageNum,int size){
            //参数一是当前页,参数二是每页的个数
       IPage<Book> bookPage = new Page<>(pageNum, size);
       bookPage = bookMapper.selectPage(bookPage, null);
       return bookPage;
   }
```

注：Page 类是 IPage 接口的实现类。

(3) 在控制器 BookController 中添加方法,代码如下：

```
//第 5 章/mybatisplus1/BookController.java
   @GetMapping("/booksPage")
    //默认查询第 1 页,每页显示 3 条
   public ModelAndView booksPage(@RequestParam(value = "start",defaultValue = "1") int
start, @RequestParam(value = "size",defaultValue = "3") int size){
    //表示起始页为 start,每页显示 size 条记录,根据 id 升序排序进行分页
       IPage<Book> page = bookService.getPage(start,size);
       ModelAndView mv = new ModelAndView();
       mv.addObject("page",page);
       mv.setViewName("booksPage");
       return mv;
   }
```

(4) 创建视图。booksPage.html 页面关键代码如下。

遍历当前页的商品：

```
//第 5 章/mybatisplus1/booksPage.html
<tr th:each = "book: ${page.records}">
       <td th:text = "${book.id}"></td>
       <td th:text = "${book.name}"></td>
       <td th:text = "${book.pnum}"></td>
       <td th:text = "${book.price}"></td>
       <td th:text = "${book.category}"></td>
       <td th:text = "${book.description}"></td>
       <td th:text = "${book.imgurl}"></td>
       <td th:text = "${book.author}"></td>
       <td th:text = "${book.sales}"></td>
</tr>
```

分页控件：

```
//第 5 章/mybatisplus1/booksPage.html
<div>
    <a th:if = "${page.current > 1}" th:href = "@{/booksPage(start = ${page.current - 1})}">上
一页</a>
      总页数：<span th:text = "${page.getPages()}"></span>  
      当前页：<span th:text = "${page.current}"></span>
      总记录数：<span th:text = "${page.total}"></span>  
```

```
      < a th:if = " ${page.current < page.getPages()}" th:href = "@{/booksPage
(start = ${page.current + 1})}">下一页</a >
</div >
```

(5) 运行代码进行测试,效果同第 4 章案例。

5.4 业务逻辑层快速开发

MyBatis-Plus 不但提供了数据访问层的快速开发,同样针对业务逻辑层也提供了快速开发技术。详情见下面的案例。

【例 5-5】 在上面案例的基础上,使用 MyBatis-Plus 业务层快速开发技术实现查询所有图书和根据 id 查询一本图书的功能。

(1) 创建业务层接口。接口名称为 IBookService,注意要继承 MyBatis-Plus 提供的 IService < Book >接口,代码如下:

```
public interface IBookService extends IService < Book > {
}
```

(2) 创建业务层实现类。实现类的名称为 BookServiceImpl2,注意要实现上述 IBookService 接口,并且要继承 MyBatis-Plus 提供的 ServiceImpl 类,代码如下:

```
@Service
public class BookServiceImpl2 extends ServiceImpl < BookMapper,Book > implements IBookService{
}
```

(3) 在控制器 BookController 中添加方法,代码如下:

```
//第 5 章/mybatisplus1/BookController.java
@Autowired
private IBookService iBookService;
@GetMapping("/books2")
public ModelAndView findAllBooks2(){
    List < Book > books = iBookService.list();
    ModelAndView mv = new ModelAndView();
    mv.addObject("books",books);
    mv.setViewName("books");
    return mv;
}

@GetMapping("/books2/{id}")
public ModelAndView findBooksById2(@PathVariable int id){
    Book book = iBookService.getById(id);
    ModelAndView mv = new ModelAndView();
    mv.addObject("book",book);
    mv.setViewName("book");
    return mv;
}
```

(4) 运行代码进行测试。

通过浏览器中访问 http://localhost:8080/books2 和 http://localhost:8080/books2/1,将分别看到所有图书的列表和 1 号图书的信息。

本章小结

本章学习了 Spring Boot 整合 MyBatis-Plus 基本 CRUD 查询,条件查询,分页查询,以及如何整合业务层进行快速开发。