

第6章 MyBatis及其增强框架

广东财经大学信息学院

罗 东 俊 博士

ZSUJONE@126.COM

(内部资料，请勿外传)



目的和要求

- 掌握**MyBatis**的关联映射机制。
- 掌握**MyBatis**的动态**SQL**语法。
- 掌握**MyBatis-Plus**的常用注解和通用**CRUD**方法。
- 掌握**Spring Boot**整合**MyBatis-Plus**的使用。



主要内容

- 6.1 MyBatis框架运行机制
- 6.2 MyBatis-Plus框架的使用



6.2 MyBatis-Plus框架的使用

- 6.2.1 MyBatis-Plus简介
- 6.2.2 MyBatis-Plus常用注解
- 6.2.3 MyBatis-Plus通用CRUD方法
- 6.2.4 MyBatis-Plus的整合支持
- 6.2.5 MyBatis-Plus代码生成器



6.2.4 MyBatis-Plus的整合支持

- 1.数据准备
- 2.基于映射文件的整合
- 3.基于注解的整合
- 4.基于MP通用CRUD方法的整合
- 5.三种整合方案的对比



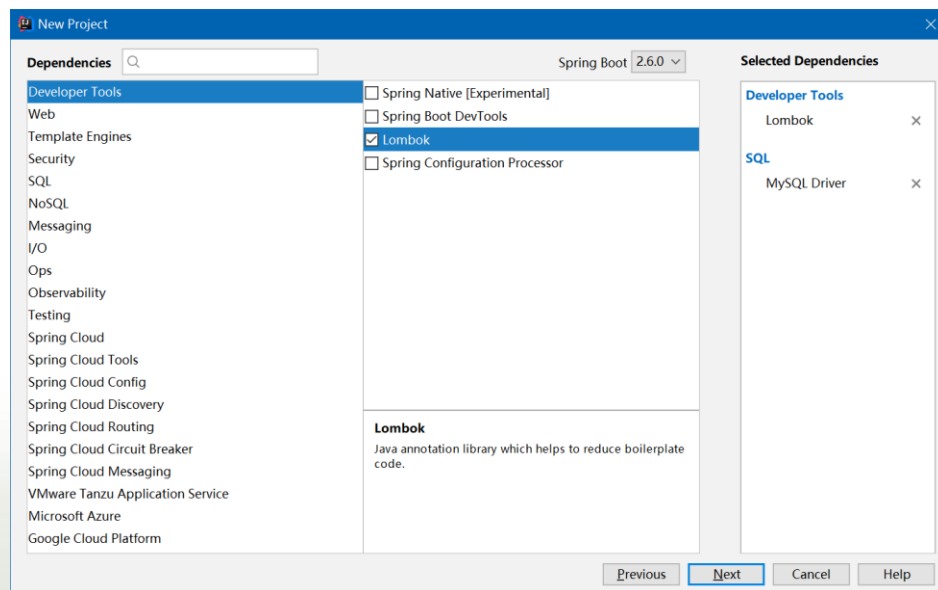
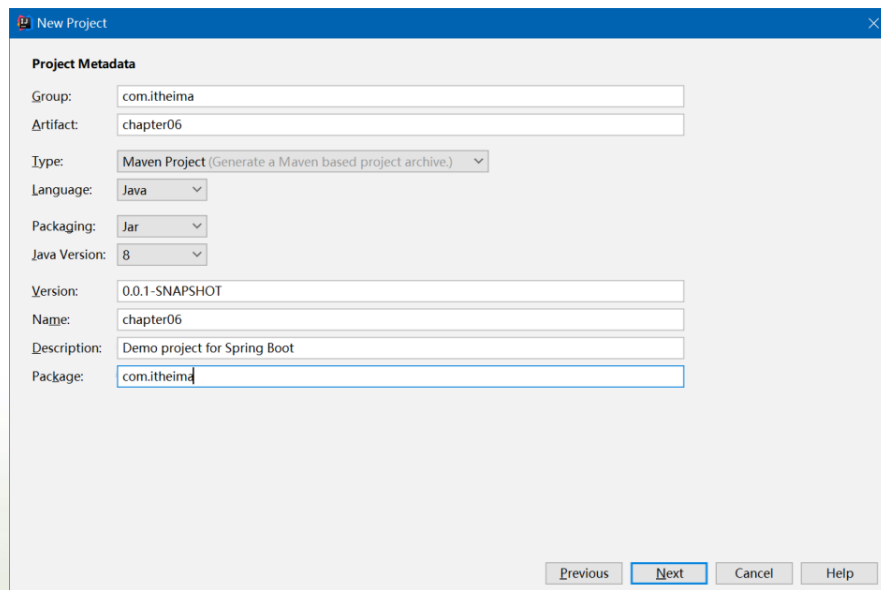
1.数据准备

- ①创建**Spring Boot**项目
- ②建立数据库连接
- ③创建**SQL**文件
- ④运行**SQL**文件
- ⑤查看结果

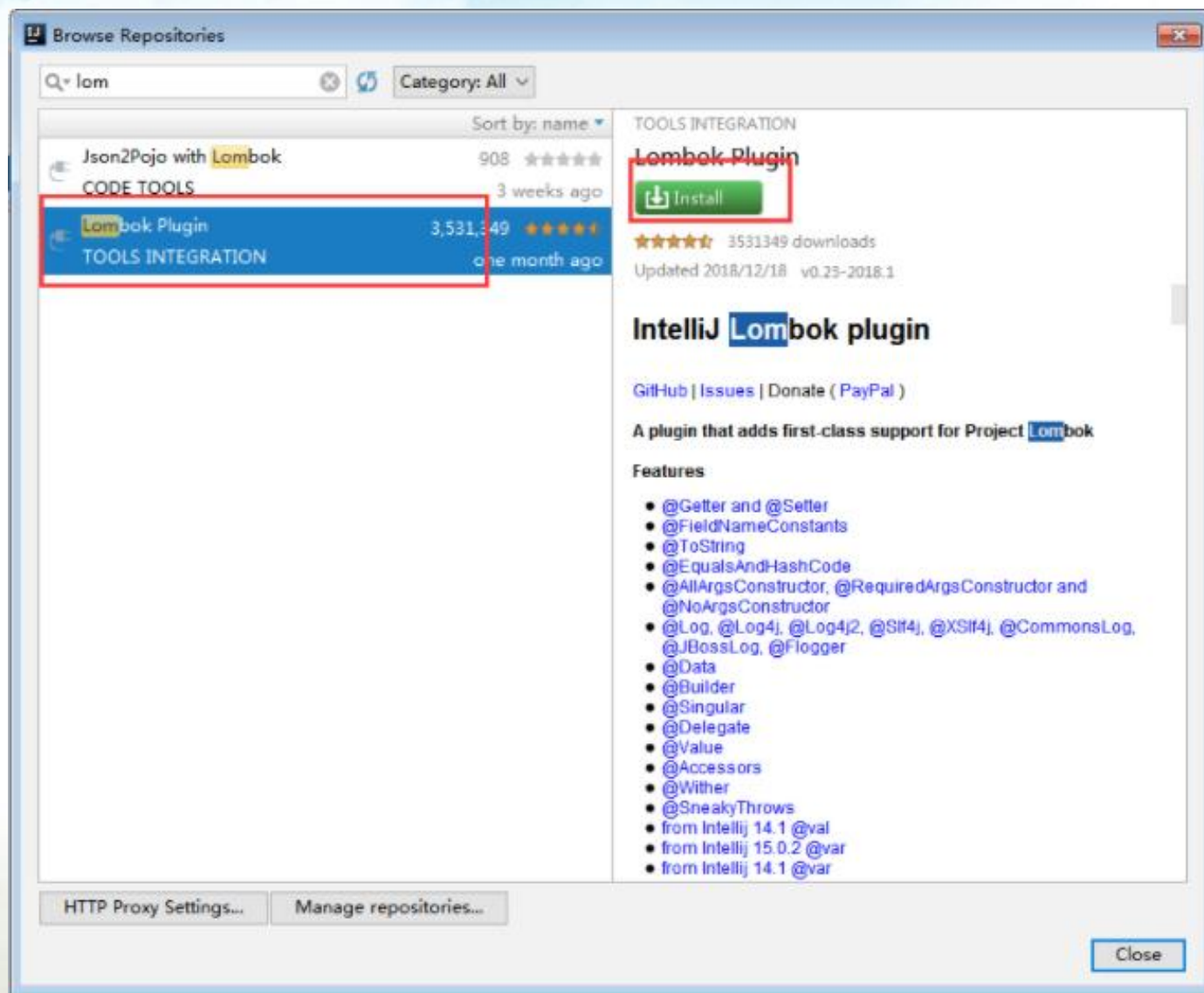


①创建Spring Boot项目

■ 使用Spring Initializr方式创建一个Spring Boot项目chapter06，在Dependencies依赖选择中选择MySQL依赖和Lombok依赖（在IDEA中需事先安装lombok插件才生效）。

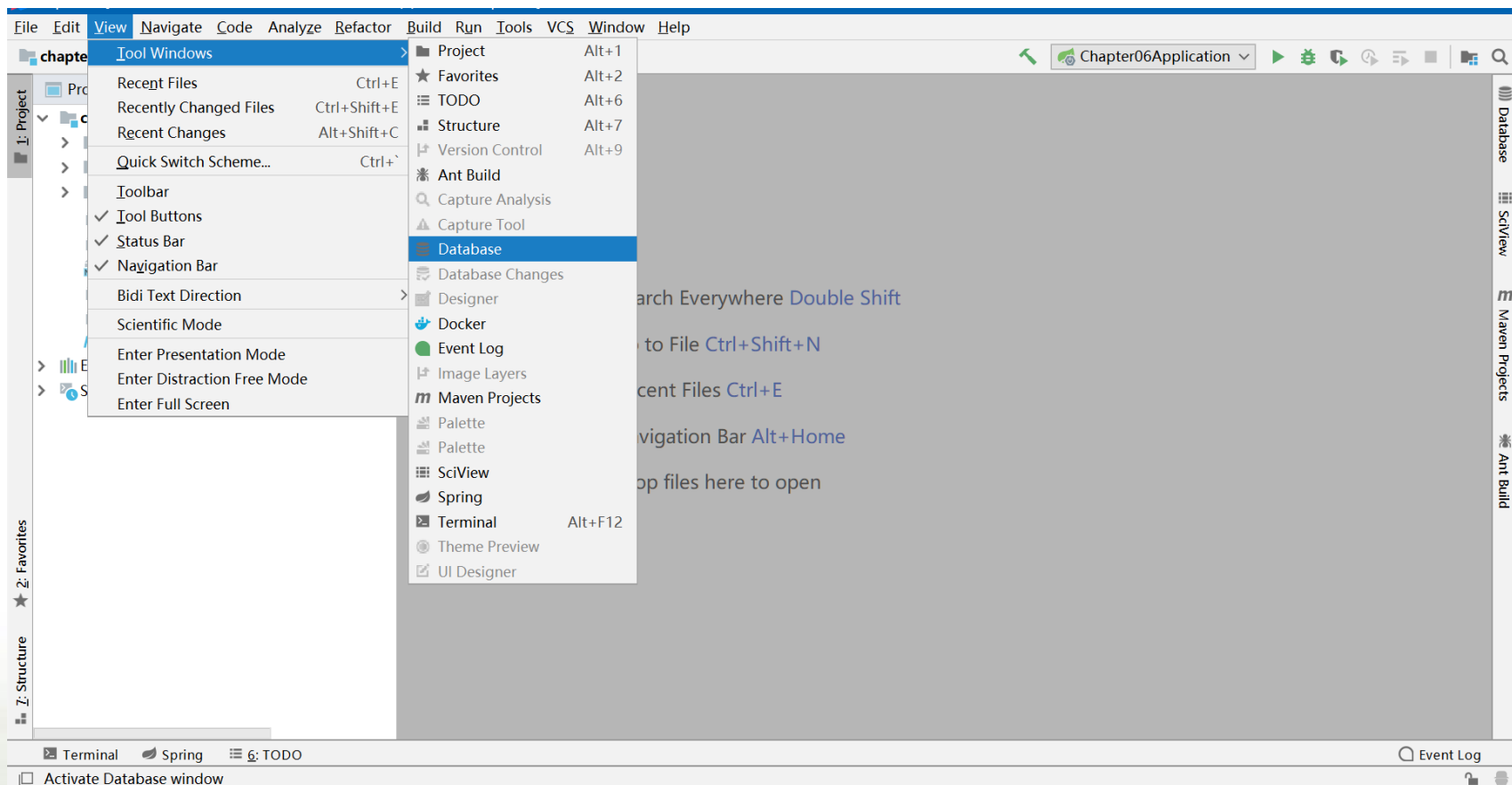


安装lombok插件



②建立数据库连接

■ 在IDEA中建立数据库连接过程如下：



建立数据库连接

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

chapter06

Project

- chapter06 D:\WORKSHOP\软件构造与构件\202
- > .idea
- > .mvn
- > src
 - .gitignore
 - chapter06.iml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
- > External Libraries
- > Scratches and Consoles

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Chapter06Application

Database

Data Source

- DDL Data Source
- Data Source from URL
- Data Source from Path
- Driver and Data Source
- Driver
- Import from Sources...

Create a data source

- Amazon Redshift
- Azure
- ClickHouse
- DB2
- Derby
- Exasol
- H2
- HSQLDB
- MariaDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server
- Sqlite
- Sybase


2: Favorites

7: Structure

Terminal Spring 6: TODO

Event Log

建立数据库连接

 Data Sources and Drivers

Project Data Sources

- springbootconnection

Drivers

- Amazon Redshift
- Azure (Microsoft)
- ClickHouse
- DB2 (JTOpen)
- DB2 (LUW)
- Derby (Embedded)
- Derby (Remote)
- Exasol
- H2
- HSQLDB (Local)
- HSQLDB (Remote)
- MariaDB
- MySQL
- Oracle
- PostgreSQL
- SQL Server (jTds)
- SQL Server (Microsoft)


Name: [Reset](#)

Comment:

General SSH/SSL Schemas Options Advanced

User:

Password: ☒ Remember password


URL: [URL only](#) 

Overrides settings above

[Test Connection](#) Successful [Details](#)

Driver: [MySQL](#)

no objects

Tx: Auto  ☐ Read-only ☒ Auto sync

[OK](#) [Cancel](#) [Apply](#)

③创建SQL文件

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

chapter06 > src > main > resources > sql > springbootdata.sql

Chapter06ApplicationTests.selectUserList

Project

- chapter06 D:\WORKSHOP\软件构造与构
 - .idea
 - .mvn
 - src
 - main
 - java
 - resources
 - mapper
 - sql

springbootdata.sql

application.properties

test

target

.gitignore

chapter06.iml

HELP.md

mvnw

mvnw.cmd

pom.xml

External Libraries

Scratches and Consoles

Search Everywhere Double Shift

Go to File Ctrl+Shift+N

Recent Files Ctrl+E

Navigation Bar Alt+Home

Drop files here to open

Database

springbootconnection 1 of 5

- schemas 1
 - information_schema
- collations 195

4: Run 6: TODO Spring Terminal 0: Messages Database Changes

1 Event Log

Tests passed: 1 (46 minutes ago)

springbootdata.sql

#创建数据库

```
CREATE DATABASE springbootdata;
```

#选择使用数据库

```
USE springbootdata;
```

#创建表t_article并插入相关数据

```
DROP TABLE IF EXISTS t_article;
```

```
CREATE TABLE t_article (
```

```
    id int(20) NOT NULL AUTO_INCREMENT COMMENT '文章id',
```

```
    title varchar(200) DEFAULT NULL COMMENT '文章标题',
```

```
    content longtext COMMENT '文章内容',
```

```
    PRIMARY KEY(id)
```

```
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

```
insert into t_article values('1','Spring Boot基础入门','从入门到精通讲解...');
```

```
insert into t_article values('2','Spring Cloud基础入门','从入门到精通讲解...');
```

springbootdata.sql

#创建表t_comment并插入相关数据

```
drop table if exists t_comment;
```

```
create table t_comment (
```

```
    id int(20) not null auto_increment comment '评论id',
```

```
    content longtext comment '评论内容',
```

```
    u_id int(20) default null comment '评论用户',
```

```
    a_id int(20) default null comment '关联的文章id',
```

```
    primary key(id)
```

```
) engine=InnoDB auto_increment=3 default charset=utf8;
```

```
insert into t_comment values('1','很全、很详细','1','1');
```

```
insert into t_comment values('2','赞一个','2','1');
```

```
insert into t_comment values('3','很详细','4','1');
```

```
insert into t_comment values('4','很好、非常详细','3','1');
```

```
insert into t_comment values('5','很不错','5','2');
```


springbootdata.sql

#创建表t_user并插入相关数据

```
drop table if exists t_user;
```

```
CREATE TABLE t_user (
```

```
    id bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键ID',
```

```
    user_name varchar(20) NOT NULL COMMENT '用户名',
```

```
    password varchar(20) NOT NULL COMMENT '密码',
```

```
    name varchar(30) DEFAULT NULL COMMENT '姓名',
```

```
    age int(10) DEFAULT NULL COMMENT '年龄',
```

```
    email varchar(50) DEFAULT NULL COMMENT '邮箱',
```

```
    PRIMARY KEY (id)
```

```
) ENGINE = InnoDB AUTO_INCREMENT = 1 DEFAULT CHARSET = utf8;
```

```
INSERT INTO t_user VALUES('1', 'zhangsan', '123456', '张三', '18', 'test1@itcast.cn');
```

```
INSERT INTO t_user VALUES('2', 'lisi', '123456', '李四', '20', 'test2@itcast.cn');
```

```
INSERT INTO t_user VALUES('3', 'wangwu', '123456', '王五', '28', 'test3@itcast.cn');
```

```
INSERT INTO t_user VALUES('4', 'zhaoliu', '123456', '赵六', '21', 'test4@itcast.cn');
```

```
INSERT INTO t_user VALUES('5', 'sunqi', '123456', '孙七', '24', 'test5@itcast.cn');
```

springbootdata.sql

```
ALTER TABLE t_user
```

```
ADD COLUMN version int(10) NULL DEFAULT 1 COMMENT '乐观锁版本字段' AFTER email;
```

```
UPDATE t_user SET version='1';
```

```
ALTER TABLE t_user
```

```
ADD COLUMN deleted int(1) NULL DEFAULT 0 COMMENT '1-被删除， 0-未被删除' AFTER  
version;
```

```
UPDATE t_user SET deleted='0';
```

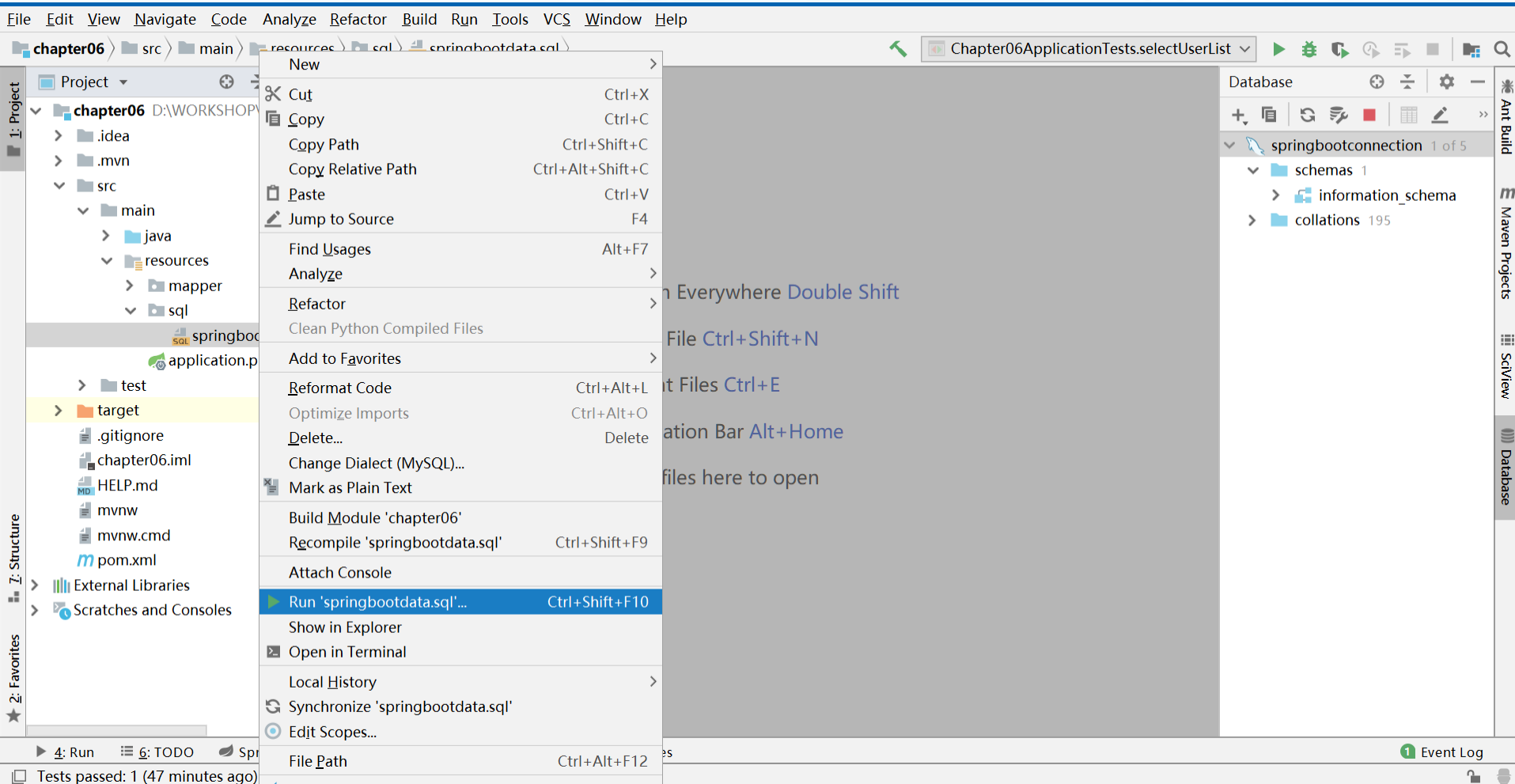
```
ALTER TABLE t_user
```

```
ADD COLUMN sex int(1) NULL DEFAULT 1 COMMENT '1-男， 2-女' AFTER deleted;
```

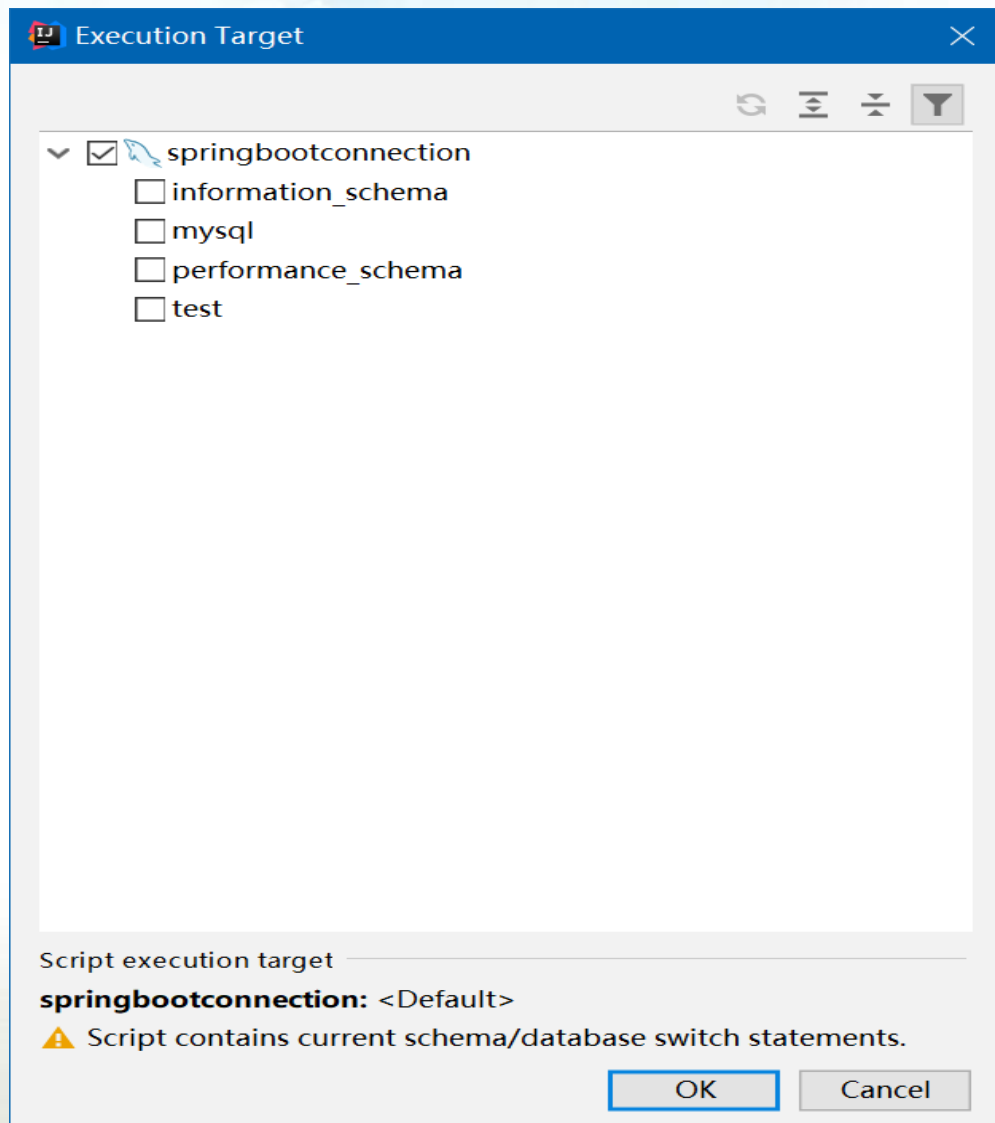
```
UPDATE t_user SET sex='1';
```



④运行SQL文件



运行SQL文件



⑤查看结果

The screenshot displays an IDE interface with the following components:

- Top Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Database View (Right):** Shows a hierarchy of tables under the 't_article' schema:
 - t_article: id int(20) (auto increment), title varchar(200), content longtext, PRIMARY (id)
 - t_authority
 - t_comment: id int(20) (auto increment), content longtext, u_id int(20), a_id int(20), PRIMARY (id)
 - t_user: id bigint(20) (auto increment), user_name varchar(20), password varchar(20), name varchar(30), age int(10), email varchar(50), version int(10), deleted int(1), sex int(1), PRIMARY (id)
 - t_user_authority
- Project View (Left):** Shows the project structure for 'chapter06' (D:\WORKSHOP\软件构造与构):
 - .idea
 - .mvn
 - src
 - main
 - java
 - resources
 - mapper
 - sql
 - springbootdata.sql
 - application.properties
 - test
 - target
 - .gitignore
 - chapter06.iml
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles
- Central Editor:** Displays a search bar with the text 'Search Everywhere Double Shift' and other navigation shortcuts: 'Go to File Ctrl+Shift+N', 'Recent Files Ctrl+E', 'Navigation Bar Alt+Home', and 'Drop files here to open'.
- Bottom Status Bar:** Shows '4: Run', '6: TODO', 'Spring', 'Terminal', 'Q: Messages', 'Database Changes', and 'Event Log'.
- Footer:** A bar at the bottom indicates 'Tests passed: 1 (52 minutes ago)'.

2.基于映射文件整合

- 本案例基于映射文件实现**Spring Boot**与**MyBatis-Plus**的整合，在数据访问层实现向**MySQL**数据库增删改查，在业务处理层实现数据处理。
- 整合步骤：
 - ✧①引入依赖启动器
 - ✧②配置数据源
 - ✧③创建持久化类
 - ✧④创建**DAO**层接口
 - ✧⑤创建业务层类
 - ✧⑥创建映射文件并配置其路径
 - ✧⑦效果测试



①引入依赖启动器

```
<!-- 阿里巴巴的Druid数据源依赖启动器 -->
```

```
<dependency>
```

```
    <groupId>com.alibaba</groupId>
```

```
    <artifactId>druid-spring-boot-starter</artifactId>
```

```
    <version>1.1.10</version>
```

```
</dependency>
```

```
<!--mybatis-plus的springboot支持-->
```

```
<dependency>
```

```
    <groupId>com.baomidou</groupId>
```

```
    <artifactId>mybatis-plus-boot-starter</artifactId>
```

```
    <version>3.1.1</version>
```

```
</dependency>
```

引入依赖启动器

```
<!-- MySQL数据库连接驱动 -->
```

```
<dependency>
```

```
  <groupId>mysql</groupId>
```

```
  <artifactId>mysql-connector-java</artifactId>
```

```
  <version>6.0.6</version>
```

```
  <scope>runtime</scope>
```

```
</dependency>
```

```
<!--简化代码的工具包-->
```

```
<dependency>
```

```
  <groupId>org.projectlombok</groupId>
```

```
  <artifactId>lombok</artifactId>
```

```
  <optional>true</optional>
```

```
</dependency>
```

②配置数据源

- 在项目的全局配置文件 **application.properties** 中设置数据源类型及其他连接配置，示例代码如下。

数据源连接配置

spring.datasource.druid.driver-class-name: com.mysql.cj.jdbc.Driver

spring.datasource.druid.url:

jdbc:mysql://192.168.31.173:3306/springbootdata?serverTimezone=UTC&useSSL=false&useUnicode=true&characterEncoding=utf8

spring.datasource.druid.username: root

spring.datasource.druid.password: root

spring.datasource.druid.initialSize=20

#初始化连接数

spring.datasource.druid.minIdle=10

#最小空闲数

spring.datasource.druid.maxActive=100

#最大连接数

③创建持久化类

- 在chapter06项目中新建一个 **com.itheima.domain**包，在包中创建与数据库表**t_article**和**t_comment**对应的实体类**Article**和**Comment**。



Article类

```
import lombok.Data;
import java.util.List;
@Data
public class Article {
    private Integer id;
    private String title;
    private String content;
    private List<Comment> commentList;
    @Override
    public String toString() {
        return "Article{" + "id=" + id + ", title=" + title + "\" + ", content=" + content + "\" + ",
commentList=" + commentList + '}'';
    }
}
```

Comment类

```
import lombok.Data;
```

```
@Data
```

```
public class Comment {
```

```
    private Integer id;
```

```
    private String content;
```

```
    private Integer uId;
```

```
    private Integer aId;
```

```
    @Override
```

```
    public String toString() {
```

```
        return "Comment{" + "id=" + id + ", content='" + content + '" + ", uId=" + uId + ", aId=" +  
aId + '}';
```

```
    }
```

```
}
```


配置持久化类策略

- 在**application.properties**文件中配置全局的表名前缀和全局id生成策略。

#配置全局的表名前缀

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

#配置全局的id生成策略

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



④创建DAO层接口

- 在chapter06项目中新建一个 **com.itheima.mapper** 包，并在包中创建一个操作数据库表 **t_article** 的接口 **ArticleMapper**，以声明增删改查操作。



ArticleMapper接口

```
import com.itheima.domain.Article;
import org.apache.ibatis.annotations.Mapper;
import java.util.List;

@Mapper
public interface ArticleMapper {
    //查找指定id的文章记录详细信息（包括评论信息）
    public Article findArticleById(Integer id);
    //查找指定ids列表的文章记录详细信息（包括评论信息）
    public List<Article> findArticlesByIds(List<Integer> ids);
    //根据标题模糊查询文章记录列表的详细信息（包括评论信息）
    public List<Article> findArticlesByLikeTitle(String liketitle);
    //添加一条文章记录
    public int insertArticle(Article article);
    //更新一条文章记录
    public int updateArticle(Article article);
    //删除指定id的文章记录详细信息（包括评论信息）
    public int deleteArticle(Integer id);
}
```

关于注解 @Mapper

- **@Mapper**: 标识该类是一个**MyBatis**接口文件，并保证能够被**Spring Boot**自动扫描到**Spring**容器中。
- 也可直接在**Spring Boot**项目启动类上添加**@MapperScan(“com.itheima.mapper”)**以避免逐个添加**@Mapper**的麻烦。



⑤创建业务层类

- 在chapter06项目中新建一个 **com.itheima.service** 包，并在包中新建一个业务类 **ArticleService**，实现对文章数据的处理。



ArticleService类

```
import com.itheima.domain.Article;
import com.itheima.mapper.ArticleMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.ArrayList;
import java.util.List;

@Service
public class ArticleService {
    @Autowired
    private ArticleMapper articleMapper;
```


ArticleService类

//查找指定ids列表的文章记录详细信息（包括评论信息）

```
public void selectArticlesByIds() {  
    List<Integer> ids = new ArrayList<>();  
    ids.add(1);  
    ids.add(2);  
    List<Article> articles = articleMapper.findArticlesByIds(ids);  
    for (Article article:articles) {  
        System.out.println(article);  
    }  
}
```



ArticleService类

//查找指定id的文章记录详细信息（包括评论信息）

```
public void selectArticleById(Integer id) {  
    Article article = articleMapper.findArticleById(id);  
    System.out.println(article);  
}
```

//根据标题模糊查询文章记录列表的详细信息（包括评论信息）

```
public void selectArticlesByLikeTitle(String liketitle) {  
    List<Article> articles = articleMapper.findArticlesByLikeTitle(liketitle);  
    for (Article article:articles) {  
        System.out.println(article);  
    }  
}
```

ArticleService类

//添加一条文章记录

@Transactional

标识该方法被纳入Spring事务管理

```
public void addArticle() {  
    Article article = new Article();  
    article.setTitle("SSM应用开发教程");  
    article.setContent("国家信息技术紧缺人才培养工程指定教材");  
    int rows=articleMapper.insertArticle(article);  
    if(rows>0){  
        System.out.println("您成功插入了"+rows+"条数据！");  
    }else {  
        System.out.println("执行插入操作失败！");  
    }  
}
```

ArticleService类

//修改指定id的文章记录标题和内容

@Transactional

```
public void updateArticle(Integer id) {  
    Article article = articleMapper.findArticleById(id);  
    article.setTitle("Spring Boot企业级开发教程");  
    article.setContent("工业和信息化“十三五”人才培养规划教材");  
    int rows=articleMapper.updateArticle(article);  
    if(rows>0){  
        System.out.println("您成功修改了"+rows+"条数据！");  
    }else {  
        System.out.println("执行修改操作失败！");  
    }  
}
```

ArticleService类

//删除指定id的文章记录详细信息（包括评论信息）

@Transactional

```
public void deleteArticle(Integer id) {  
    int rows=articleMapper.deleteArticle(id);  
    if(rows>0){  
        System.out.println("您成功删除了"+rows+"条数据！");  
    }else {  
        System.out.println("执行删除操作失败！");  
    }  
}  
}
```



⑥创建映射文件并配置其路径

- 在**chapter06**项目的**resources**目录下，新建一个统一管理映射文件的目录**mapper**，并在该目录下创建一个与接口**ArticleMapper**对应且同名的映射文件**ArticleMapper.xml**，在其中实现增删改查操作。



ArticleMapper.xml映射文件

```
<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE mapper  
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<!--namespace为Mapper接口的类路径-->  
<mapper namespace="com.itheima.mapper.ArticleMapper">
```



ArticleMapper.xml映射文件

<!--结果映射集-->

<resultMap id="articleWithComments" type="Article">

 <id property="id" column="id" />

 <result property="title" column="title" />

 <result property="content" column="content" />

 <collection property="commentList" ofType="Comment">

 <id property="id" column="c_id" />

 <result property="content" column="c_content" />

 <result property="uId" column="c_uid" />

 <result property="aId" column="c_aid" />

 </collection>

</resultMap>

ArticleMapper.xml映射文件

```
<!-- 1、 对应 “public Article findArticleById(Integer id)” -->  
<select id="findArticleById" resultMap="articleWithComment">  
    SELECT a.*, c.id c_id, c.content c_content, c.u_id c_uid, c.a_id c_aid  
    FROM t_article a  
        LEFT JOIN t_comment c ON a.id = c.a_id  
    WHERE a.id = #{id}  
</select>
```



ArticleMapper.xml映射文件

```
<!-- 2、 对应 “public List<Article> findArticlesByIds(List<Integer> ids)” -->
<select id="findArticlesByIds" parameterType="List" resultMap="articleWithComment">
    SELECT a.*, c.id c_id, c.content c_content, c.u_id c_uid, c.a_id c_aid
    FROM t_article a
        LEFT JOIN t_comment c ON a.id = c.a_id
    WHERE a.id in
    <foreach item="id" index="index" collection="list" open="(" separator="," close=")">
        #{id}
    </foreach>
</select>
```



ArticleMapper.xml映射文件

```
<!-- 3、 对应 “public List<Article> findArticlesByLikeTitle(String liketitle)” -->
<select id="findArticlesByLikeTitle" resultMap="articleWithComment">
    <bind name="pattern" value="'%' + _parameter + '%'"/>
    SELECT a.*, c.id c_id, c.content c_content, c.u_id c_uid, c.a_id c_aid
    FROM t_article a
        LEFT JOIN t_comment c ON a.id = c.a_id
    WHERE a.title LIKE #{pattern}
</select>
```



ArticleMapper.xml映射文件

```
<!-- 3、 对应 “public List<Article> findArticlesByLikeTitle1(String liketitle)” -->
<select id="findArticlesByLikeTitle1" resultMap="articleWithComment">
    SELECT a.*, c.id c_id, c.content c_content, c.u_id c_uid, c.a_id c_aid
    FROM t_article a
        LEFT JOIN t_comment c ON a.id = c.a_id
    WHERE a.title LIKE concat('%', #{value}, '%')
</select>
```

```
<!-- 4、 对应 “public int insertArticle(Article article)” -->
<insert id="insertArticle" parameterType="Article">
    insert into t_article (title, content)
        values (#{title}, #{content})
</insert>
```

ArticleMapper.xml映射文件

<!-- 5、对应 “public int updateArticle(Article article)” -->

<update id="updateArticle" parameterType="Article">

UPDATE t_article

<set>

<if test="title !=null and title !="">

title=#{title},

</if>

<if test="content !=null and content !="">

content=#{content}

</if>

</set>

WHERE id=#{id}

</update>

ArticleMapper.xml映射文件

<!-- 6、对应 “public int deleteArticle(Integer id)” -->

<delete id="deleteArticle" parameterType="Integer">

DELETE a, c

FROM t_article a

LEFT JOIN t_comment c ON a.id = c.a_id

WHERE a.id = #{id}

</delete>



配置映射文件路径

- 创建映射文件后，**Spring Boot**并无从知晓，所以还须在全局配置文件**application.properties**中添加**MyBatis-Plus**映射文件路径的配置，同时需要添加实体类别名映射路径，示例代码如下。

#配置MyBatis-Plus的映射文件路径

```
mybatis-plus.mapper-locations=classpath:mapper/*.xml
```

#配置映射文件中指定的实体类别名路径

```
mybatis-plus.type-aliases-package=com.itheima.domain
```



⑦效果测试

- 在测试类中引入**ArticleService**类的**Bean**对象，并新增测试方法进行输出测试。



测试方法

@Autowired

```
private ArticleService articleService;
```

@Test

```
public void selectArticlesByIds() { this.articleService.selectArticlesByIds();}
```

@Test

```
public void addArticle() { this.articleService.addArticle();}
```

@Test

```
public void selectArticleById() { this.articleService.selectArticleById(3);}
```

@Test

```
public void selectArticlesByLikeTitle() {this.articleService.selectArticlesByLikeTitle("S");}
```

@Test

```
public void updateArticle() {this.articleService.updateArticle(3);}
```

@Test

```
public void deleteArticle() {this.articleService.deleteArticle(3);}
```

测试结果

Tests passed: 1 of 1 test – 417 ms

Test Results 417 ms

- Chapter06ApplicationTests 417 ms
 - addArticle() 417 ms

Registering transaction synchronization for SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@3d904e9c] JDBC Connection [com.alibaba.druid.proxy.jdbc.ConnectionProxyImpl@5fa23c] will be managed by Spring
==> Preparing: insert into t_article (title, content) values (?, ?)
==> Parameters: SSM应用开发教程(String), 国家信息技术紧缺人才培养工程指定教材(String)
<== Updates: 1
Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@3d904e9c]
您成功插入了1条数据!
Transaction synchronization committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@3d904e9c]
Transaction synchronization deregistering SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@3d904e9c]

Database Changes Terminal Spring 0: Messages 4: Run 6: TODO Event Log

Tests passed: 1 of 1 test – 420 ms

Test Results 420 ms

- Chapter06ApplicationTests 420 ms
 - selectArticleById() 420 ms

==> Preparing: SELECT a.*, c.id c_id, c.content c_content, c.author, c.a_id c_aid FROM t_article a LEFT JOIN t_comment c ON a.id = c.a_id
==> Parameters: 3(Integer)
<== Columns: id, title, content, c_id, c_content, author, c_aid
<== Row: 3, SSM应用开发教程, <<BLOB>>, null, <<BLOB>>, null, null
<== Total: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@71904469]
Article{id=3, title='SSM应用开发教程', content='国家信息技术紧缺人才培养工程指定教材', commentList=[]}
2021-11-21 21:17:37.686 INFO 14728 --- [ionShutdownHook] com.alibaba.druid.pool.DruidDataSource : {d

Tests passed: 1 of 1 test – 428 ms

Test Results 428 ms

- Chapter06ApplicationTests 428 ms
 - updateArticle() 428 ms

Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@210308d5]
Fetched SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@210308d5] from current transaction
==> Preparing: UPDATE t_article SET title=?, content=? WHERE id=?
==> Parameters: Spring Boot企业级开发教程(String), 工业和信息化“十三五”人才培养规划教材(String), 3(Integer)
<== Updates: 1
Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@210308d5]
您成功修改了1条数据!
Transaction synchronization committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@210308d5]
Transaction synchronization deregistering SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@210308d5]

Tests passed: 1 of 1 test – 425 ms

Test Results 425 ms

- Chapter06ApplicationTests 425 ms
 - selectArticleById() 425 ms

==> Preparing: SELECT a.*, c.id c_id, c.content c_content, c.author, c.a_id c_aid FROM t_article a LEFT JOIN t_comment c ON a.id = c.a_id
==> Parameters: 3(Integer)
<== Columns: id, title, content, c_id, c_content, author, c_aid
<== Row: 3, Spring Boot企业级开发教程, <<BLOB>>, null, <<BLOB>>, null, null
<== Total: 1
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@71904469]
Article{id=3, title='Spring Boot企业级开发教程', content='工业和信息化“十三五”人才培养规划教材', commentList=[]}
2021-11-21 21:24:12.471 INFO 14188 --- [ionShutdownHook] com.alibaba.druid.pool.DruidDataSource : {d

3.基于注解的整合

■ 本案例在上述案例数据源基础上基于注解实现**Spring Boot**与**MyBatis-Plus**的整合，在**DAO**层实现向**MySQL**数据库增删改查，在业务处理层实现数据处理。

■ 整合步骤：

- ✧①创建**DAO**层接口
- ✧②创建业务层类
- ✧③效果测试



①创建DAO层接口

- 在chapter06项目的com.itheima.mapper包中创建一个操作数据库表t_comment的接口CommentMapper，在接口内部分别通过@Select、@Insert、@Update、@Delete注解配合SQL语句完成对数据库表数据的增删改查操作。
- 注意：在Mapper接口类中不要定义同名的方法，尽管方法参数或返回类型不完全一样，但被调用时只要定义在前的方法匹配成功，定义在后的同名方法将会失效。

CommentMapper接口

```
import com.itheima.domain.Comment;
```

```
import org.apache.ibatis.annotations.*;
```

```
@Mapper
```

```
public interface CommentMapper {
```

```
    @Select("SELECT * FROM t_comment WHERE id =#{id}")
```

```
    public Comment findCommentById(Integer id);
```

```
    @Select("SELECT * FROM t_comment WHERE u_id =#{id}")
```

```
    public List<Comment> findCommentsByUserId(Integer id);
```

```
    @Insert("INSERT INTO t_comment(content,u_id,a_id) " + "values (#{content},#{uId},#{aId})")
```

```
    public int insertComment(Comment comment);
```

```
    @Update("UPDATE t_comment SET content=#{content} WHERE id=#{id}")
```

```
    public int updateComment(Comment comment);
```

```
    @Delete("DELETE FROM t_comment WHERE id=#{id}")
```

```
    public int deleteComment(Integer id);
```

```
}
```


②创建业务层类

- 在chapter06项目的com.itheima.service包中新建一个业务类CommentService，实现对评论数据的处理。



CommentService类

```
import com.itheima.domain.Comment;
import com.itheima.mapper.CommentMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service

public class CommentService {

    @Autowired

    private CommentMapper commentMapper;

    //查找指定id的记录详细信息

    public void selectComment(Integer id) {

        Comment comment = commentMapper.findCommentById(id);

        System.out.println(comment);

    }
```

CommentService类

//添加一条记录

@Transactional

标识该方法被纳入Spring事务管理

```
public void addComment() {  
    Comment comment = new Comment();  
    comment.setContent("这篇文章值得收藏");  
    comment.setUid(3);  
    comment.setAId(2);  
    int rows=commentMapper.insertComment(comment);  
    if(rows>0){  
        System.out.println("您成功插入了"+rows+"条数据！");  
    }else {  
        System.out.println("执行插入操作失败！");  
    }  
}
```

CommentService类

//修改指定id的记录字段值

@Transactional

```
public void updateComment(Integer id) {  
    Comment comment = commentMapper.findCommentById(id);  
    comment.setContent("这篇文章太精彩了");  
    int rows=commentMapper.updateComment(comment);  
    if(rows>0){  
        System.out.println("您成功修改了"+rows+"条数据！");  
    }else {  
        System.out.println("执行修改操作失败！");  
    }  
}
```

CommentService类

//删除指定id的记录

@Transactional

```
public void deleteComment(Integer id) {  
    int rows=commentMapper.deleteComment(id);  
    if(rows>0){  
        System.out.println("您成功删除了"+rows+"条数据！");  
    }else {  
        System.out.println("执行删除操作失败！");  
    }  
}  
}
```

③效果测试

- 在测试类中引入**CommentService**类的**Bean**对象，并新增测试方法进行输出测试。



测试方法

@Autowired

```
private CommentService commentService;
```

@Test

```
public void addComment() {this.commentService.addComment();}
```

@Test

```
public void selectComment() {this.commentService.selectComment(6);}
```

@Test

```
public void updateComment() {this.commentService.updateComment(6);}
```

@Test

```
public void deleteComment() {this.commentService.deleteComment(6); }
```



测试结果

Tests passed: 1 of 1 test – 474 ms

Test Results 474 ms

- Chapter06Application 474 ms
 - addComment() 474 ms

(content,u_id,a_id)
values
('这篇文章值得收藏',3,2)

Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@5d43409a] 您成功插入了1
Transaction synchronization committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@5d434

Tests passed: 1 of 1 test – 533 ms

Test Results 533 ms

- Chapter06Application 533 ms
 - selectComment() 533 ms

Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@277bf091]
Comment{id=6, content='这篇文章值得收藏', uId='3', aId=2}
2021-12-05 21:32:14.458 INFO 7636 --- [ionShutdownHook] com.alibaba.druid.pool.DruidDataSource : {dataSou

Process finished with exit code 0

Tests passed: 1 of 1 test – 429 ms

Test Results 429 ms

- Chapter06Application 429 ms
 - updateComment() 429 ms

parser sql: UPDATE t_comment SET content = ? WHERE id = ?
==> Preparing: UPDATE t_comment SET content=? WHERE id=?
==> Parameters: 这篇文章太精彩了(String), 6(Integer)
<== Updates: 1
Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@22a736d7]
您成功修改了1条数据!

Tests passed: 1 of 1 test – 335 ms

Test Results 335 ms

- Chapter06Application 335 ms
 - deleteComment() 335 ms

WHERE
id=6

<== Updates: 1
Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@7add323c]
您成功删除了1条数据!

4.基于MP通用CRUD方法的整合

- 本案例在上述案例数据源基础上基于AR化实体类的通用CRUD方法实现Spring Boot与MyBatis-Plus的整合，**间接**在DAO层实现向MySQL数据库增删改查，而在业务处理层实现数据处理。
- 整合步骤：
 - ✧①创建AR化实体类
 - ✧②创建DAO层接口
 - ✧③创建业务层类
 - ✧④创建MP配置类
 - ✧⑤创建自动填充处理器
 - ✧⑥效果测试



①创建AR化实体类

- 在chapter06项目的com.itheima.domain包中创建与数据库表t_user对应的实体类User，并对User类进行AR化。
- 在chapter06项目中新建一个com.itheima.enums包，并在包中创建枚举类SexEnum，作为User枚举属性sex的类型。



User类

```
import com.baomidou.mybatisplus.annotation.*;
import com.baomidou.mybatisplus.extension.activerecord.Model;
import com.itheima.enums SexEnum;
import lombok.Data;
import java.util.List;

@Data

public class User extends Model<User> {
    private Integer id;
    private String userName;
    //查询时不返回该字段的值. 插入数据时进行填充
    @TableField(select = false, fill = FieldFill.INSERT)
    private String password;
```



User类

```
private String name;  
private Integer age;  
@TableField(value = "email")    //指定数据表中字段名  
private String mail;  
@Version                        //乐观锁的版本字段  
private Integer version;  
@TableLogic                      //逻辑删除字段，1-删除，0-未删除  
private Integer deleted;  
private SexEnum sex;             //性别，枚举类型  
@TableField(exist = false)      //数据库表中不存在相应字段  
private List<Comment> commentList;  
}
```

SexEnum枚举类

```
import com.baomidou.mybatisplus.core.enums.IEnum;

public enum SexEnum implements IEnum<Integer> {

    MAN(1,"男"),
    WOMAN(2,"女");

    private Integer value;
    private String desc;

    SexEnum(Integer value, String desc) {
        this.value = value;
        this.desc = desc;
    }

    @Override
    public Integer getValue() { return this.value;}

    @Override
    public String toString() { return this.desc; }

}
```


配置枚举类包路径

- 为了让Spring Boot能扫描到枚举类，在 **application.properties** 文件中配置枚举类的包路径。

#配置枚举包扫描

mybatis-plus.type-enums-package=com.itheima.enums



②创建DAO层接口

- 在chapter06项目的com.itheima.mapper包中创建一个操作数据库表t_user的接口UserMapper，通过继承BaseMapper完成对数据库表数据的增删改查操作。



UserMapper接口

```
import com.baomidou.mybatisplus.core.mapper.BaseMapper;  
import com.itheima.domain.User;  
import org.apache.ibatis.annotations.Mapper;
```

@Mapper

```
public interface UserMapper extends BaseMapper<User> {  
}
```



③创建业务层类

- 在chapter06项目的com.itheima.service包中新建一个业务类UserService，实现对用户数据的处理，包括条件查询、分页查询、多表查询、添加记录、更新记录、基于乐观锁修改、全表更新、逻辑删除等。



UserService类

```
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import com.baomidou.mybatisplus.extension.plugins.pagination.Page;
import com.itheima.domain.Comment;
import com.itheima.domain.User;
import com.itheima.enums SexEnum;
import com.itheima.mapper.CommentMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.util.List;

@Service
public class UserService {
```

UserService类

//根据实体条件查询所有用户记录

```
public void selectUserList() {  
    User user = new User();  
    QueryWrapper<User> queryWrapper = new QueryWrapper<>();  
    queryWrapper.ge("age","20");  
    user.selectList(queryWrapper).forEach(System.out::println);  
}
```



UserService类

//根据分页设置和实体条件查询用户记录

```
public void selectUserPage() {  
    User user = new User();  
    QueryWrapper<User> wrapper = new QueryWrapper<>();  
    wrapper.gt("age", 20); //年龄大于20岁  
    Page<User> page = new Page<>(1,2); //设置当前页号和分页大小  
    IPage<User> iPage = user.selectPage(page, wrapper);  
    System.out.println("数据总条数: " + iPage.getTotal());  
    System.out.println("总页数: " + iPage.getPages());  
    List<User> users = iPage.getRecords(); //获取当前页中的记录  
    for (User u : users) {  
        System.out.println("user = " + u);  
    }  
}
```

UserService类

@Autowired

CommentMapper commentMapper;

//多表查询：查找指定id的用户记录详细信息（包括评论信息）

```
public void selectUserWithCommentsById(Integer id) {
```

```
    User user = new User();
```

```
    User u = user.selectById(id);
```

```
    List<Comment> list = commentMapper.findCommentsByUserId(id);
```

```
    u.setCommentList(list);
```

```
    System.out.println(u);
```

```
}
```



UserService类

//添加一条用户记录

@Transactional

public void addUser() {

 User user = new User();

 user.setName("刘备");

 user.setAge(30);

 user.setUserName("liubei");

 user.setMail("liubei@itcast.cn");

 user.setSex(SexEnum.MAN);

 boolean result = user.insert();

 if(result){ System.out.println("您成功插入数据！");}

 else { System.out.println("执行插入操作失败！");}

}

UserService类

//修改指定id的用户记录年龄

@Transactional

public void updateUser(Integer id) {

 User user = new User();

 user.setId(id);

 user.setAge(35);

 boolean result = **user.updateById()**;

 if(result){

 System.out.println("您成功修改了数据！");

 }else {

 System.out.println("执行修改操作失败！");

 }

}

UserService类

//不需事务处理，基于乐观锁修改指定id的用户记录年龄

```
public void updateUserWithOptimisticLocker(Integer id) {
```

```
    User user = new User();
```

```
    User u1 = user.selectById(id);
```

```
    User u2 = user.selectById(id);
```

```
    u1.setAge(35);
```

```
    u2.setAge(25);
```

```
    boolean result2 = u2.updateById();
```

```
    boolean result1 = u1.updateById();
```

```
    if(result2 && !result1){
```

```
        System.out.println("您成功修改了2号数据，但执行1号修改操作失败！");
```

```
    }
```

```
}
```

UserService类

//在开启逻辑删除下进行全表更新或删除（不受配置的SQL执行分析插件拦截）

@Transactional

```
public void updateAllUser() {
```

```
    User user = new User();
```

```
    //user.delete(null);
```

```
    user.setAge(50);
```

```
    user.update(null);
```

```
}
```



UserService类

//逻辑删除指定id的用户记录详细信息

@Transactional

public void deleteUser(Integer id) {

 User user = new User();

 user.setId(id);

 boolean result = **user.deleteById()**;

 if(result){

 System.out.println("您成功删除了数据！");

 }else {

 System.out.println("执行删除操作失败！");

 }

}

}

④创建MP配置类

- 为了实现分页查询、基于乐观锁修改、SQL性能分析以及防止在未开启逻辑删除时执行全表更新、删除等误操作，需要创建MP配置类。
- 在chapter06项目中新建一个com.itheima.config包，并在包中创建配置类MybatisPlusConfig。



MybatisPlusConfig配置类

```
import com.baomidou.mybatisplus.core.parser.ISqlParser;
import com.baomidou.mybatisplus.extension.parsers.BlockAttackSqlParser;
import com.baomidou.mybatisplus.extension.plugins.PaginationInterceptor;
import com.baomidou.mybatisplus.extension.plugins.SqlExplainInterceptor;
import com.baomidou.mybatisplus.extension.plugins.PerformanceInterceptor;
import com.baomidou.mybatisplus.extension.plugins.OptimisticLockerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import java.util.ArrayList;
import java.util.List;

@Configuration
public class MybatisPlusConfig {
```


MybatisPlusConfig配置类

@Bean //配置分页插件

```
public PaginationInterceptor paginationInterceptor(){  
    return new PaginationInterceptor();  
}
```

@Bean //配置乐观锁插件

```
public OptimisticLockerInterceptor optimisticLockerInterceptor() {  
    return new OptimisticLockerInterceptor();  
}
```



MybatisPlusConfig配置类

@Bean //配置SQL性能分析插件，该插件只用于开发环境，不建议生产环境使用。

```
public PerformanceInterceptor performanceInterceptor(){  
    PerformanceInterceptor performanceInterceptor = new PerformanceInterceptor();  
    //设置SQL是否格式化  
    performanceInterceptor.setFormat(true);  
    //设置SQL最大执行时间（ms），超过时间会抛出异常。  
    performanceInterceptor.setMaxTime(100);  
    return performanceInterceptor;  
}
```



MybatisPlusConfig配置类

@Bean //配置SQL执行分析插件，该插件只用于开发环境，不建议生产环境使用。

```
public SqlExplainInterceptor sqlExplainInterceptor(){
```

```
    SqlExplainInterceptor sqlExplainInterceptor = new SqlExplainInterceptor();
```

```
    List<ISqlParser> list = new ArrayList<>();
```

//添加全表更新、删除的阻断器，在未开启逻辑删除时，当执行全表更新、删除会抛出异常，有效防止一些误操作。

```
    list.add(new BlockAttackSqlParser());
```

```
    sqlExplainInterceptor.setSqlParserList(list);
```

```
    return sqlExplainInterceptor;
```

```
}
```

```
}
```

关于MyBatis-Plus的SQL分析插件

■ **SqlExplainInterceptor**: SQL语句执行分析插件，用于对SQL语句的执行进行分析拦截，防止一些误操作的发生，如全表更新、全表删除等。

✧ 当开启逻辑删除（使用注解@**TableLogic**）时，该插件将失效

■ **PerformanceInterceptor**: SQL语句性能分析插件，用于测量一条SQL语句执行的时间

■ 以上插件只用于开发环境，不建议生产环境使用。

⑤创建自动填充处理器

- 在chapter06项目中新建一个 `com.itheima.handler` 包，并在包中创建处理器类 `MyMetaObjectHandler`，以实现 `password` 字段的自动填充。



MyMetaObjectHandler类

```
import com.baomidou.mybatisplus.core.handlers.MetaObjectHandler;
import org.apache.ibatis.reflection.MetaObject;
import org.springframework.stereotype.Component;

@Component
public class MyMetaObjectHandler implements MetaObjectHandler {

    @Override
    public void insertFill(MetaObject metaObject) {
        this.setFieldValByName("password","123456",metaObject);
    }

    @Override
    public void updateFill(MetaObject metaObject) {
    }

}
```


⑥效果测试

- 在测试类中引入UserService类的Bean对象，并新增测试方法进行输出测试。



测试方法

@Autowired

```
private UserService userService;
```

@Test

```
public void selectUserList(){ this.userService.selectUserList(); }
```

@Test

```
public void selectUserPage(){ this.userService.selectUserPage(); }
```

@Test

```
public void selectUserWithCommentsById(){ this.userService.selectUserWithCommentsById(5); }
```



测试方法

@Test

```
public void addUser() { this.userService.addUser(); }
```

@Test

```
public void updateUser() { this.userService.updateUser(6); }
```

@Test

```
public void updateUserWithOptimisticLocker() {this.userService.updateUserWithOptimisticLocker(6); }
```

@Test

```
public void updateAllUser() { this.userService.updateAllUser(); }
```

@Test

```
public void deleteUser() { this.userService.deleteUser(6); }
```



测试结果

Tests passed: 1 of 1 test – 515 ms

Test Results 515 ms

- Chapter06Application 515 ms
 - selectUserPage() 515 ms

Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@48d293ee]
数据总条数: 3
总页数: 2
user = User(id=3, userName=wangwu, password=null, name=王五, age=28, mail=test3@itcast.cn, version=1, deleted=0)
user = User(id=4, userName=zhaoliu, password=null, name=赵六, age=21, mail=test4@itcast.cn, version=1, deleted=0)
2021-12-06 16:12:08.141 INFO 6720 --- [ionShutdownHook] com.alibaba.druid.pool.DruidDataSource : {dataSo

Tests passed: 1 of 1 test – 407 ms

Test Results 407 ms

- Chapter06Application 407 ms
 - selectUserWithCon 407 ms

:SqlSession@277b8fa4]
1, version=1, deleted=0, sex=男, commentList=[Comment{id=5, content='很不错', uId='5', aId=2}]) Time: 16 ms -

Tests passed: 1 of 1 test – 489 ms

Test Results 489 ms

- Chapter06Application 489 ms
 - addUser() 489 ms

(user_name, password, name, age, email, sex)
VALUES
('liubei', '123456', '刘备', 30, 'liubei@itcast.cn', 1)
Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@48d293ee]
您成功插入数据!
Transaction synchronization committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@48d293ee]
Transaction synchronization deregistering SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@48d293ee]

Tests passed: 1 of 1 test – 439 ms

Test Results 439 ms

- Chapter06Application 439 ms
 - updateUserWithOj 439 ms

WHERE
id=6
AND version=1
AND deleted=0
==> Parameters: liubei(String), 刘备(String), 35(Integer), liubei@itcast.cn(String), 2(Integer), 1(Integer),
<== Updates: 0
Closing non transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@748d2277]
您成功修改了2号数据, 但执行1号修改操作失败!

5.三种整合方案的对比

- 基于映射文件的整合方案将**SQL**的操作语句及映射关系集中放在映射文件中方便统一管理，也可充分利用动态**SQL**特性提高访问数据库的灵活性。
- 基于注解的整合方案将**SQL**的操作语句和映射关系都基于注解放在**Mapper**接口中，使接口变得臃肿，特别是涉及多表字段关联查询情况。
- 基于**MP**通用**CRUD**方法的整合方案不需程序员构造**SQL**操作语句，对于单表操作非常便利，提高了开发效率，但对多表关联的操作会带来嵌套查询加载问题。
- 结论：对于多表字段复杂关联查询一般采用基于映射文件的整合方案。



6.2.5 MyBatis-Plus代码生成器

- **AutoGenerator**是**MyBatis-Plus**的代码生成器，通过**AutoGenerator**可以根据数据库表快速自动生成**Entity**、**Mapper**、**Mapper XML**、**Service**、**Controller**等各个模块的代码，极大的提升了开发效率。



相关依赖

<!--代码生成器-->

<dependency>

<groupId>com.baomidou</groupId>

<artifactId>mybatis-plus-generator</artifactId>

<version>3.1.1</version>

</dependency>

<!--模板引擎依赖-->

<dependency>

<groupId>org.apache.velocity</groupId>

<artifactId>velocity-engine-core</artifactId>

<version>2.2</version>

</dependency>



相关依赖

```
<dependency>
```

```
  <groupId>com.baomidou</groupId>
```

```
  <artifactId>mybatis-plus-boot-starter</artifactId>
```

```
  <version>3.1.1</version>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>mysql</groupId>
```

```
  <artifactId>mysql-connector-java</artifactId>
```

```
  <scope>runtime</scope>
```

```
</dependency>
```

```
<dependency>
```

```
  <groupId>org.projectlombok</groupId>
```

```
  <artifactId>lombok</artifactId>
```

```
  <optional>true</optional>
```

```
</dependency>
```


本章小结

■ 本章具体讲解了：

✧ 6.1 MyBatis框架运行机制

- MyBatis简介
- MyBatis的关联映射
- MyBatis的动态SQL
- MyBatis的延迟加载机制
- MyBatis的缓存机制

✧ 6.2 MyBatis-Plus框架的使用

- MyBatis-Plus简介
- MyBatis-Plus常用注解
- MyBatis-Plus通用CRUD方法
- MyBatis-Plus的整合支持
- MyBatis-Plus代码生成器



