MyBatis

一、简介

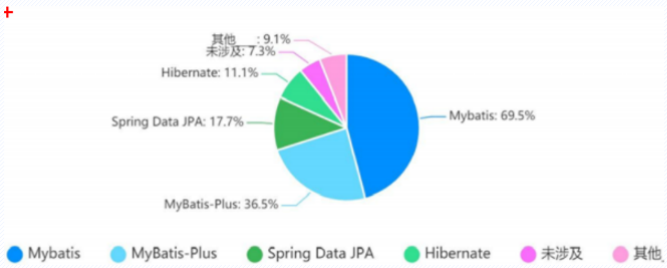
概念：MyBatis 是一款优秀的==持久层框架[DAO]==，用于简化 JDBC 开发

1.MyBatis**怎么念?**

2.MyBatis 本是 Apache 的一个开源项目iBatis，2013年11月迁移到Github。

3.官网：https://mybatis.org/mybatis-3

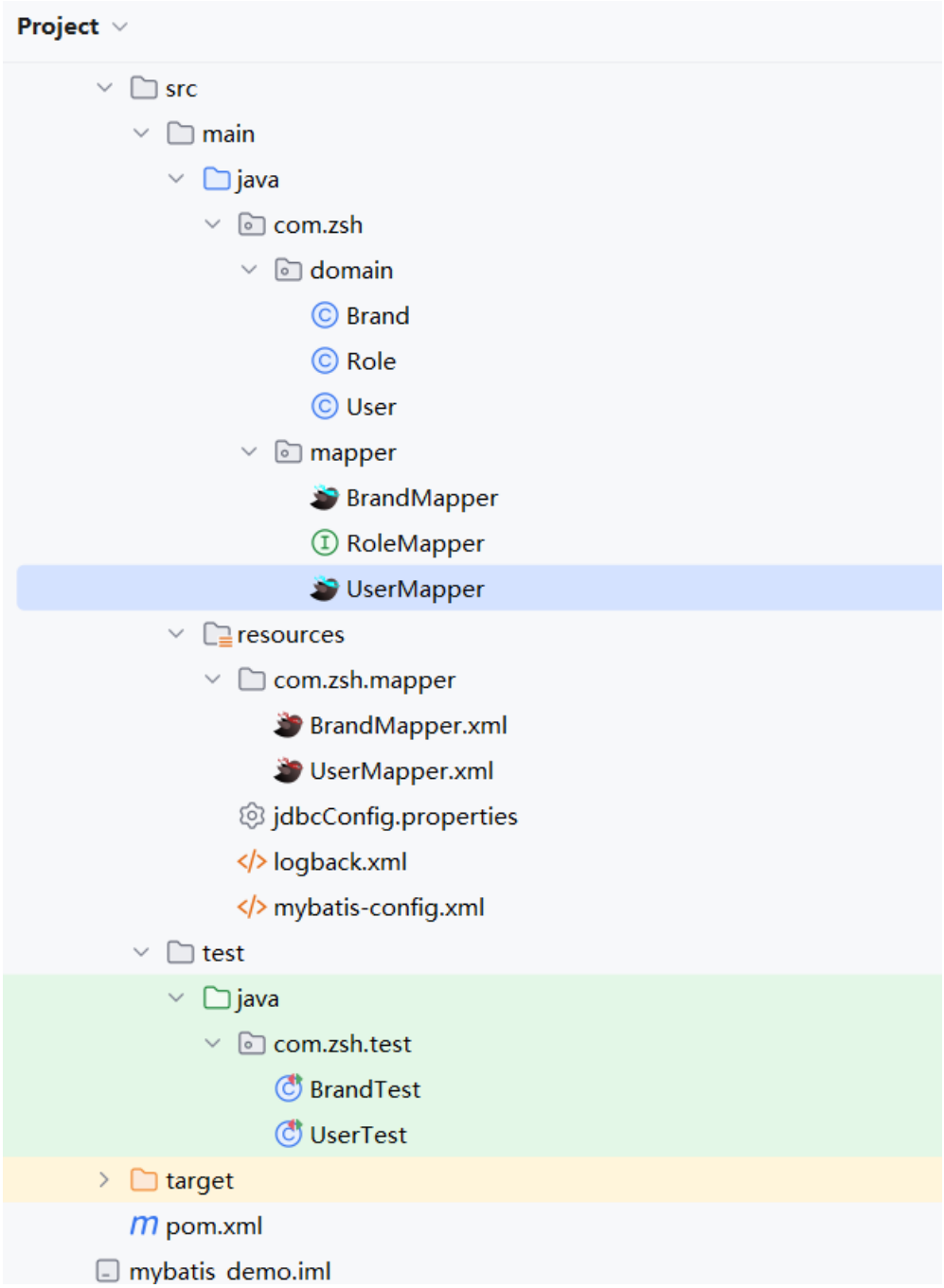
MyBatis **是目前JAVA最主流持久层框架**



MyBatis-Plus

二、MyBatis入门案例：查询所有用户信息-`selectAll`

0.项目整体结构



1.导入`mybatis.sql`文件，初始化`mybatis`数据库

```
CREATE DATABASE IF NOT EXISTS `mybatis` DEFAULT CHARACTER SET utf8 COLLATE
utf8_bin

USE `mybatis`;

/*Table structure for table `account` */
DROP TABLE IF EXISTS `account`;
CREATE TABLE `account` (
  `id` int(11) NOT NULL COMMENT '编号',
  `uid` int(11) DEFAULT NULL COMMENT '用户编号',
  `money` double DEFAULT NULL COMMENT '金额',
  PRIMARY KEY (`id`),
  KEY `FK_Reference_8` (`uid`),
  CONSTRAINT `FK_Reference_8` FOREIGN KEY (`uid`) REFERENCES `user` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```sql
/*Data for the table `account` */
insert  into `account`(`id`,`uid`,`money`) values (1,46,1000),(2,45,1000),
(3,46,2000);

/*Table structure for table `brand` */
DROP TABLE IF EXISTS `brand`;
CREATE TABLE `brand` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `brand_name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `company_name` varchar(20) COLLATE utf8_bin DEFAULT NULL,
  `ordered` int(11) DEFAULT NULL,
  `description` varchar(100) COLLATE utf8_bin DEFAULT NULL,
  `status` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

/*Data for the table `brand` */
insert  into
`brand`(`id`,`brand_name`,`company_name`,`ordered`,`description`,`status`)
values
(1,'格力','格力科技有限公司',1,'格力改变世界',0),
(2,'华为','华为技术有限公司',2,'华为科技世界之巅',1),
(3,'美的','美的科技有限公司',3,'美的改变世界',1);

/*Table structure for table `role` */
DROP TABLE IF EXISTS `role`;
CREATE TABLE `role` (
  `id` int(11) NOT NULL COMMENT '编号',
  `rolename` varchar(30) DEFAULT NULL COMMENT '角色名称',
  `roledesc` varchar(60) DEFAULT NULL COMMENT '角色描述',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `role` */
insert  into `role`(`id`,`rolename`,`roledesc`) values
(1,'仓库管理员','管理整个公司仓库模块'),
(2,'财务管理员','管理整个公司财务模块');

/*Table structure for table `user` */
DROP TABLE IF EXISTS `user`;
CREATE TABLE `user` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `username` varchar(20) NOT NULL COMMENT '用户名称',
  `password` varchar(20) DEFAULT NULL COMMENT '生日',
  `gender` char(1) DEFAULT NULL COMMENT '性别',
  `address` varchar(100) DEFAULT NULL COMMENT '地址',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=52 DEFAULT CHARSET=utf8;

/*Data for the table `user` */
insert  into `user`(`id`,`username`,`password`,`gender`,`address`) values
(41,'李四','123456','男','北京'),
(42,'小王','1234567','女','浙江金华'),
(43,'大王','1234568','女','浙江杭州'),
(45,'王五','1234569','男','山东廊坊'),
```

```
(46,'老赵','1234560','男','北京朝阳区'),
(48,'小李','123456','女','福建福州'),
(49,'小强','123456','男','浙江东阳');

/*Table structure for table `user_role` */
DROP TABLE IF EXISTS `user_role`;
CREATE TABLE `user_role` (
  `uid` int(11) NOT NULL COMMENT '用户编号',
  `rid` int(11) NOT NULL COMMENT '角色编号',
  PRIMARY KEY (`uid`,`rid`),
  KEY `FK_Reference_10` (`rid`),
  CONSTRAINT `FK_Reference_10` FOREIGN KEY (`rid`) REFERENCES `role` (`id`),
  CONSTRAINT `FK_Reference_9` FOREIGN KEY (`uid`) REFERENCES `user` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

/*Data for the table `user_role` */
insert  into `user_role`(`uid`,`rid`) values (41,1),(45,1),(41,2);
```

2.创建空工程`MyBatisProject`,创建模块`mybatis`，在`pom.xml`中导入各种需要的依赖坐标，在项目的`resources`目录下复制`logback.xml`这一配置文件

- `pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.zsh</groupId>
    <artifactId>mybatis</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <!-- mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.5.5</version>
        </dependency>
        <!-- mysql 驱动 -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.30</version>
        </dependency>
        <!-- junit 驱动 -->
        <dependency>
```

```
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
            <scope>test</scope>
        </dependency>
        <!-- 添加slf4j日志api -->
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.20</version>
        </dependency>
        <!-- 添加logback-classic依赖 -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>1.2.3</version>
        </dependency>
        <!-- 添加logback-core依赖 -->
        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-core</artifactId>
            <version>1.2.3</version>
        </dependency>
    </dependencies>
</project>
```

- `logback.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <!-- CONSOLE ：表示当前的日志信息是可以输出到控制台的。 -->
    <appender name="Console" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>
                [%level] %blue(%d{HH:mm:ss.SSS}) %cyan([%thread])
%boldGreen(%logger{15}) - %msg %n
            </pattern>
        </encoder>
    </appender>

    <logger name="com.itbbj" level="DEBUG" additivity="false">
        <appender-ref ref="Console"/>
    </logger>

    <!--
      level:用来设置打印级别，大小写无关：TRACE, DEBUG, INFO, WARN, ERROR, ALL 和
OFF，默认debug
      <root>可以包含零个或多个<appender-ref>元素，标识这个输出位置将会被本日志级别控制。
    -->
    <root level="DEBUG">
        <appender-ref ref="Console"/>
    </root>
</configuration>
```

3.创建实体层`com.zsh.domain`，创建实体类`User`，参考数据库字段编写代码

```
package com.zsh.domain;

public class User {
    private Integer id;
    private String username;
    private String password;
    private String gender;
    private String address;

    ...
}
```

4.创建数据库映射层`com.zsh.mapper`,创建接口`UserMapper`,编写`selectAll`方法

```
package com.zsh.mapper;

import com.zsh.domain.User;
import java.util.List;

public interface UserMapper {
    List<User> selectAll();
}
```

5.在模块的`resources-com.zsh.mapper`目录下创建映射配置文件`UserMapper.xml`

```
<?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">
<mapper namespace="com.zsh.mapper.UserMapper">
    <select id="selectAll" resultType="User">
        select * from user
    </select>
</mapper>
```

6.在模块的`resources`目录下创建`mybatis`的配置文件`mybatis-config.xml`，从而建立与数据库之间的连接（**注册驱动**）

```
<?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="development">
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.cj.jdbc.Driver"/>
                <property name="url" value="jdbc:mysql://localhost:3306/mybatis?
characterEncoding=utf8"/>
                <property name="username" value="root"/>
                <property name="password" value="123456"/>
            </dataSource>
        </environment>
    </environments>
    <mappers>
        <mapper resource="com/zsh/mapper/UserMapper.xml"/>
    </mappers>
</configuration>
```

7.在模块的 `test-java` 目录下创建 `com.zsh.test` 层，创建 `UserTest` 类，编码

```
package com.zsh.test;

import com.zsh.domain.User;
import com.zsh.mapper.UserMapper;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.Test;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class UserTest {
    @Test
    public void testSelectAll() throws IOException {
        //1.读取配置文件--在resources文件夹下可以直接读到
        InputStream inputStream = Resources.getResourceAsStream("mybatis-
config.xml");
        //2.创建SqlSessionFactory工厂
        SqlSessionFactory sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        //3.使用工厂生产SqlSession对象
        SqlSession sqlSession=sqlSessionFactory.openSession();
        //4.使用SqlSession创建Dao接口的代理对象
        UserMapper userMapper=sqlSession.getMapper(UserMapper.class);
        //5.使用代理对象执行方法
        List<User> users=userMapper.selectAll();
        System.out.println(users);
        //6.释放资源
        sqlSession.close();
        inputStream.close();
```

```
    }
}
```

三、根据Id查询用户信息-`selectById`

1.重构`UserMapper`接口，加入`selectById`方法。

```
package com.zsh.mapper;

import com.zsh.domain.User;
import java.util.List;

public interface UserMapper {
    List<User> selectAll();

    User selectById(Integer id);
}
```

2.重构`UserMapper.xml`，加入`selectById`节点

```
<?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">
<mapper namespace="com.zsh.mapper.UserMapper">
    <select id="selectAll" resultType="com.zsh.domain.User">
        select * from user
    </select>

    <select id="selectById" resultType="com.zsh.domain.User">
        select * from user where id=#{id}
    </select>
</mapper>
```

3.重构`UserTest.java`（使用了`@Before`、`@After`注解来优化测试类代码，大大减少了重复累赘的代码）

```
package com.zsh.test;

import com.zsh.domain.User;
import com.zsh.mapper.UserMapper;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
```

```java
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class UserTest {
    private InputStream inputStream;
    private SqlSessionFactory sqlSessionFactory;
    private SqlSession sqlSession;
    private UserMapper userMapper;

    @Before
    public void init() throws IOException {
        System.out.println("Before============================");
        //1.读取配置文件--在resources文件夹下可以直接读到
        inputStream = Resources.getResourceAsStream("mybatis-config.xml");
        //2.创建SqlSessionFactory工厂
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        //3.使用工厂生产SqlSession对象
        sqlSession=sqlSessionFactory.openSession();
        //4.使用SqlSession创建Dao接口的代理对象
        userMapper=sqlSession.getMapper(UserMapper.class);
    }

    @After
    public void destroy() throws IOException {
        System.out.println("After============================");
        //释放资源
        sqlSession.close();
        inputStream.close();
    }

    @Test
    public void testSelectAll() throws IOException {
        //使用代理对象执行方法
        List<User> users=userMapper.selectAll();
        System.out.println(users);
    }

    @Test
    public void testSelectById() throws IOException {
        //使用代理对象执行方法
        System.out.println(userMapper.selectById(46));
    }
}
```

四、核心配置文件

MyBatis 的配置文件包含了会深深影响 MyBatis 行为的设置和属性信息。 配置文档的顶层结构如下：

1-properties（属性）

- 创建 `jdbcConfig.properties` 文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/mybatis?characterEncoding=utf8
jdbc.username=root
jdbc.password=123456
```

- 重构主配置文件 `mybatis-config.xml`（**注意连接数据库的4个基本信息要修改**）

```
<configuration>
    <!-- 配置属性 -->
    <properties resource="jdbcConfig.properties"></properties>

    <!-- 配置环境 -->
    <environments default="mysql">
        <!-- 配置mysql的环境-->
        <environment id="mysql">
            <!-- 配置事务的类型-->
            <transactionManager type="JDBC"></transactionManager>
            <!-- 配置数据源（连接池） -->
            <dataSource type="POOLED">
                <!-- 配置连接数据库的4个基本信息 -->
                <property name="driver" value="${jdbc.driver}"></property>
                <property name="url" value="${jdbc.url}"></property>
                <property name="username" value="${jdbc.username}"></property>
                <property name="password" value="${jdbc.password}"></property>
            </dataSource>
        </environment>
    </environments>
    ...
```

2-typeAliases（类型别名）

注意节点的位置顺序不能写错（properties（属性） ->settings（设置） ->typeAliases（类型别名） ）

方式一：配置在主配置文件`configuration`节点之下

```
<!--使用typeAliases配置别名，它只能配置实体类的别名 -->
<typeAliases>
<!--
    typeAlias用于配置别名，指定一个对象实体。type属性指定的是实体类全限定类名。alias属性指定别名，当指定了别名就不再区分大小写
-->
    <typeAlias type="com.zsh.domain.User" alias="User"></typeAlias>
</typeAliases>
```

方式二：扫包配置（推荐使用）

```
<typeAliases>
    <package name="com.zsh.domain"/>
</typeAliases>
```

3-environments（环境配置）

在核心配置文件的`environments`标签中其实是可以配置多个`environment`，使用`id`给每段环境起名，在`environments`中使用`default='环境id'`来指定使用哪个环境的配置。我们一般就配置一个`environment`即可

参考代码

```
    <!--配置环境-->
    <environments default="development">
        <!--配置开发环境-->
        <environment id="development">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>

        <!--配置测试环境-->
        <environment id="test">
            <transactionManager type="JDBC"/>
            <dataSource type="POOLED">
                <property name="driver" value="${jdbc.driver}"/>
                <property name="url" value="${jdbc.url}"/>
                <property name="username" value="${jdbc.username}"/>
                <property name="password" value="${jdbc.password}"/>
            </dataSource>
        </environment>
    </environments>
```

4-mappers（映射器）

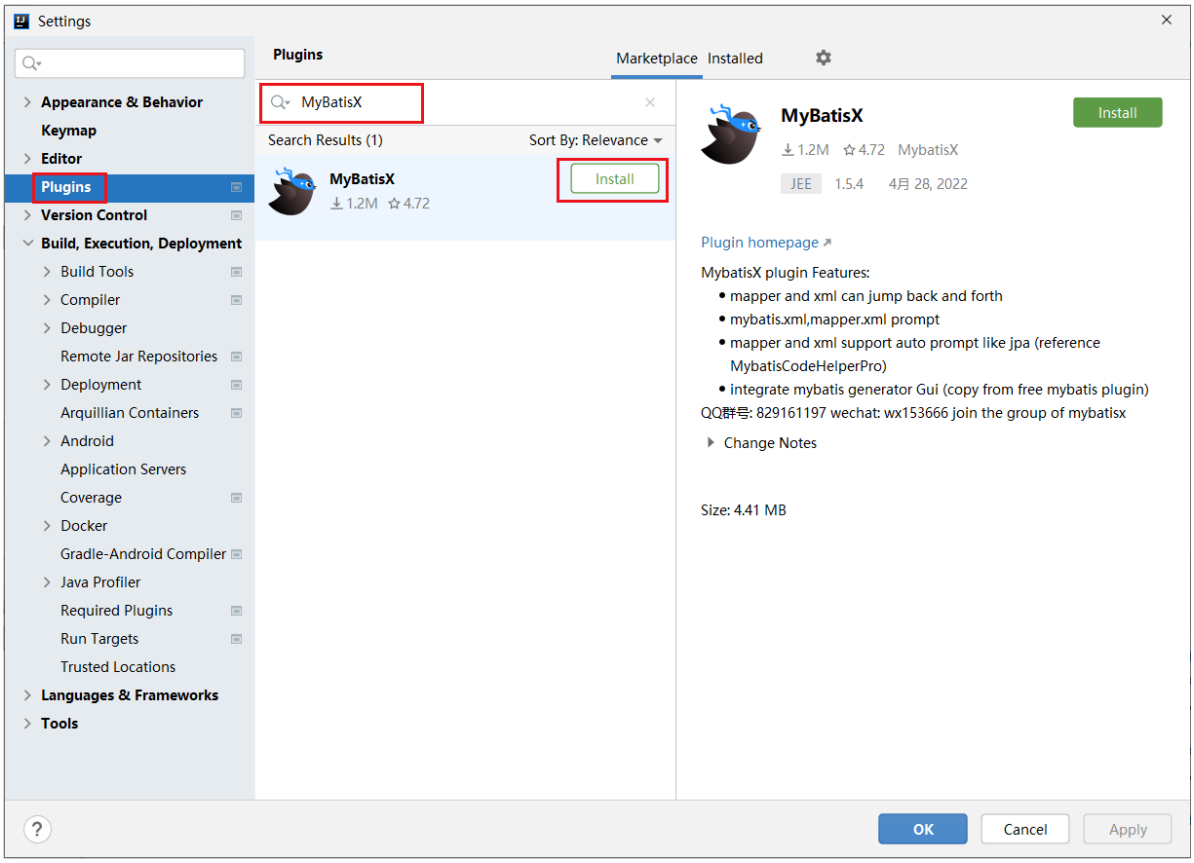
配置`<package>`，那么就会自动包含`mapper`包下的所有`xml`，无需再一一配置映射器。

参考代码

```
<!--配置映射器-->
    <mappers>
    <!--
        <mapper resource="com/zsh/mapper/UserMapper.xml"/>
    -->
        <package name="com.zsh.mapper"/>
    </mappers>
```

五、开发利器

1.MyBatisX



2.lombok

加入依赖坐标

```
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.20</version>
        </dependency>
```

实例类

```
package com.itbbj.domain;

import lombok.Data;

@Data//自动生成get&set方法
@NoArgsConstructor//自动生成无参构造器
@AllArgsConstructor//自动生成包含全部参数的有参构造器
public class User {
    private Integer id;
    private String username;
    private String password;
    private String gender;
    private String address;
}
```

六、CRUD操作

A.查询

A-1 `ResultMap`应用

A-1-1 案例需求：查询brand表中所有数据

	id	brand_name	company_name	ordered	description	status
1	1	格力	格力科技有限公司	1	格力改变世界	0
2	2	华为	华为技术有限公司	2	华为科技世界之巅	1
3	3	美的	美的科技有限公司	3	美的改变世界	1

A-1-2 实现步骤：

Step1.在实体层`com.zsh.domain`中创建实体类`Brand`,参考数据库中`brand`表的相应字段编码。

注意事项：留意属性名称和数据库字段是否一致

思考会造成什么现象？**会造成查询出来的数据中，`brand_name`和`company_name`均为`null`**

```java
@Data
public class Brand {
    private Integer id;
    private String brandName;
    private String companyName;
    private Integer ordered;
    private String description;
    //1-启用状态 0-禁用状态
    private Integer status;
}
```

Step2.在数据库映射层`com.zsh.mapper`中创建接口`BrandMapper`,编码。

```java
package com.zsh.mapper;

import com.zsh.domain.Brand;
import java.util.List;

public interface BrandMapper {
    List<Brand> selectAll();
}
```

Step3.在`resources`目录下创建映射配置文件`BrandMapper.xml`

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <select id="selectAll" resultType="Brand">
        select * from brand;
    </select>
</mapper>
```

Step4.在`test-java-com.zsh.test`目录下创建`BrandTest`类，编码`testSeleteAll()`。

```java
package com.zsh.test;

import com.zsh.domain.Brand;
import com.zsh.mapper.BrandMapper;
import org.apache.ibatis.io.Resources;
import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

public class BrandTest {
    private InputStream inputStream;
```

```
    private SqlSessionFactory sqlSessionFactory;
    private SqlSession sqlSession;
    private BrandMapper brandMapper;

    @Before
    public void init() throws IOException {
        System.out.println("Before============================");
        //1.读取配置文件--在resources文件夹下可以直接读到
        inputStream = Resources.getResourceAsStream("mybatis-config.xml");
        //2.创建SqlSessionFactory工厂
        sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
        //3.使用工厂生产SqlSession对象
        sqlSession=sqlSessionFactory.openSession();
        //4.使用SqlSession创建Dao接口的代理对象
        brandMapper=sqlSession.getMapper(BrandMapper.class);
    }

    @After
    public void destroy() throws IOException {
        System.out.println("After============================");
        //释放资源
        sqlSession.close();
        inputStream.close();
    }

    @Test
    public void testSelectAll() throws IOException {
        //使用代理对象执行方法
        List<Brand> brands=brandMapper.selectAll();
        for (Brand brand : brands) {
            System.out.println(brand);
        }
    }
}
```

运行结果截图如下：

```
[DEBUG] 00:10:40.496 [main] c.z.m.B.selectAll - <==      Total: 3
Brand(id=1, brandName=null, companyName=null, ordered=1, description=格力改变世界, status=0)
Brand(id=2, brandName=null, companyName=null, ordered=2, description=华为科技世界之巅, status=1)
Brand(id=3, brandName=null, companyName=null, ordered=3, description=美的改变世界, status=1)
```

A-1-3 对于`brand_name、company_name`显示为`null`的解决方案：

重构`BrandMapper.xml`

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
```

```
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>
    <select id="selectAll" resultMap="brandMap">
        select * from brand;
    </select>
</mapper>
```

可以简化，只写命名不同的属性

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
    </resultMap>
    <select id="selectAll" resultMap="brandMap">
        select * from brand;
    </select>
</mapper>
```

运行结果截图如下：

```
[DEBUG] 23:50:43.856 [main] c.z.m.B.selectAll - <==      Total: 3
Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)
Brand(id=2, brandName=华为, companyName=华为技术有限公司, ordered=2, description=华为科技世界之巅, status=1)
Brand(id=3, brandName=美的, companyName=美的科技有限公司, ordered=3, description=美的改变世界, status=1)
```

A-2 模糊查询

A-2-1 案例需求：根据公司名称进行全匹配模糊查询，查询公司名称中包含"科技有限"的数据信息

WHERE　ORDER BY

	id	brand_name	company_name	ordered	description	status
1	1	格力	格力科技有限公司	1	格力改变世界	0
2	2	华为	华为技术有限公司	2	华为科技世界之巅	1
3	3	美的	美的科技有限公司	3	美的改变世界	1

A-2-2 实现方式

方式一（不推荐）：`brandMapper.selectByCompanyName("%科技有限%")`

步骤1：在`BrandMapper`层创建`selectByCompanyName`方法

```java
package com.zsh.mapper;
import com.zsh.domain.Brand;
import java.util.List;

public interface BrandMapper {
    List<Brand> selectAll();

    List<Brand> selectByCompanyName(String companyName);
}
```

步骤2：重构`BrandMapper.xml`//#占位符

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCompanyName" resultMap="brandMap">
        select * from brand where company_name like #{companyName};
    </select>
</mapper>
```

步骤3：创建测试方法`testSeleteByCompanyName`进行测试

```java
    @Test
    public void testSelectByCompanyName() throws IOException{
        List<Brand> brands=brandMapper.selectByCompanyName("%科技有限%");
        for (Brand brand : brands) {
            System.out.println(brand);
        }
    }
```

运行结果截图如下：

```
[DEBUG] 00:33:41.303 [main] c.z.m.B.selectByCompanyName - ==>  Preparing: select * from brand where company_name like ?;
[DEBUG] 00:33:41.328 [main] c.z.m.B.selectByCompanyName - ==> Parameters: %科技有限%(String)
[DEBUG] 00:33:41.355 [main] c.z.m.B.selectByCompanyName - <==      Total: 2
Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)
Brand(id=3, brandName=美的, companyName=美的科技有限公司, ordered=3, description=美的改变世界, status=1)
```

方式二（不推荐）：`like '%${companyName}%'`

步骤1：在`BrandMapper`层创建`selectByCompanyName`方法：同方式一步骤1

步骤2：重构`BrandMapper.xml`//字符串拼接

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCompanyName" resultMap="brandMap">
        select * from brand where company_name like '%${companyName}%';
    </select>
</mapper>
```

步骤3：创建测试方法`testSeleteByCompanyName`进行测试

```java
    @Test
    public void testSelectByCompanyName() throws IOException{
        List<Brand> brands=brandMapper.selectByCompanyName("科技有限");//无需%号
        for (Brand brand : brands) {
            System.out.println(brand);
        }
    }
```

运行结果截图如下：

```
[DEBUG] 00:42:18.630 [main] c.z.m.B.selectByCompanyName - ==>  Preparing: select * from brand where company_name like '%科技有限
[DEBUG] 00:42:18.655 [main] c.z.m.B.selectByCompanyName - ==> Parameters:
[DEBUG] 00:42:18.687 [main] c.z.m.B.selectByCompanyName - <==      Total: 2
Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)
Brand(id=3, brandName=美的, companyName=美的科技有限公司, ordered=3, description=美的改变世界, status=1)
```

方式三（推荐）：`concat('%',#{companyName},'%')`

步骤1：在`BrandMapper`层创建`selectByCompanyName`方法：同方式一步骤1

步骤2：重构`BrandMapper.xml`

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
```

```
    </resultMap>

    <select id="selectByCompanyName" resultMap="brandMap">
        select * from brand where company_name like concat('%',#
{companyName},'%');
    </select>
</mapper>
```

步骤3：创建测试方法`testSeleteByCompanyName`进行测试：同方式二步骤3

运行结果截图如下：

```
[DEBUG] 00:45:54.345 [main] c.z.m.B.selectByCompanyName - ==>  Preparing: select * from brand where company_name like concat('%',?,'%');
[DEBUG] 00:45:54.369 [main] c.z.m.B.selectByCompanyName - ==> Parameters: 科技有限(String)
[DEBUG] 00:45:54.396 [main] c.z.m.B.selectByCompanyName - <==      Total: 2
Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)
Brand(id=3, brandName=美的, companyName=美的科技有限公司, ordered=3, description=美的改变世界, status=1)
```

A-2-3 总结

```
方式一：封装数据时耦合业务需求（不推荐）
brandMapper.selectByCompanyName("%科技有限%");
方式二：like '%${companyName}%';（不推荐）
'%${companyName}%'表示字符串连接，相当于'%'+XXX+'%',但是会产生SQL注入，有安全漏洞
方式三：like  concat('%',#{companyName},'%')-推荐
```

A-3 多参数查询

A-3-1 案例需求：根据公司名称及品牌状态进行全匹配模糊查询，查询公司名称中包含"科技有限"及状态为'0'的数据信息

	id	brand_name	company_name	ordered	description	status
1	1	格力	格力科技有限公司	1	格力改变世界	0
2	2	华为	华为技术有限公司	2	华为科技世界之巅	1
3	3	美的	美的科技有限公司	3	美的改变世界	1

A-3-2 实现方法

方法一：使用`@param`注解

步骤1：在BrandMapper层创建selectByCompanyNameAndStatus方法

```
package com.zsh.mapper;

import com.zsh.domain.Brand;
import org.apache.ibatis.annotations.Param;
import java.util.List;

public interface BrandMapper {
```

```
    List<Brand> selectAll();
    List<Brand> selectByCompanyName(String companyName);

    List<Brand> selectByCompanyNameAndStatus
            (@Param("companyName") String companyName,
             @Param("status") Integer status);
}
```

步骤2：重构BrandMapper.xml

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCompanyNameAndStatus" resultMap="brandMap">
        select *
        from brand where company_name like concat('%',#{companyName},'%')
        and status=#{status}
    </select>
</mapper>
```

步骤3：创建测试方法testSelectByCompanyNameAndStatus进行测试

```
    @Test
    public void testSelectByCompanyNameAndStatus() throws IOException {
        List<Brand> brands=brandMapper.selectByCompanyNameAndStatus("科技有限",0);
        System.out.println(brands);
    }
```

运行结果截图如下：

```
[DEBUG] 00:55:51.262 [main] c.z.m.B.selectByCompanyNameAndStatus - ==>  Preparing: select * from brand where comp
[DEBUG] 00:55:51.293 [main] c.z.m.B.selectByCompanyNameAndStatus - ==> Parameters: 科技有限(String), 0(Integer)
[DEBUG] 00:55:51.319 [main] c.z.m.B.selectByCompanyNameAndStatus - <==      Total: 1
[Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)]
```

方法二：通过封装实体对象数据进行传参

步骤1：在BrandMapper层创建selectByBrand方法

```
package com.zsh.mapper;

import com.zsh.domain.Brand;
import org.apache.ibatis.annotations.Param;
import java.util.List;
```

```
public interface BrandMapper {
    List<Brand> selectAll();
    List<Brand> selectByCompanyName(String companyName);
    List<Brand> selectByCompanyNameAndStatus
            (@Param("companyName") String companyName,
             @Param("status") Integer status);

    List<Brand> selectByBrand(Brand brand);
}
```

步骤2：重构BrandMapper.xml

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByBrand" resultMap="brandMap">
        select *
        from brand where company_name like concat('%',#{companyName},'%')
        and status=#{status}
    </select>
</mapper>
```

步骤3：创建测试方法testSeleteByBrand进行测试

```
    @Test
    public void testSelectByBrand() throws IOException {
        Brand brand=new Brand();
        brand.setCompanyName("科技有限");
        brand.setStatus(0);
        List<Brand> brands=brandMapper.selectByBrand(brand);
        System.out.println(brands);
    }
```

运行结果截图如下：

```
[DEBUG] 01:05:41.664 [main] c.z.m.B.selectByBrand - ==>  Preparing: select * from brand where company_name like concat('%',?,'%') and status=?
[DEBUG] 01:05:41.687 [main] c.z.m.B.selectByBrand - ==> Parameters: 科技有限(String), 0(Integer)
[DEBUG] 01:05:41.716 [main] c.z.m.B.selectByBrand - <==      Total: 1
[Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)]
```

方法三：通过封装`Map`对象数据进行传参

步骤1：在BrandMapper接口层创建selectByCondition方法

```java
package com.zsh.mapper;

import com.zsh.domain.Brand;
import org.apache.ibatis.annotations.Param;
import java.util.List;
import java.util.Map;

public interface BrandMapper {
    List<Brand> selectAll();
    List<Brand> selectByCompanyName(String companyName);
    List<Brand> selectByCompanyNameAndStatus
            (@Param("companyName") String companyName,
             @Param("status") Integer status);
    List<Brand> selectByBrand(Brand brand);

    List<Brand> selectByCondition(Map map);
}
```

步骤2：重构BrandMapper.xml

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCondition" resultMap="brandMap">
        select *
        from brand where company_name like concat('%',#{companyName},'%')
        and status=#{status}
    </select>
</mapper>
```

步骤3：创建测试方法testSeleteByCondition进行测试

```java
    @Test
    public void testSelectByCondition() throws IOException {
        Map map=new HashMap();
        map.put("companyName","科技有限");
        map.put("status",0);
        List<Brand> brands=brandMapper.selectByCondition(map);
        System.out.println(brands);
    }
```

运行结果截图如下：

```
[DEBUG] 01:13:07.421 [main] c.z.m.B.selectByCondition - ==>  Preparing: select * from brand where company_name like concat('%',?,'%') and status=?
[DEBUG] 01:13:07.450 [main] c.z.m.B.selectByCondition - ==> Parameters: 科技有限(String), 0(Integer)
[DEBUG] 01:13:07.486 [main] c.z.m.B.selectByCondition - <==      Total: 1
[Brand(id=1, brandName=格力, companyName=格力科技有限公司, ordered=1, description=格力改变世界, status=0)]
```

A-3-3 总结：

方式1-直接通过参数-缺点：参数一变，就要修改接口方法
`selectByCompanyNameAndStatus(@Param("companynName") String companyName, ...)`
方法2：通过封装实体对象数据进行传参数-缺点：如果数据来自多个对象，也要修改接口方法
`selectByBrand(Brand brand)`
方式3：通过封装Map对象数据进行传参数-推荐
`selectByCondition(Map map)`

A-4 动态查询-where-if

A-4-1 案例需求

A-4-2 案例代码

步骤1：重构映射文件xml-`<select id="selectByCondition">...<select>`

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
```

```
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCondition1" resultMap="brandMap">
        select * from brand
        <where>
            <if test="companyName!=null and companyName!=''">
                and company_name like concat('%',#{companyName},'%')
            </if>
            <if test="brandName!=null and brandName!=''">
                and brand_name like concat('%',#{brandName},'%')
            </if>
            <if test="status!=null">
                and status=#{status}
            </if>
        </where>
    </select>
</mapper>
```

步骤2：测试代码（略）

A-4-3 总结

```
if 标签：条件判断
- test 属性：逻辑表达式 与是and,或是or

where 标签
- 作用：
  - 替换where关键字
  - 会动态的去掉第一个条件前的 and （如果有and的话，没有也不会报错）
  - 如果所有的参数没有值则不加where关键字
```

A-5 动态查询-choose-when

A-5-1 案例需求

A-5-2 案例代码

步骤1：重构映射文件xml-`<select id="selectByCondition">...<select>`

若一个when满足了，则后续的所有when都会忽略

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <resultMap id="brandMap" type="Brand">
        <id column="id" property="id"></id>
        <result column="brand_name" property="brandName"></result>
        <result column="company_name" property="companyName"></result>
        <result column="ordered" property="ordered"></result>
        <result column="description" property="description"></result>
        <result column="status" property="status"></result>
    </resultMap>

    <select id="selectByCondition1" resultMap="brandMap">
        select * from brand
        <where>
            <choose>
                <when test="companyName!=null and companyName!=''">
                    and company_name like concat('%',#{companyName},'%')
                </when>
                <when test="brandName!=null and brandName!=''">
                    and brand_name like concat('%',#{brandName},'%')
                </when>
                <when test="status!=null">
                    and status=#{status}
                </when>
            </choose>
        </where>
    </select>
</mapper>
```

步骤2：创建测试代码（略）

A-6 SQL语句特殊符号处理

A-6-1 案例需求：查询id<45的用户信息

A-6-2 总结

总结：方式一：使用转义字符

符号	原始字符	转义字符
大于	>	>
大于等于	>=	>=
小于	<	<
小于等于	<=	<=
和	&	&
单引号	'	'
双引号	"	"

方式二：使用CD
<![CDATA[内容]]>

B.新增

B-1 单行新增

步骤1-在`BrandMapper`层创建insert方法

```
package com.zsh.mapper;

import com.zsh.domain.Brand;

public interface BrandMapper {
    int insert(Brand brand);
}
```

步骤2-重构`BrandMapper.xml`

参考代码-普通新增

```
    <insert id="insert">
        insert into brand (brand_name, company_name, ordered, description,
status)
        values (#{brandName},#{companyName},#{ordered},#{description},#
{status});
    </insert>
```

参考代码-返回新增成功对象的id

```
    <insert id="insert" useGeneratedKeys="true" keyProperty="id">
        insert into brand (brand_name, company_name, ordered, description,
status)
        values (#{brandName},#{companyName},#{ordered},#{description},#
{status});
    </insert>
```

步骤3-创建`testInsert`测试方法进行测试

```
    @Test
    public void testInsert() throws IOException {
        Brand brand = new Brand();
        brand.setBrandName("小米");
        brand.setCompanyName("小米科技有限公司");
        brand.setOrdered(4);
        brand.setDescription("小米科技天下无敌");
        brand.setStatus(1);
        brandMapper.insert(brand);
    }
```

B-2 批量新增

步骤1-在`BrandMapper`层创建i`nsertBatch`方法

```
package com.zsh.mapper;

import com.zsh.domain.Brand;
import java.util.List;

public interface BrandMapper {
    int insertBatch(@Param("brands") List<Brand> brands);
}
```

步骤2-重构`BrandMapper.xml`

```xml
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <insert id="insertBatch">
        insert into brand values
        <foreach collections="brand" item="brand" separator=",">
            (#{brand.brandName},#{brand.companyName},#{brand.ordered},#
{brand.decription},#{brand.status})
        </foreach>
        ;
    </insert>
</mapper>
```

步骤3-创建 `testInsertBatch` 测试方法进行测试

```java
    @Test
    public void testInsertBatch() throws IOException {
        Brand brand1 = new Brand();
        brand1.setBrandName("小米");
        brand1.setCompanyName("小米科技有限公司");
        brand1.setOrdered(7);
        brand1.setDescription("小米科技天下无敌");
        brand1.setStatus(1);

        Brand brand2 = new Brand();
        brand2.setBrandName("联想");
        brand2.setCompanyName("联想科技有限公司");
        brand2.setOrdered(8);
        brand2.setDescription("联想电脑牛逼");
        brand2.setStatus(0);

        brands.add(brand1);
        brands.add(brand2);
        brandMapper.insertBatch(brands);
    }
```

C.修改

C-1 案例需求：根据品牌Id，修改品牌信息

C-2 实现步骤

步骤1-在`BrandMapper`层创建`updateById`方法

```
package com.zsh.mapper;

import com.zsh.domain.Brand;

public interface BrandMapper {
    void updateById(Brand brand);
}
```

步骤2-重构`BrandMapper.xml`（动态修改）

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <update id="updateById">
        update brand
        <set>
            <if test="brandName!=null and brandName!=''">
                brand_name=#{brandName}
            </if>
            <if test="companyName!=null and companyName!=''">
                company_name=#{companyName}
            </if>
            <if test="ordered!=null">
                ordered=#{ordered}
            </if>
            <if test="description!=null and description!=''">
                description=#{description}
            </if>
            <if test="status!=null">
                status=#{status}
            </if>
        </set>
        where id=#{id};
    </update>
</mapper>
```

步骤3-创建`testUpdateById`测试方法

```
    @Test
    public void testUpdateById() throws IOException {
        Brand brand=new Brand();
        brand.setId(3);
        brand.setBrandName("小米米");
        brand.setCompanyName("小米米科技有限公司");
        brand.setStatus(0);
        brandMapper.updateById(brand);
    }
```

D.删除

D-1 单行删除

步骤1-在`BrandMapper`层创建`deleteById`方法

```
package com.zsh.mapper;

public interface BrandMapper {
    void deleteById(Integer id);
}
```

步骤2-重构`BrandMapper.xml`

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <delete id="deleteById">
        delete from brand where id=#{id};
    </delete>
</mapper>
```

步骤3-创建`testDeleteById`测试方法

```
    @Test
    public void testDeleteById() throws IOException {
        brandMapper.deleteById(8);
    }
```

D-2 批量删除

步骤1-在`BrandMapper`层创建`deleteByIds`方法

```
package com.zsh.mapper;

public interface BrandMapper {
    void deleteByIds(Integer[] ids);
}
```

步骤2-重构BrandMapper.xml

参考代码-方式1

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <delete id="deleteByIds">
        delete from brand
        where id in(
            <foreach collection="array" item="id" separator=",">
                #{id}
            </foreach>
            );
    </delete>
</mapper>
```

参考代码-方式2

```
<?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">
<mapper namespace="com.zsh.mapper.BrandMapper">
    <delete id="deleteByIds">
        delete from brand
        where id in
        <foreach collection="array" item="id" separator="," open="(" close=")">
            #{id}
        </foreach>
    </delete>
</mapper>
```

步骤3-创建 `testDeleteByIds` 测试方法

```
    @Test
    public void testDeleteByIds() throws IOException {
        Integer[] ids={6,7};
        brandMapper.deleteById(ids);
    }
```

E.注解实现CRUD（适合简单逻辑）

E-1 需求-实现角色role表的增删改查

E-2 实现步骤

步骤1-新增实体类`Role`

```
package com.zsh.domain;

import lombok.Data;

@Data
public class Role {
    private Integer id;
    private String rolename;
    private String roledesc;
}
```

步骤2-`RoleMapper`

```
package com.zsh.mapper;

import com.zsh.domain.Role;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Options;
import org.apache.ibatis.annotations.Select;
import java.util.List;

public interface RoleMapper {
    @Select("select * from role")
    List<Role> selectAll();

    @Insert("insert into role values (#{id},#{rolename},#{roledesc});")
    @Options(useGeneratedKeys = true,keyProperty = "id")
    int insert(Role role);
}
```

步骤3-`RoleTest`测试方法（略，记得开启事务--`sqlSession=sqlSessionFactory.openSession(true)`）