一、MyBatis回顾

1、准备工作

1.1、数据库新建表

新建库 mybatisdemo，新建如下表：

```
CREATE TABLE `user` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `username` varchar(255) DEFAULT NULL,
  `password` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1.2、新建 Maven 项目和设置编译版本及添加依赖

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
</properties>

<dependencies>
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.5</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.45</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>log4j</groupId>
        <artifactId>log4j</artifactId>
        <version>1.2.17</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope><!-- 测试阶段才用 -->
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.16.20</version>
        <scope>provided</scope><!-- 编译阶段和测试阶段才使用 -->
    </dependency>
```

```
</dependencies>
```

2、配置文件

在 resources 目录下添加下面这些配置文件。

2.1、db.properties

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/mybatisdemo
username=root
password=admin
```

2.2、mybatis-config.xml

```
<?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>

    <properties resource="db.properties"/>
    <!-- 类型别名配置 -->
    <typeAliases>
        <!-- 包名范围不要太大，一般到 domain，在没有注解的情况下，会使用实体类的首字母小写的非限定类名来作为它的别名 -->
        <package name="cn.wolfcode.domain"/>
    </typeAliases>
    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="${driver}" />
                <property name="url" value="${url}" />
                <property name="username" value="${username}" />
                <property name="password" value="${password}" />
            </dataSource>
        </environment>
    </environments>
    <!-- 关联 Mapper 文件 -->
    <mappers>
        <mapper resource="cn/wolfcode/mapper/UserMapper.xml" />
    </mappers>
</configuration>
```

2.3、log4j.properties

```
log4j.rootLogger=ERROR, stdout
log4j.logger.cn.wolfcode=TRACE
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%5p [%t] - %m%n
```

3、编写实体类

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class User {
    private Long id;
    private String username;
    private String password;
}
```

4、编写 UserMapper.xml

在 resources 目录下新建 cn/wolfcode/mapper/UserMapper.xml，具体内容如下：

```
<?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="cn.wolfcode.mapper.UserMapper">
    <insert id="save" keyColumn="id" keyProperty="id" useGeneratedKeys="true">
        INSERT INTO user(username, password) VALUES(#{username}, #{password})
    </insert>
    <update id="update">
        UPDATE user SET
        username = #{username},
        password = #{password}
        WHERE id = #{id}
    </update>
    <select id="get" resultType="User">
        SELECT id, username, password FROM user WHERE id = #{id}
    </select>

    <delete id="delete">
        DELETE FROM user WHERE id = #{id}
    </delete>
</mapper>
```

5、编写 MyBatisUtil

```
package cn.wolfcode.util;

public abstract class MyBatisUtil {
    private static SqlSessionFactory sqlSessionFactory;
    static {
        InputStream inputStream;
        try {
            inputStream = Resources.getResourceAsStream("mybatis-config.xml");
            sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
    public static SqlSession getSession() {
        return sqlSessionFactory.openSession();
    }
}
```

6、编写单元测试类

在 src/test/java 下新建 cn.wolfcode.mapper.UserMapperTest 类，代码如下：

```
public class UserMapperTest {
    @Test
    public void testSave() throws Exception {
        User user = new User();
        user.setUsername("哈哈");
        user.setPassword("123456");
        SqlSession session = MyBatisUtil.getSqlSession();
        session.insert("cn.wolfcode.mapper.UserMapper.save", user);
        session.commit();
        session.close();
        System.out.println(user);
    }
    @Test
    public void testGet() throws Exception {
        SqlSession session = MyBatisUtil.getSqlSession();
        User user = (User)session.selectOne("cn.wolfcode.mapper.UserMapper.get",
1L);
        session.close();
        System.out.println(user);
    }

}
```


二、Mapper 接口和原理

1、之前代码问题

```
@Test
public void testGet() throws Exception {
    SqlSession session = MyBatisUtil.getSqlSession();
    User user = (User)session.selectOne("cn.wolfcode.crud.mapper.UserMapper.get", 1L);
    session.close();
    System.out.println(user);
}
```

- namespace.id 使用的是 String 类型（第一个参数），一旦编写错误，只有等到运行代码才能报错；
- 传入的实际参数类型不被检查，因为第二个参数类型是 Object。

2、使用 Mapper 接口

2.1、使用注意

类似之前的 DAO，在接口中定义 CRUD 等操作方法。Mapper 组件 = Mapper 接口 + Mapper XML 文件。

- 接口的命名为实体名 Mapper，一般和其对应 XML 文件放一起（只要编译之后字节码文件和 XML 文件在一起）；
- XML 命名空间用其对应接口的全限定名；
- Mapper 接口的方法名要和 Mapper XML 文件元素（select | update | delete | insert） id 值一样；
- 方法的返回类型对应 SQL 元素中定义的 resultType / resultMap 类型；
- 方法的参数类型对应 SQL 元素中定义的 paramterType 类型（一般不写）。

2.2、定义 Mapper 接口

```
package cn.wolfcode.mapper;

public interface UserMapper {
    User get(Long id);
}
```

2.3、使用 Mapper 接口

修改 UserMapperTest 代码，如下：

```
public void testGet() throws Exception{
    SqlSession session = MyBatisUtil.getSession();
    UserMapper userMapper = session.getMapper(UserMapper.class);
    User user = userMapper.get(1L);
    System.out.println(user);
    session.close();
}
```


3、Mapper 接口的原理

通过打印接接口，发现打印的是：class com.sun.proxy.$Proxy5，底层使用的是动态代理（后面 Spring 再讲），生成 Mapper 接口的实现类。

接口是规范，实质做的实现还是要由实现类对象来做，而这个实现类不需要我们写，实现类对象也不由我们创建，这些都 MyBatis 使用动态代理帮我们做了；
我们只需提供 Mapper 接口对应 Mapper XML 文件，获取实现类对象的时候传入 Mapper 接口就可以了（不然 MyBatis 也不知道你要获取哪个 Mapper 接口的实现类对象）；

至于实现类中操作方式底层还是和之前的一样，因为 Mapper XML 命名空间是使用 Mapper 接口的全限定名，方法名又与对应 XML 元素 id 一致，所以可以通过获取调用方法所在 Mapper 接口的权限名和方法名，拼接出 namespace + id，再配合调用方法的实参就可以像之前一样操作了。

三、MyBatis 的参数处理

1、需求

实现一个登录需求，本质就是调用 Mapper 接口中的方法根据用户名和密码查询用户。

2、修改 UserMapper 接口及 UserMapper.xml

里面提供对应的查询方法及 SQL。

```
User login(String username, String password);
```

```
<select id="login" resultType="User">
    SELECT id, username, password FROM user
    WHERE username = #{username} AND password = #{password}
</select>
```

3、问题及原因

发现调用时会抛出异常，原因本质接口的底层还是原来之之前的方法，就只支持一个参数的。

4、封装成一个参数解决方式

修改 UserMapper 接口及 UserMapper.xml。

```
User login1(Map<String, Object> param);
User login2(User param);
```

```
<select id="login1" resultType="User">
    SELECT id, username, password FROM user
    WHERE username = #{username} AND password = #{password}
</select>
<select id="login2" resultType="User">
    SELECT id, username, password FROM user
    WHERE username = #{username} AND password = #{password}
</select>
```


5、使用 @Param 注解解决方式

修改 UserMapper 接口中的 login 方法，在形参贴注解即可。

```
// 本质相当于构建一个 Map 对象，key 为注解 @Param 的值，value 为参数的值。
User login(@Param("username")String username, @Param("password")String
password);
```

6、集合/数组参数

当传递一个 List 对象或数组对象参数给 MyBatis 时，MyBatis 会自动把它包装到一个 Map 中，此时：List 对象会以 list 作为 key，数组对象会以 array 作为 key，也可以使用注解 @Param 设置 key 名。

四、MyBatis 的 # 和 $

1、相同点

都可以获取对象（Map 对象或者 JavaBean 对象）的信息。

2、不同点

- 使用 # 传递的参数会先转换为，无论传递是什么类型数据都会带一个单引号；使用 $ 传递的参数，直接把值作为 SQL 语句的一部分。
- 使用 # 支持把简单类型（八大基本数据类型及其包装类、String、BigDecimal 等等）参数作为值；使用 $ 不支持把简单类型参数作为值。
- 使用 # 好比使用 PrepareStatement，没有 SQL 注入的问题，相对比较安全；使用 $ 好比使用 Statement，可能会有 SQL 注入的问题，相对不安全。

3、代码证明

修改 UserMapper 接口及 UserMapper.xml

```
// #
User queryByUsername1(String username);
// $
User queryByUsername2(@Param("username")String name);

// #
List<User> orderByDesc1(String columnName);
// $
List<User> orderByDesc2(@Param("columnName")String columnName);
```

```
<select id="queryByUsername1" resultType="User">
    SELECT id, username, password FROM user WHERE username = #{username}
</select>
<select id="queryByUsername2" resultType="User">
    SELECT id, username, password FROM user WHERE username = ${username}
</select>

<select id="orderByDesc1" resultType="User">
    SELECT id, username, password FROM user ORDER BY #{columnName} DESC
</select>
<select id="orderByDesc2" resultType="User">
    SELECT id, username, password FROM user ORDER BY ${columnName} DESC
</select>
```


4、使用总结

若需要作为 ORDER BY 或 GROUP BY 子句获取参数值使用 $；若需要作为其它子句获取参数值使用 #。

五、动态 SQL

1、定义

MyBatis 的强大特性之一便是它的动态 SQL。如果你有使用 JDBC 或其它类似框架的经验，你就能体会到根据不同条件拼接 SQL 语句的痛苦。例如拼接时要确保不能忘记添加必要的空格，还要注意去掉列表最后一个列名的逗号。利用动态 SQL 这一特性可以彻底摆脱这种痛苦。

```
public class ProductQueryObject extends QueryObject {
    private String name;//商品名称
    private BigDecimal minSalePrice;//最低零售价
    private BigDecimal maxSalePrice;//最高零售价
    private Long dirId;//商品分类编号
    private String keyword;//关键字

    //自身对象的定义查询
    public void customizedQuery() {
        if (hasLength(name)) {
            super.addQuery("productName LIKE ?", "%" + name + "%");
        }
        if (minSalePrice != null) {
            super.addQuery("salePrice >= ?", minSalePrice);
        }
        if (maxSalePrice != null) {
            super.addQuery("salePrice <= ?", maxSalePrice);
        }
        if (dirId != null && dirId != -1) {
            super.addQuery("dir_id = ?", dirId);
        }
        if(hasLength(keyword)){
            super.addQuery("productName LIKE ? OR brand LIKE ?", "%"+keyword+"%","%"+keyword+"%");
        }
    }
}
```

虽然在以前使用动态 SQL 并非一件易事，但正是 MyBatis 提供了可以被用在任意 SQL 映射语句中的强大的动态 SQL 语言得以改进这种情形。动态 SQL 元素和 JSTL 或基于类似 XML 的文本处理器相似。

2、数据准备

```
CREATE TABLE `employee` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) DEFAULT NULL,
  `sn` varchar(20) DEFAULT NULL,
  `salary` decimal(8,2) DEFAULT NULL,
  `deptId` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=8 DEFAULT CHARSET=utf8;

INSERT INTO `employee` VALUES ('1', '赵一', '001', '800.00', '10');
INSERT INTO `employee` VALUES ('2', '倩儿', '002', '900.00', '10');
INSERT INTO `employee` VALUES ('3', '孙三', '003', '800.00', '20');
INSERT INTO `employee` VALUES ('4', '李四', '004', '1000.00', '30');
INSERT INTO `employee` VALUES ('5', '周五', '005', '900.00', '30');
INSERT INTO `employee` VALUES ('6', '吴六', '006', '1200.00', '40');
INSERT INTO `employee` VALUES ('7', '郑七', '007', '1400.00', '10');
```


3、编写对应的实体类、Mapper 接口及 Mapper XML

```
package cn.wolfcode.domain;

@Setter
@Getter
@ToString
public class Employee {
    private Long id;
    private String name;
    private String sn;
    private BigDecimal salary;
    private Long deptId;
}
```

```
package cn.wolfcode.mapper;

public interface EmployeeMapper {
}
```

在 resources 目录下的 cn/wolfcode/mapper 下新建 EmployeeMapper.xml，内容如下：

```
<?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="cn.wolfcode.mapper.EmployeeMapper">
</mapper>
```

六、动态 SQL 之 if 和 where

1、查询询工资大于等于某个值的员工

1.1、修改 EmployeeMapper 接口及 EmployeeMapper.xml

```
List<Employee> queryByMinSalary(@Param("minSalary")BigDecimal minSalary);
```

```
<select id="queryByMinSalary" resultType="Employee">
    SELECT id, name, sn, salary, deptId
    FROM employee
    <if test="minSalary != null">
        WHERE salary >= #{minSalary}
    </if>
</select>
```


1.2、编写单元测试类

```
public class EmployeeMapperTest {
    @Test
    public void testQueryByMinSalary() {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        List<Employee> employees = employeeMapper.queryByMinSalary(new
BigDecimal("1000"));
        // 若传了 null 就要把所有员工查询出来
        // List<Employee> employees = employeeMapper.queryByMinSalary(null);
        System.out.println(employees);
        session.close();
    }
}
```

2、按照工资范围查询员工

2.1、修改 EmployeeMapper 接口及 EmployeeMapper.xml

```
List<Employee> queryByMinSalaryAndMaxSalary(@Param("minSalary")BigDecimal
minSalary, @Param("maxSalary")BigDecimal maxSalary);
```

```
<select id="queryByMinSalaryAndMaxSalary" resultType="Employee">
    SELECT id, name, sn, salary, deptId
    FROM employee
    <where>
        <if test="minSalary != null">
            AND salary &gt;= #{minSalary}
        </if>
        <if test="maxSalary != null">
            AND salary &lt;= #{maxSalary}
        </if>
    </where>
</select>
```

2.2、编写单元测试方法

```
public class EmployeeMapperTest {
    @Test
    public void testQueryByMinSalaryAndMaxSalary() {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        /*List<Employee> employees = employeeMapper
                .queryByMinSalaryAndMaxSalary(new BigDecimal("1000"), new
BigDecimal("1200"));*/
        List<Employee> employees =
employeeMapper.queryByMinSalaryAndMaxSalary(null, null);
        System.out.println(employees);
        session.close();
    }
}
```


七、动态 SQL 之 set

set 元素会动态前置 SET 关键字，同时也会删掉无关的逗号，若里面条件都不成立，就会去除 SET 关键字。其用来解决更新时丢失数据的问题。

1、需求

给某个员工加工资。

2、错误代码实现

```
<update id="update">
    UPDATE employee SET
        name = #{name},
        sn = #{sn},
        salary = #{salary},
        deptId = #{deptId}
    WHERE id = #{id}
</update>
```

问题：传入的对象某个属性值为 null，则会造成更新丢失数据。

```
<update id="update">
    UPDATE employee SET
        <if test="name != null">
            name = #{name},
        </if>
        <if test="sn != null">
            sn = #{sn},
        </if>
        <if test="salary != null">
            salary = #{salary},
        </if>
        <if test="deptId != null">
            deptId = #{deptId}
        </ifS>
    WHERE id = #{id}
</update>
```

问题：虽然可以解决更新丢失数据的问题，但会造成多逗号或者少逗号的问题（比如就仅第一个条件成立）。

3、使用 set 元素实现


```
<update id="update">
    UPDATE employee
    <set>
        <if test="name != null">
            name = #{name},
        </if>
        <if test="sn != null">
            sn = #{sn},
        </if>
        <if test="salary != null">
            salary = #{salary},
        </if>
        <if test="deptId != null">
            deptId = #{deptId},
        </if>
    </set>
    WHERE id = #{id}
</update>
```

4、编写单元测试方法

```
public class EmployeeMapperTest {
    @Test
    public void testUpdate() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        Employee employee = new Employee();
        employee.setSalary(new BigDecimal("900"));
        employee.setId(3L);
        employeeMapper.update(employee);
        session.commit();
        session.close();
    }
}
```

八、动态 SQL 之 foreach

动态 SQL 的另外一个常用的操作需求是对一个集合进行遍历，通常是在构建 IN 条件语句的时候，这里就会使用到 foreach 元素。

1、需求

批量地根据员工 id 删除员工。

2、代码实现

修改 EmployeeMapper 接口及 EmployeeMapper.xml

```
void batchDelete(@Param("ids")Long[] ids);
```

```
<delete id="batchDelete">
    DELETE FROM employee WHERE id IN
    <!--
        collection  遍历数组或集合的 key 或者属性名
        open        遍历开始拼接的字符串
        index       遍历索引
        item        遍历元素
        separator   每遍历元素拼接字符串
        close       遍历结束拼接字符串
    -->
    <foreach collection="ids" open="(" item="id" separator="," close=")">
        #{id}
    </foreach>
</delete>
```

3、编写单元测试方法

```java
public class EmployeeMapperTest {
    @Test
    public void testBatchDelete() throws Exception {
        SqlSession session = MyBatisUtil.getSession();
        EmployeeMapper employeeMapper = session.getMapper(EmployeeMapper.class);
        employeeMapper.batchDelete(new Long[] {1L, 2L});
        session.commit();
        session.close();
    }
}
```