MyBatis

主讲：崔译

一、简介

- 是一个 持久层(dao) 框架
- 是一个 ORM(Object Relation Mapping 对象关系映射) 框架
- 对 持久层技术（JDBC）的封装
- **JDBC是Java访问数据库的唯一途径**

二、关于持久层框架

- 所有的ORM框架都是对JDBC封装
- 所有的ORM框架，关心的
  - 数据源(dataSource)
    - 数据库连接信息，事务处理方式

      `driverClassName`，`url`，`username`，`password`，`transactionManager`
    - SQL 语句
    - RM 关系映射

三、HelloWorld

1、建库、建表

```
create database mybatis;
use mybatis;
create table t_user(
  id int primary key auto_increment,
  name varchar(200),
  pwd varchar(200),
  age int
)engine=Innodb default charset=utf8;
```

2、添加mybatis相关的dtd文件

`mybatis-3-config.dtd`

`mybatis-3-mapper.dtd`

3、添加jar包

- `mybatis.jar` mybatis 核心包

- `mysql-connector.jar` 数据库驱动包
- `log4j.jar` 选加 日志包

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.4.6</version>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.18</version>
</dependency>
<dependency>
    <groupId>org.log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

4、添加`log4j.properties`

5、配置`mybatis-config.xml`

在src下，根据对应的dtd 创建`mybatis-config.xml`文件

输入以下内容

```
<?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="jdbc">
        <environment id="jdbc">
            <transactionManager type="jdbc"></transactionManager>
            <dataSource type="pooled">
                <property name="url"
                          value="jdbc:mysql://localhost:3306/mybatis?
                                 useUnicode=true&amp;characterEncoding=utf8"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
                <property name="driver" value="com.mysql.jdbc.Driver"/>
            </dataSource>
        </environment>
    </environments>

</configuration>
```

6、配置`userMapper.xml`

在`com/itany/mybatis/dao/mapper`下创建`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="abc">
    <insert id="def">
        insert into
            t_user
                (name,pwd,age)
            values
                ('admin','123',22)
    </insert>
</mapper>
```

在 config 文件中 配置mapper

```
<mappers>
    <!-- 写的是相对于src的路径 -->
    <mapper resource="com/itany/mybatis/dao/mapper/userMapper.xml"></mapper>
</mappers>
```

7、编写java代码

```
package com.itany.mybatis.test;

import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

public class Test01 {

    public static void main(String[] args) {
        // SqlSessionFactoryBuilder
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        //SqlSessioFactory
        SqlSessionFactory sf = builder.build(
                Test01.class.getClassLoader()
                .getResourceAsStream("mybatis-config.xml")
        );
        //SqlSession 打开session，同时开启事务
        SqlSession session = sf.openSession();
        // 参数是mapper文件的namespace.id
        // 前提是mapper文件必须存在于config中
        session.insert("abc.def");
        //提交事务
        session.commit();
    }
}
```

四、HelloWorld-2

1、前5步不变

2、创建dao 接口

```
public interface UserDao {
    public void insertUser();
}
```

3、配置`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" >
<!-- 将namespace改成dao接口的包名.类名 -->
<mapper namespace="com.itany.mybatis.dao.UserDao">

    <!-- 把标签的id 改成 方法名 -->
    <insert id="insertUser">
        insert into
            t_user
                (name,pwd,age)
            values
                ('admin','123',22)
    </insert>


</mapper>
```

4、编写java代码

```
package com.itany.mybatis.test;

import org.apache.ibatis.session.SqlSession;
import org.apache.ibatis.session.SqlSessionFactory;
import org.apache.ibatis.session.SqlSessionFactoryBuilder;

import com.itany.mybatis.dao.UserDao;

public class Test02 {
    public static void main(String[] args) {
        SqlSessionFactoryBuilder builder = new SqlSessionFactoryBuilder();
        SqlSessionFactory sf = builder.build(
                Test02.class.getClassLoader()
                .getResourceAsStream("mybatis-config.xml")
```

```
        );
        SqlSession session = sf.openSession();

        UserDao userDao = session.getMapper(UserDao.class);

        userDao.insertUser();

        //提交事务
        session.commit();

    }
}
```

五、`config`文件

```
<?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>

    <!-- 引入其他的properties文件 -->
    <properties resource="datasource.properties"></properties>

    <!--
        配置jdbc环境(数据源和事务)
            mybatis 支持多数据源配置
            default：默认的数据源，值是某个数据源的id值
     -->
    <environments default="jdbc">
        <!--
            配置一个jdbc环境
         -->
        <environment id="jdbc">
            <!-- 事务管理器
                type：管理器类型
                    jdbc:
                        使用jdbc方式管理事务
                    managed:
                        如果设置为managed，mybatis将不再关心事务
                        事务的管理将交给其他容器（框架）进行管理 spring
             -->
            <transactionManager type="jdbc"></transactionManager>
            <!--
                配置数据源
                    type:
                        pooled 使用数据库连接池
                        unpooled 不使用数据库连接池，每次访问数据库之前都要获取连接对象
                        jndi 向其他容器索要数据库连接对象
             -->

            <dataSource type="pooled">
```

```
                <!--
                <property name="url" value="jdbc:mysql://localhost:3306/mybatis?
useUnicode=true&amp;characterEncoding=utf8"/>
                <property name="username" value="root"/>
                <property name="password" value="root"/>
                <property name="driver" value="com.mysql.jdbc.Driver"/>
                 -->
                 <!-- 根据引入的properties文件中的key 获取 value值 -->
                <property name="url" value="${url}"/>
                <property name="username" value="${username}"/>
                <property name="password" value="${password}"/>
                <property name="driver" value="${driver}"/>
            </dataSource>
        </environment>
    </environments>

    <!-- 指定mapper文件位置 -->
    <mappers>
        <!-- 写的是相对于src的路径 -->
        <mapper resource="com/itany/mybatis/dao/mapper/userMapper.xml">
      </mapper>
    </mappers>

</configuration>
```

六、`mapper`文件

```
<!--
    如果没有dao接口，namespace值可以随便写
    否则，将namespace改成dao接口的包名.类名
-->
<mapper namespace="com.itany.mybatis.dao.UserDao">

    <!--
        如果没有dao接口,id值可以随便写
        否则，把标签的id 改成 接口中的方法名
    -->
    <insert id="insertUser">
        insert into
            t_user
                (name,pwd,age)
            values
                ('admin','123',22)
    </insert>

</mapper>
```

七、单表操作

1、基本的增删改查操作

```
    <!--
        如果没有dao接口,id值可以随便写
        否则，把标签的id 改成 接口中的方法名
    -->
    <insert id="insertUser">
        insert into
            t_user
                (name,pwd,age)
            values
                ('admin','123',22)
    </insert>


    <delete id="deleteById">
        delete from
            t_user
        where
            id = 3
    </delete>


    <update id="update">
        update
            t_user
        set
            name = 'aaa',
            age = 99
    </update>

    <select id="selectAll" resultType="User">
        select
            id,name,pwd,age
        from
            t_user
    </select>
```

dao 接口

```
public interface UserDao {
    public void insertUser();
    public void deleteById();
    public void update();
    public List<User> selectAll();
}
```

2、对象关系映射

2-1、默认情况

Mybatis 对于查询结果，会默认的 将 列名 和 对象属性名 一一对于

将列的值 赋值到 同名的 属性上

2-2、使用重命名

```
<select id="selectUs" resultType="User">
  select
     id,
     username as name,
     password as pwd,
     age
  from
     t_us
</select>
```

2-3、使用`resultMap`

```
<!-- 定义映射关系
  id:（当前mapper文件的）唯一标识
  type: 最终要映射成的类的名字（表名.类名/别名）
  -->
<resultMap type="User" id="rm">
  <!-- 对于主键列，必须使用id标签定义映射关系
   column:列名
   property:属性名
  -->
  <id column="id" property="id"/>
  <!-- 对于普通列，使用result标签定义映射关系 -->
  <result column="username" property="name" />
  <result column="password" property="pwd" />
  <result column="age" property="age"/>
</resultMap>
<select id="selectUs2" resultMap="rm">
        select
            id,
            username,
            password,
            age
        from
            t_us
</select>
```

3、方法参数

Mybatis dao方法支持写0个或者多个参数，但是 官方建议 最多写 一个参数

3-1 对象类型参数

使用#{}语法取出参数对象的对应的属性值

```
<!-- 使用#{}语法，将参数的属性名对应的值取出 -->
<select id="selectByNameAndPwd" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name = #{name} and pwd = #{pwd}
</select>
```

```
public User selectByNameAndPwd(User user);
```

3-2 Map 类型参数

使用#{}语法根据key取出value

```
<select id="selectByNameAndPwd2" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name = #{a} and pwd = #{b}
</select>
```

```
public User selectByNameAndPwd2(Map<String, String> param);
```

```
Map<String, String> param =new HashMap<String, String>();
param.put("a", "bbb");
param.put("b", "123");
```

3-3 简单类型参数

基本数据类型， 对应的包装类，String 等

```
public List<User> selectByName(String name);
```

```
<select id="selectByName" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name = #{name}
</select>
```

如果dao方法中只有一个参数，并且参数类型是简单类型

- `#{}`中的值可以随便写
- 建议使用参数名
- 通用写法：配合`@Param`注解使用

```
// 为name参数取名字 叫做username
public List<User> selectByName( @Param("username") String name);
```

```
<!-- 由于存在注解@Param,所以#{}中只能写username -->
<select id="selectByName" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name = #{username}
</select>
```

3-4 多个参数

```
public List<User> selectByNameAndAge(String name,int age);
```

```
<select id="selectByNameAndAge" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name = #{param1}/#{0}
            and
            age = #{param2}/#{1}
</select>
```

使用注解

```
public List<User> selectByNameAndAge2(
            @Param("name")  String name,
            @Param("age")   int age);
```

4、保存返回主键

```
<insert id="insertUser" parameterType="User"
        useGeneratedKeys="true"
        keyColumn="id"
        keyProperty="id"
    >
        insert into
            t_user
                (name,pwd,age)
            values
                (#{name},#{pwd},#{age})
</insert>
```

```
public void insertUser(User user);
```

```
User u = new User();
u.setAge(66);
u.setName("zhangsan");
u.setPwd("aaa");
userDao.insertUser(u);

int id = u.getId();
System.out.println(id);
```

八、动态SQL

1、if

```
/*
     * 组合查询
     *      user.name 不为null，那么作为查询条件出现在sql中
     *      user.pwd 不为null，那么作为查询条件出现在sql中
     *      user.age 不为null，那么作为查询条件出现在sql中
     */
    public List<User> selectFuzzy(User user);
```

```
<!--
        if 标签
            test 值为boolean类型表达式
                在test中获取参数值的时候，不需要#{},直接写属性名
            如果test值为true
            那么 if正反标签中的sql片段会作为sql的一部分
     -->
    <select id="selectFuzzy" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            1=1
        <if test="name != null">
            and name = #{name}
        </if>
        <if test="age != null">
            and age = #{age}
        </if>
        <if test="pwd != null">
            and pwd = #{pwd}
        </if>
    </select>
```

2、where

```
public List<User> selectFuzzy2(User user);
```

```
    <!--
        where标签
            1. 添加where关键字
            2. 删除前面多余的and
     -->
    <select id="selectFuzzy2" resultType="User">
        select
```

```
            id,
            name,
            pwd,
            age
        from
            t_user
        <where>
            <if test="name != null">
                and name = #{name}
            </if>
            <if test="age != null">
                and age = #{age}
            </if>
            <if test="pwd != null">
                and pwd = #{pwd}
            </if>
        </where>
    </select>
```

3、set

```
public void update2(User user);
```

```
    <!--
        set:
            1. 添加set关键字
            2. 删除末尾多余的 逗号
     -->
    <update id="update2">
        update
            t_user
        <set>
            <if test="name != null">
                name = #{name},
            </if>
            <if test="pwd != null">
                pwd = #{pwd},
            </if>
            <if test="age != null">
                age = #{age},
            </if>
        </set>
        where
            id = #{id}
    </update>
```

4、trim

- 在正反标签内容 前 (prefix) 后 (suffix) 添加关键字

- 删除正反标签内容 开头（prefixOverrides）或者 结尾(suffixOverrides) 的内容

```
    <select id="selectFuzzy3" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        <trim prefix="where" prefixOverrides="and">
            <if test="name != null">
                and name = #{name}
            </if>
            <if test="age != null">
                and age = #{age}
            </if>
            <if test="pwd != null">
                and pwd = #{pwd}
            </if>
        </trim>
    </select>

    <update id="update3">
        update
            t_user
        <trim prefix="set" suffixOverrides=",">
            <if test="name != null">
                name = #{name},
            </if>
            <if test="pwd != null">
                pwd = #{pwd},
            </if>
            <if test="age != null">
                age = #{age},
            </if>
        </trim>
        where
            id = #{id}

    </update>
```

5、foreach

```
public void deleteBatch(@Param("ids") String[] ids);
```

```
    <!--
        foreach
            遍历

            collection ： 要遍历的参数
```

```
            separator: 元素分隔符
            open 遍历前添加的字符串
            close 遍历后添加的字符串
            item 每次遍历的元素的别名(List/Set/Array),value值（Map）
            index 每次遍历的下标（List/Set/Array）,key值（Map）
     -->
    <delete id="deleteBatch">
        delete
        from
            t_user
        where
            id in
            <foreach collection="ids" separator=","
                open="(" close=")" item="id">
                #{id}
            </foreach>
    </delete>
```

6、bind

```
/**
     * 根据name 模糊查询
     * @param name
     * @return
     */
public List<User> selectByName(
            @Param("name") String name);
```

```
<select id="selectByName" resultType="User">
        <!-- 注意bind标签的位置
            绑定一个变量
                name：变量名
                value 变量值
                可以在value中 获取参数值
                    1. 有@Param注解，使用注解对应的值
                    2. 没有注解，_parameter
        -->
        <bind name="n" value=" '%' + name + '%' "></bind>
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
            name like #{n}
</select>
```

7、choose-when-otherwise

```
// 如果age值是-1 那么查询所有,否则 根据age的值查询
public List<User> selectByAge( @Param("age") String age);
```

```
<select id="selectByAge" resultType="User">
        select
            id,
            name,
            pwd,
            age
        from
            t_user
        where
        <choose>
            <when test="age == -1">
                1=1
            </when>
            <!-- 如果age不为null，并且不为空字符串 -->
            <when test="age != null &amp;&amp; age != ''">
                age = #{age}
            </when>
            <otherwise>
                age is null
            </otherwise>
        </choose>
    </select>
```

8、include

```
<sql id="selectCols">
        id,
        name,
        pwd,
        age
</sql>
<select id="selectByPwd" resultType="User">
        select
            <!-- 引入sql片段，refid 是sql片段的id值 -->
            <include refid="selectCols"></include>
        from
            t_user
        where
            pwd = #{pwd}
</select>
```

九、多表关系映射

1、准备工作

```sql
use mybatis;

create table t_dept(
    id int primary key auto_increment,
    name varchar(200)
)engine=Innodb default charset=utf8;

insert into t_dept values (null,'后勤部');
insert into t_dept values (null,'行政部');
insert into t_dept values (null,'人事部');

create table t_emp(
    id int primary key auto_increment,
    name varchar(200),
    salary int,
    deptId int,
    foreign key(deptId) references t_dept(id)
)engine=Innodb default charset=utf8;

insert into t_emp values (null,'张三',5000,1)
insert into t_emp values (null,'李四',8000,2)
insert into t_emp values (null,'李五',5000,2)
insert into t_emp values (null,'王六',5000,3)
insert into t_emp values (null,'张七',5000,1)
insert into t_emp values (null,'张八',5000,1)
insert into t_emp values (null,'王九',5000,3)
insert into t_emp values (null,'王十',5000,3)
```

```java
public class Emp {
    private Integer id;
    private String name;
    private Double salary;
    private Dept dept;
    //constructor... get/set
}
public class Dept {
    private Integer id;
    private String name;
    private Set<Emp> emps;
    //constructor... get/set
}
```

2、多对一映射方式

2-1 方式1

```java
public List<Emp> selectAll1();
```

```
<!--
    将d.id 映射到 dept属性的id属性上
    将d.name 映射到 dept属性的name属性上
-->
<select id="selectAll1" resultType="Emp">
        select
            e.id,
            e.name,
            e.salary,
            e.deptId,
            d.id as 'dept.id',
            d.name as 'dept.name'
        from
            t_emp e
            inner join t_dept d on d.id = e.deptId
</select>
```

2-2 方式2（最常用）

```
    <!--
        association:映射多对一关系
            property：对应的是多方类（Emp）中一方（Dept）的属性名
     -->
    <resultMap type="Emp" id="rm2">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="salary" property="salary"/>
        <association property="dept" javaType="Dept">
            <!-- 此处的配置和resultMap外部的配置完全一致
                column:列名
                property ： dept属性的属性名
            -->
            <id column="did" property="id"/>
            <result column="dname" property="name"/>
        </association>
    </resultMap>
    <select id="selectAll2" resultMap="rm2">
        select
            e.id,
            e.name,
            e.salary,
            e.deptId,
            d.id as did,
            d.name as dname
        from
            t_emp e
            inner join t_dept d on d.id = e.deptId
    </select>
```

2-3 方式3

```
    <resultMap type="Emp" id="rm3">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="salary" property="salary"/>
        <!--
            resultMap 引用其他的映射关系
                值是mapper的namespace.其他resultMap的id值
                如果其他resultMap也在当前mapper文件中，namespace 可以省略
         -->
        <association property="dept" javaType="Dept" resultMap="rm4">
        </association>
    </resultMap>

    <resultMap type="Dept" id="rm4">
        <id column="did" property="id"/>
        <result column="dname" property="name"/>
    </resultMap>
```

2-4 方式4

```
    <!--
        select: 另一个查询的namespace.id
        column: 查询条件，也是当前查询的某个列
        效率低，进行了多次查询，存在N+1问题
            进行了N+1次查询
            1：当前的查询语句 查询多方(Emp)的数据
            N: 在1次查询的查询结果中，每一个不同的deptId，都会多进行一次查询

     -->
    <resultMap type="Emp" id="rm5">
        <id column="id" property="id"/>
        <result column="name" property="name"/>
        <result column="salary" property="salary"/>
        <association property="dept" javaType="Dept"
            select="com.itany.mybatis.dao.DeptDao.selectById"
            column="deptId"
        ></association>
    </resultMap>
    <select id="selectAll5" resultMap="rm5">
        select
            id,
            name,
            salary,
            deptId
        from
            t_emp
    </select>
```

`deptMapper.xml`

```
<select id="selectById" resultType="Dept">
        select
            id,
            name
        from
            t_dept
        where
            id = #{id}
</select>
```

3、一对多映射方式

从`t_dept`查询`t_emp`

最终查询的是`Dept`对象

有三种方式，对应多对一的方式2-方式4

只需要将:

- `association`改成`collection`
- `javaType`改成`ofType`

十、分页

1、导入jar包

```
<!--分页核心包-->
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>5.1.0</version>
</dependency>
<!--依赖包 jsqlparser.jar-->
```

2、配置拦截器

```
<plugins>
        <plugin interceptor="com.github.pagehelper.PageInterceptor">
            <!-- 方言，用于决定分页的sql语句
                    低版本pageHelper 必须写，写的是数据库名或者包名.类名
                    5.x 版本下，dialet不用配置，根据jdbc url 选择合适的分页方式
            -->
            <!--<property name="dialect"
value="com.github.pagehelper.dialect.helper.MySqlDialect"/> -->
            <!-- 默认值为 false，当该参数设置为 true 时，如果 pageSize=0 就会查询出全部的结果 -->
            <property name="pageSizeZero" value="true"/>
            <!-- 分页合理化参数，默认值为false。当该参数设置为 true 时，
            pageNum<=0 时会查询第一页，

             pageNum>pages（超过总数时），会查询最后一页 -->
```

```
            <property name="reasonable" value="true"/>
        </plugin>
</plugins>
```

3、service代码

```
PageHelper.startPage(40, 4);
List<User> userList = userDao.selectAll();
PageInfo info = new PageInfo(userList);

System.out.println("上一页"+info.getPrePage());
System.out.println("下一页"+info.getNextPage());
System.out.println("总页数"+info.getPages());
System.out.println("数据总数"+info.getTotal());
System.out.println("当前页"+info.getPageNum());
System.out.println("是否是第一页"+info.isIsFirstPage());
System.out.println("是否是最后一页"+info.isIsLastPage());
System.out.println("数据"+info.getList());
```