**MyBetis3.2框架技术**

# 1.MyBetis介绍及基本环境搭建

## 1.1 MyBatis介绍：

基于SQL语句的ORM框架， Apache--iBatis。代码库迁致Google Code--MyBatis.

## 1.2 与Hibernate比较

a. 学习成本：简单易学，较接近JDBC

b. 程序灵活性：直接使用SQL，灵活性高

c. 程序执行效率高

d. 可移植性：hibernate较好（与数据库关联在配置中完成，HQL语句与数据库无关）

## 1.3 适用场合

以下场合不建议：

1. 需要支持多种数据库或数据库有移植要求
2. 完全动态SQL，例如：字段都要动态生成
3. 使用的不是关系型数据库

## 1.4 开发步骤（推荐）

|  |
| --- |
| **新建JAVA项目或WEB项目**  **部署jar包（包括数据库驱动包）：下载地址http://code.google.com/p/mybatis**  **编写主配置文件**  **创建数据库及表（如已创建，可省略）**  **创建实体类及SQL映射文件**  **编写数据库接口及实现**  **编写测试类及测试** |

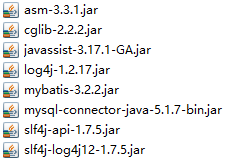
## 1.5 开发示例

**1.5.1 新建项目**

新建java项目或web 项目。

**1.5.2 导入JAR包**

导入mybatis和数据库驱动包、日志包（配置日志配置文件）。



**1.5.3 创建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">  <!--可设置多个运行环境，满足不同需要，如 开发、测试 -->  <configuration>  <!--environments指定数据源环境，default指的是使用哪个数据源 -->  <environments default="development">    <!--environment定义数据源的信息 -->  <environment id="development">    <!-- type="JDBC"表示事务由jdbc连接管理，  type="MANAGED"表示事务由容器来管理 -->  <transactionManager type="JDBC" />    <!-- type="POOLED"表示使用连接池，  type="UNPOOLED"表示不使用连接池 -->  <dataSource type="POOLED">  <property name="driver" value="com.mysql.jdbc.Driver" />  <property name="url" value="jdbc:mysql://localhost:3306/mybatis?useUnicode=true&amp;characterEncoding=UTF-8" />  <property name="username" value="root" />  <property name="password" value="root" />  </dataSource>    </environment>  </environments>    <!-- 加入映射文件信息 -->  <mappers>  <mapper resource="cn/it/entity/DeptMapper.xml" />  </mappers>  </configuration> |

**1.5.4 创建数据库及表(在navicat中操作)**

drop database if exists mybatis;

create database mybatis CHARACTER SET UTF8;

use mybatis;

create table dept(

dept\_id int primary key auto\_increment,

dept\_name varchar(50),

dept\_address varchar(50)

);

insert into dept(dept\_name, dept\_address) values('研发部一部','广州');

insert into dept(dept\_name, dept\_address) values('研发部二部','广州');

insert into dept(dept\_name, dept\_address) values('研发部三部','深圳');

select \* from dept;

**1.5.5 创建实体类：Dept.java**

|  |
| --- |
| package cn.it.entity;  import java.io.Serializable;  public class Dept implements Serializable {  private static final long serialVersionUID = -2525513725816258556L;  private Integer deptId;//部门编号  private String deptName;//部门名称  private String deptAddress;//部门地址  public Integer getDeptId() {  return deptId;  }  public void setDeptId(Integer deptId) {  this.deptId = deptId;  }  public String getDeptName() {  return deptName;  }  public void setDeptName(String deptName) {  this.deptName = deptName;  }  public String getDeptAddress() {  return deptAddress;  }  public void setDeptAddress(String deptAddress) {  this.deptAddress = deptAddress;  }  @Override  public String toString() {  return "Dept [deptId=" + deptId + ", deptName=" + deptName  + ", deptAddress=" + deptAddress + "]";  }  } |

**1.5.6 创建SQL映射文件及修改主配置文件**

SQL映射文件：DeptMapper.xml（可以参考用户手册）

|  |
| --- |
| <?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">  <!-- 命名空间可以任选命名，但最好要定义一定规则，便于后继的使用 -->  <mapper namespace="cn.it.entity.DeptMapper">  <!-- type指定的是对应的实体类 -->  <!-- 一般在查询时使用-->  <resultMap type="cn.it.entity.Dept" id="deptResultMap">    <!-- id用来配置表的主键与类的属性的映射关系 ,column指定表的字段名; property指定类的属性名-->  <id column="dept\_id" property="deptId"/>    <!-- result用来配置 普通字段与类的属性的映射关系 ,column指定的是表的字段名; property指定的是类的属性名-->  <result column="dept\_name" property="deptName"/>  <result column="dept\_address" property="deptAddress"/>    </resultMap>    <!-- 添加一第记录 ; 定义插入的sql语句，通过命名空间+id方式被定位 -->  <insert id="insertDept" parameterType="cn.it.entity.Dept">  <!-- #{} 用来获取传过来的参数 -->  insert into dept(dept\_name,dept\_address) values(#{deptName},#{deptAddress})  </insert>  </mapper> |

**1.5.7 编写数据库操作**

包括操作接口及实现，接口略，实现类为：DeptDaoImpl.jav

package cn.it.dao.impl;

import java.io.IOException;

import java.io.Reader;

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 cn.it.entity.Dept;

public class DeptDaoImpl {

/\*

\* 1.读取配置文件信息

\* 2。构建session工厂

\* 3。创建session

\* 4. 开启事务（可选）

\* 5。处理数据

\* 6。提交、回滚事务

\* 7。关闭session

\*

\*/

public int save(Dept dept){

int i =0;

String path="myBatis-config.xml";

SqlSession session = null;

Reader reader = null;

try {

//获取配置文件的信息

reader = Resources.getResourceAsReader(path);

//构建sessionfactory

SqlSessionFactory sessionFactory = new SqlSessionFactoryBuilder().build(reader);

//创建session

session = sessionFactory.openSession();

//启用事务，这里默认已启用

//执行数据处理,第一个参数用“命名空间+sql id"来定位sql,

//第二个参数用来给sql传参数

i = session.insert("cn.it.entity.DeptMapper.insertDept", dept);

//提交事务

session.commit();

} catch (Exception e) {

e.printStackTrace();

session.rollback();

}finally{

if(reader!=null){

try {

reader.close();

} catch (IOException e) {

e.printStackTrace();

}

}

if(session!=null){

session.close();

}

}

return i;

}

}

**1.5.8 编写测试类**

需要导入junit包

|  |
| --- |
| package cn.it.test;  import static org.junit.Assert.\*;  import org.junit.AfterClass;  import org.junit.BeforeClass;  import org.junit.Test;  import cn.it.dao.impl.DeptDaoImpl;  import cn.it.entity.Dept;  public class TestDeptDaoImpl {  private static DeptDaoImpl deptDaoImpl;  @BeforeClass  public static void setUpBeforeClass() throws Exception {  deptDaoImpl = new DeptDaoImpl();  }  @AfterClass  public static void tearDownAfterClass() throws Exception {  deptDaoImpl = null;  }  @Test  public void test() {  Dept dept = new Dept();  dept.setDeptName("综合部");  dept.setDeptAddress("广州天河");  System.out.println("受影响 的行数："+deptDaoImpl.save(dept));  }  } |

# 2.基本的CRUD操作

## 2.1 准备工作（继续使用前面的库表和代码）

## 2.2 别名与自定义别名

**2.2.1 内置别名**

对常用的 java 类型，已经内置了一些别名支持。不区分大小写.

**2.2.2 自定义别名**

在myBatis的主配置文件给cn.itcast.entity.Dept类创建别名Dept,

后继的DeptMapper.xml配置文件中可以使用别名

<!-- 通过别名简化对类的使用 -->

<typeAliases>

<typeAlias type=*"cn.it.entity.Dept"* alias=*"Dept"* />

</typeAliases>

## 2.3 MyBatisUtil工具类

**public** **class** MyBatisUtil {

**private** **static** **final** ThreadLocal<SqlSession> *threadLocal* = **new** ThreadLocal<SqlSession>();

**private** **static** SqlSessionFactory *sessionFactory*;

**private** **static** String *CONFIG\_FILE\_LOCATION* = "myBatis-config.xml";

**static** {

**try** {

*buildSessionFactory*();

} **catch** (Exception e) {

System.*err*.println("%%%% Error Creating SessionFactory %%%%");

e.printStackTrace();

}

}

**private** MyBatisUtil() {

}

/\*\*

\* Returns the ThreadLocal Session instance. Lazy initialize

\* the <code>SessionFactory</code> if needed.

\*

\* **@return** Session

\* **@throws** Exception

\*/

**public** **static** SqlSession getSession() **throws** Exception {

SqlSession session = (SqlSession) *threadLocal*.get();

**if** (session == **null**) {

**if** (*sessionFactory* == **null**) {

*buildSessionFactory*();

}

session = (*sessionFactory* != **null**) ? *sessionFactory*.openSession()

: **null**;

*threadLocal*.set(session);

}

**return** session;

}

/\*\*

\* build session factory

\*

\*/

**public** **static** **void** buildSessionFactory() {

Reader reader=**null**;

**try** {

reader=Resources.*getResourceAsReader*(*CONFIG\_FILE\_LOCATION*);

*sessionFactory* = **new** SqlSessionFactoryBuilder().build(reader);

} **catch** (Exception e) {

System.*err*.println("%%%% Error Creating SessionFactory %%%%");

e.printStackTrace();

}**finally**{

**try** {

reader.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

/\*\*

\* Close the single session instance.

\*

\*/

**public** **static** **void** closeSession(){

SqlSession session = (SqlSession) *threadLocal*.get();

*threadLocal*.set(**null**);

**if** (session != **null**) {

session.close();

}

}

/\*\*

\* return session factory

\*

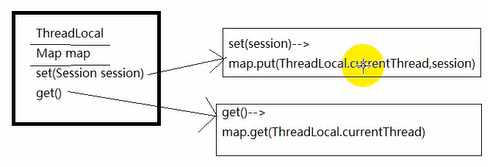
\*/

**public** **static** SqlSessionFactory getSessionFactory() {

**return** *sessionFactory*;

}

}



## 2.3 CRUD操作

**2.3.1 新增操作**

配置文件DeptMapper.xml使用别名， DeptDaoImpl.java新增方法使用工具类。

修改配置文件DeptMapper.xml（使用别名）：

<!--parameterType="Dept"不写时，也能自动根据代码传递的参数Dept自动匹配 内容-->

<insert id=*"insert"* parameterType=*"Dept"*>

insert into dept(dept\_name) values(#{deptName});

</insert>

修改DeptDaoImpl.java新增方法（使用MyBatisUtil.java工具类）：

/\*\*

\* **@param** dept 部门信息

\* **@return** 保存信息后受影响的行数

\*/

**public** **int** saveDept(Dept dept) {

**int** i = 0;

**try** {

session = MyBatisUtil.*getSession*();

i = session.insert("cn.itcast.entity.DeptMapper.insertDept", dept);

session.commit();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

session.rollback();

} **finally** {

**try** {

MyBatisUtil.*closeSession*();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

**return** i;

}

**2.3.2 修改操作**

修改配置文件deptMapper.xml，添加

<update id=*"update"* parameterType=*"Dept"*>

update dept set dept\_name=#{deptName} ,dept\_address=#{deptAddress} where dept\_id=#{deptId}

</update>

修改DeptDaoImpl.java，添加update方法：

**public** **int** update(Dept dept){

SqlSession session = **null**;

**int** i=0;

**try** {

session=MyBatisUtil.*getSession*();

//方法的第一个参数为DeptMapper.xml文件的命名空间+id

i=session.update("cn.itcast.entity.DeptMapper.update",dept);

System.*out*.println("受影响行数："+i);

session.commit();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

session.rollback();

}**finally**{

MyBatisUtil.*closeSession*();

}

**return** i;

}

**2.3.3 删除操作**

修改配置文件deptMapper.xml，添加

<delete id=*"delete"* parameterType=*"Dept">*

delete from dept where dept\_id=#{deptId}

</delete>

修改DeptDaoImpl.java，添加delete方法：

**public** **int** delete(Dept dept){

**int** i = 0;

**try** {

session=MyBatisUtil.*getSession*();

//方法的第一个参数为DeptMapper.xml文件的命名空间+id

i=session.delete("cn.itcast.entity.DeptMapper.delete",dept);

//System.out.println("受影响行数："+i);

session.commit();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

session.rollback();

}**finally**{

MyBatisUtil.*closeSession*();

}

**return** i;

}

**2.3.4 查询操作（返回单条记录）**

配置deptMapper.xml文件的resultMap元素及SQL查询语句

<!-- 表字段和实体属性命名一致时可以不配置 -->

<resultMap id=*"deptResultMap"* type=*"Dept"*>

<id property=*"deptId"* column=*"dept\_id"*/>

<result property=*"deptName"* column=*"dept\_name"*/>

<result property=*"deptAddress"* column=*"dept\_address"*/>

</resultMap>

<!—省略其它的配置信息 -->

<!—返回单条记录，表字段和对应实体属性命名一致时可以不使用resultMap属性配置，直接使用resultType="返回的全类名或别名"，建议使用前者;查询结果为所有字段时，也可以用\*表示 -->

<select id=*"selectOne"* parameterType=*"int"* resultMap=*"deptResultMap"* >

select dept\_id, dept\_name from dept where dept\_id=#{deptId}

</select>

修改DeptDaoImpl.java，添加selectOne方法：

**public** Dept selectOne(**int** deptId){

Dept dept=**null**;

**try** {

session=MyBatisUtil.*getSession*();

dept=(Dept)session.selectOne("cn.itcast.entity.DeptMapper.selectOne",deptId);

System.*out*.println("dept:"+dept);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

MyBatisUtil.*closeSession*();

}

**return** dept;

}

**2.3.5 查询操作（返回多条记录）**

修改配置文件deptMapper.xml，添加

<!-- 返回多条记录，返回结果配置的不是集合类型，而是集合元素的类型；参数也可以通过Map等方式封装 -->

<select id=*"selectList"* parameterType=*"Map"* resultMap=*"deptResultMap"*>

select \* from dept where dept\_name like #{deptName}

</select>

修改DeptDaoImpl.java，添加selectList方法：

**public** List<Dept> selectList(Map map){

List<Dept> depts=**null**;

**try** {

session=MyBatisUtil.*getSession*();

depts=session.selectList("cn.itcast.entity.DeptMapper.selectList",map);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

MyBatisUtil.*closeSession*();

}

**return** depts;

}

测试类代码：

@Test

**public** **void** testSelectList() {

Map map=**new** HashMap();

map.put("deptName", "%研%");

List<Dept> depts=*deptDaoImpl*.selectList(map);

**for**(Dept dept:depts){

System.*out*.println("dept:"+dept);

}

}

## 3.动态SQL操作

## 3.1 准备工作

创建表及库，实体类，配置文件（参考上章节内容），以下为建表和库的SQL：

drop database if exists mybatis;

create database mybatis CHARACTER SET UTF8;

use mybatis;

create table dept(

dept\_id int primary key auto\_increment,

dept\_name varchar(50),

dept\_address varchar(50)

);

insert into dept(dept\_name,dept\_address) values('研发部一部','广州');

insert into dept(dept\_name,dept\_address) values('研发部二部','广州');

insert into dept(dept\_name,dept\_address) values('研发部三部','深圳');

select \* from dept;

## 3.2 IF语句

修改配置文件deptMapper.xml，添加

<!-- 动态IF条件 -->

<select id=*"selectListUseIf"* parameterType=*"Dept"* resultMap=*"deptResultMap"*>

select \* from dept where 1=1

<if test=*"deptId!=null"*>

and dept\_id=#{deptId}

</if>

<if test=*"deptName!=null"*>

and dept\_name=#{deptName}

</if>

<if test=*"deptAddress!=null"*>

and dept\_address=#{deptAddress}

</if>

</select>

修改DeptDaoImpl.java，添加selectListUseIf方法：

//根据参数使用配置文件的IF语句自动填充查询的过滤条件

**public** List<Dept> selectListUseIf(Dept dept){

List<Dept> depts=**null**;

**try** {

session=MyBatisUtil.*getSession*();

depts=session.selectList("cn.itcast.entity.DeptMapper.selectListUseIf",dept);

} **catch** (Exception e) {

e.printStackTrace();

}**finally**{

MyBatisUtil.*closeSession*();

}

**return** depts;

}

## 3.3 WHERE语句

修改配置文件deptMapper.xml，添加

<!-- 动态Where条件 ,一般也需要与if结合使用，与纯if比较，省略了where 1=1-->

<select id=*"selectListUseWhere"* parameterType=*"Dept"* resultMap=*"deptResultMap"*>

select \* from dept

<where>

<if test=*"deptId!=null"*>

and dept\_id=#{deptId}

</if>

<if test=*"deptName!=null"*>

and dept\_name=#{deptName}

</if>

<if test=*"deptAddress!=null"*>

and dept\_address=#{deptAddress}

</if>

</where>

</select>

## 3.4 choose(when,otherwise)语句

修改配置文件deptMapper.xml，添加

<select id=*"selectListUseChoose"* parameterType=*"Dept"* resultMap=*"deptResultMap"*>

select \* from dept where 1=1

<choose>

<when test=*"deptId!=null"*>and dept\_id=#{deptId}</when>

<when test=*"deptName!=null"*>and dept\_name=#{deptName}</when>

<when test=*"deptAddress!=null"*>and dept\_address=#{deptAddress}</when>

<otherwise>and !1 = 1</otherwise>

</choose>

</select>

## 3.5 SET语句

修改配置文件deptMapper.xml，添加

<!--动态set语句可以用来更新数据 -->

<update id=*"updateUseSet"* parameterType=*"Dept"*>

update dept

<set>

<if test=*"deptName!=null"*>dept\_name=#{deptName},</if>

<if test=*"deptAddress!=null"*>dept\_address=#{deptAddress},</if>

</set>

where dept\_id=#{deptId}

</update>

## 3.6 ForEach语句

修改配置文件deptMapper.xml，添加

<!-- 定义根据多个部门ID查询部门相关部门信息的SQL语句 ,resultMap的值是指集合里元素的类型,parameterType不用指定 -->

<select id=*"selectListUseForeach"* resultMap=*"deptResultMap"*>

select \* from dept where dept\_id in

<!-- collection＝"array或list",array用来对应参数为数组，list对应参数为 集合 -->

<foreach collection=*"array"* item=*"deptId"* open=*"("* separator=*","* close=*")"*>

#{deptId}

</foreach>

</select>

## 3.7 include语句

修改配置文件deptMapper.xml，添加

<!-- 使用include语句动态插入表的字段及对应的值 -->

<sql id=*"key"*>

<!--suffixOverrides="," 可以忽略最后“，”号 -->

<trim suffixOverrides=*","*>

<if test=*"deptName!=null"*>

dept\_name,

</if>

<if test=*"deptAddress!=null"*>

dept\_address,

</if>

</trim>

</sql>

<sql id=*"value"*>

<trim suffixOverrides=*","* >

<if test=*"deptName!=null"*>

#{deptName},

</if>

<if test=*"deptAddress!=null"*>

#{deptAddress},

</if>

</trim>

</sql>

<insert id=*"insertUseInclude"* parameterType=*"Dept"*>

insert into dept(

<include refid=*"key"* />

) values(

<include refid=*"value"*/>

)

</insert>

## 练习

1。完成环境的配置及部门信息的添加

2。解决sqlsession线程安全问题（工具类封装）

3。完成单个部门查询和多个部门查询

4。根据指定部门编号（列表）修改部门信息地址为“武汉”