**Mybatis**

一.持久层

（视图层 jsp（运行在服务器端，太笨重）、freemark（模板技术），bootrap）

（web层---servlet（太过笨重生命周期太长）---struts spring mvc）

（持久层（与数据库打交道）---jdbc原始强大（最快的），hibernate,mybatis,jpa,jdbTemplet）

二.mybatis（持久层的轻量级orm框架）框架 mybatis是一个轻量级的jdbc框架

orm*对象关系映射（Object Relational Mapping，简称ORM）*

mybatis（持久层的轻量级orm框架）框架

类对应表

属性对应表的字段

对象对应表的一行数据

优缺点：

jdbc技术-----速度最快、效率最低

缺点：

1、存在硬编码。（解决办法：把java代码和sql代码分离）

2、数据库管理不方便

3、数据库连接频繁关闭和打开。（connection ---因为远程拿mysql数据，所以受到网络影响）

连接池 DataSource类 连接池的大小=2\*CPU内核数+1

4、preparedStatement预处理占位符的处理麻烦

5、查询数据需要手动封装的对象，放入集合

Mybatis的使用步骤

上http://www.mybatis.cn/（ MyBatis中文官网）

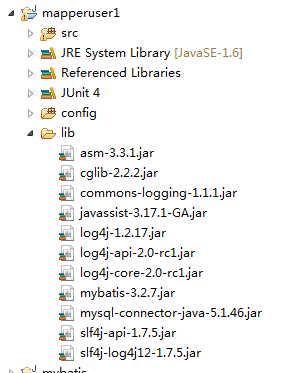
点击下载



下载以后将包解压！



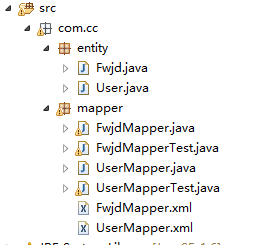
将这一个驱动和lib包下所有驱动都要放入java工程里面，放的位置就是在工程下建立一个lib文件夹，记得放入以后都要buildpath哦！

这就是buildpath以后的亚子！

记得：除了上述的驱动要安装以外，如果要连接上mysql数据库，那么mysql-connector驱动也是必不可少的。

第二步：现在我们有了所需的驱动，接下来就可以为数据库表格设置增删改查的方法了。

准备好了么？让我们开始吧！

首先在src底下建包用来储存文件，包名com.cc 底下分别建立实体类包entity和映射包mapper。此处为user1表设置实体类User和为Fwjd表设置实体类Fwjd类。

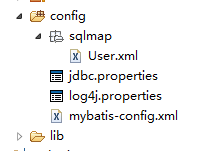
【注意】我建议呢，大家为了后面代码不会出现不必要的麻烦，这里还是尽量类名和表名拥有相似性。

等实体类建立完毕以后，我们就要开始编写配置文件了！！！！可喜可贺！

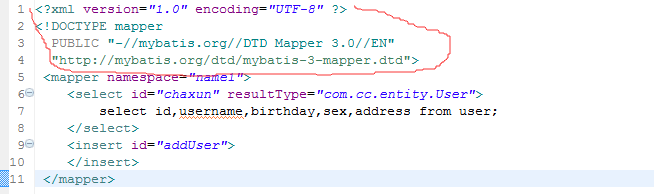
第三步：创建配置文件

首先，我们要创建一个Source文件夹，命名就决定是config吧，因为这个文件夹类型决定了它里面的文件会参与编译，正是有这个需求，所以才需要这个文件夹，千万不要建错了^……^！

下面给出截图！

大家看这里，是不是有一个叫sqlmap的包呢！其实这是老实人写法，也较传统写法。就是要给实体类弄出个专属的配置文件来保存所谓的sql语句。

想不想看看里面有什么？满足一下你的好奇心吧！



就是这个亚子的，画了圈的部分呢，就是sql语句所在的xml文件的必要标签，可以复制去

<http://www.mybatis.org/mybatis-3/zh/configuration.html#mappers> 这个地址复制，也可以用爸爸我给你复制好的哦！

<?xml version="1.0" encoding="UTF-8" ?>

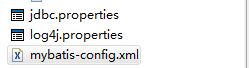
<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd"> 感谢爸爸吧！！！！

下面的部分就是用映射标签<mapper>包裹的sql语句部分，查就用<select>标签，增就用<insert>标签。

不过，传统模式也是很累的,那我们为了简单点，就用更加有意思的mapper框架吧。

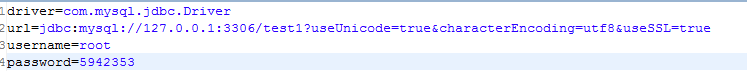
此处三个文件，xml后缀的不用说也知道是配置文件啦！~

jdbc.properties呢，里面封装的就是jdbc的与mysql创建连接那几个代码？

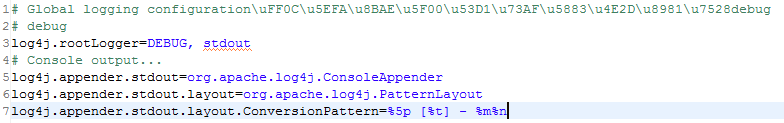
log4g.properties则是一个写好的文件，是记录程序运行的日志。

来！大家看看里面！

首先是jdbc.properties：



然后是log4g.properties



最后就是

mybatis-config.xml

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd"> 此处有底纹的也是可以复制粘贴的！

下面的配置都要写在<configuration>标签里面，

然后子标签第一个就要用<properties resource="jdbc.properties">

</properties>标签进行加载数据库信息文件，这里用的就是上面的jdbc.properties。

我们也可以给实体类取别名减少工作量

<!-- 其他配置2.实体类的别名alias别名-->

<typeAliases>

<!-- 方式一:自己给类取别名 -->

<typeAlias type="com.cc.entity.User" alias="User"/>

<!-- 方式二:自动将指定的包下所有类取别名，别名是类名（自动生成，大小写都可调-->

<package name="com.cc.entity"/>

</typeAliases>

【注意】推荐使用第二种方式，因为一次就能将一个包路径下文件夹内所有类都自动生成一个别名，别名就是类名，首字母大写小写都行。

环境配置（environments）

environments标签可以配置成适应多种环境，这种机制有助于将 SQL 映射应用于多种数据库之中， 现实情况下有多种理由需要这么做。例如，开发、测试和生产环境需要有不同的配置；或者想在具有相同 Schema 的多个生产数据库中 使用相同的 SQL 映射。有许多类似的使用场景。

【注意】虽然呢！可以配置多个环境，但每个 SqlSessionFactory 实例只能选择一种环境。那就是标签内default属性确定的那一个值（值为environment 子标签的id属性。）

如果你想连接两个数据库，就需要创建两个 SqlSessionFactory 实例，每个数据库对应一个

<environments default="development">

<!-- 对应的数据库信息 -->

<environment id="development">

<!-- 事务管理方式：采用jdbc -->

<transactionManager type="JDBC" />

<!-- 数据源的类型 -->

<dataSource type="POOLED">

<!-- 驱动 -->

<property name="driver" value="${driver}" />

<!-- 协议地址 -->

<property name="url" value="${url}" />

<!-- 用户名 -->

<property name="username" value="${username}" />

<!-- 密码 -->

<property name="password" value="${password}" />

</dataSource>

</environment>

</environments>

在确定了数据库环境的设置以后呢，下面就是sql映射文件的加载

<mappers>

<mapper resource="sqlmap/User.xml"/>

<package name="com.cc.mapper"/>

</mappers>

</configuration>

依然是在configuration标签内部的哦，千万不要写出去了！

此处写了两种，蓝色字的，是传统方法一次只加载一个实体类sql配置文件。

绿色字的是mapper方法将所有实体类sql配置文件放在了mapper文件夹下以后，一次性就能加载所有xml配置。

说了这么多，是不是想看一下整体的mybatis-config.xml文件内容。来吧！大度的爸爸给你看了。为了让你看清楚，给你加个底纹！

<configuration>

<!-- 一、加载数据库信息 -->

<!-- 其他配置1.数据库的驱动 -->

<!-- 加载jdbc.properties数据库信息文件 -->

<properties resource="jdbc.properties">

</properties>

<!-- 其他配置2.实体类的别名alias别名-->

<typeAliases>

<!-- 方式一:自己给类取别名 -->

<typeAlias type="com.cc.entity.User" alias="User"/>

<!-- 方式二:自动将指定的包下所有类取别名，别名是类名（自动生成，大小写都可调用） -->

<package name="com.cc.entity"/>

</typeAliases>

<environments default="development">

<!-- 对应的数据库信息 -->

<environment id="development">

<!-- 事务管理方式：采用jdbc -->

<transactionManager type="JDBC" />

<!-- 数据源的类型 -->

<dataSource type="POOLED">

<!-- 驱动 -->

<property name="driver" value="${driver}" />

<!-- 协议地址 -->

<property name="url" value="${url}" />

<!-- 用户名 -->

<property name="username" value="${username}" />

<!-- 密码 -->

<property name="password" value="${password}" />

</dataSource>

</environment>

</environments>

<!-- 二、sql映射文件-->

<mappers>

<mapper resource="sqlmap/User.xml"/>

<package name="com.cc.mapper"/>

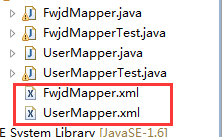
</mappers>

</configuration>

# 创建实体类映射文件

命名方式：千万不要错，错了超哥给你拼命。（就问你怕不怕）

文件命名：实体类名.xml（早期是这样的） 实际上是这样的：实体类名+Mapper.xml



这些配置文件里面就写了重要的增删改查的sql语句。具体代码请看下面：

依然还是这个作为头部

<?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">

 由我来组成头部！！！！ 6不6？！

接下来的代码就是：

<mapper namespace="com.cc.mapper.FwjdMapper">

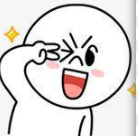
<sql id="Base\_Column\_List">2

jdid,jd,qxid

</sql>

<select id="list" resultType="com.cc.entity.Fwjd">这里如果设置了别名，就可以写成Fwjd

select <include refid="Base\_Column\_List"/> from fwjd

 </select>

<update id="modifyFwjd" parameterType="Fwjd">

update fwjd

<set>

<if test="jd!=null">蓝色代表传过来的属性名

jd=#{jd},紫色代表要赋值的字段名

</if> 红色代表占位符？！

<if test="qxid!=null">名字取什么无所谓的啦

qxid=#{qxid},

</if>

</set>

where jdid = #{jdid}

</update>

<insert id="adduser" parameterType="Fwjd">

<selectKey keyProperty="uid" order="AFTER" resultType="int">

Select last\_insert\_id()

</selectKey> 黄色部分是写在增加数据里面的，如果有表格设置了自增就能在

insert into user1增加了新数据行以后返回增加的那行的主键id

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="uname!=null">

uname,

</if>

<if test="upass!=null">

upass,

</if>

</trim>

values

<trim prefix="(" suffix=")" suffixOverrides=",">

<if test="uname!=null">

#{uname},

</if>

<if test="upass!=null">

#{upass},

</if>

</trim>

</insert>

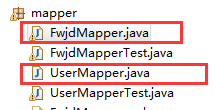
<delete id="dropuser" parameterType="int">

delete from user1 where uid=#{uid}

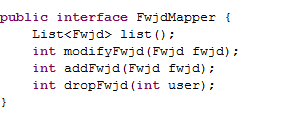
</delete>

</mapper>

现在，我们再为实体类创造一个接口，以便设置抽象方法。

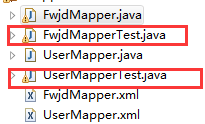


打开以后是这样的，里面就是增删改查的抽象方法。



大家或许注意到了。我的工程里还有个这玩意。

哈哈！这就是单元测试咯。



这个就是单元测试，来看看里面的代码吧！

public class FwjdMapperTest {

SqlSessionFactory sqlSessionFactory =null;

@Before

public void init() throws IOException{

//创建会话工厂（工厂模式）sqlSessionFactory

InputStream inputStream = Resources.getResourceAsStream("mybatis-config.xml");

sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);

}

@Test

public void chaxun() {

SqlSession sqlSession = sqlSessionFactory.openSession();

//相当于根据UserMapper接口和sql映射 创建一个UserMapperImpl代理类

FwjdMapper fwjdMapper = sqlSession.getMapper(FwjdMapper.class);

List<Fwjd> users = fwjdMapper.list();

System.out.println(users);

}

@Test

public void modifyUser() {

SqlSession sqlSession = sqlSessionFactory.openSession();

//相当于根据UserMapper接口和sql映射 创建一个UserMapperImpl代理类

FwjdMapper fwjdMapper = sqlSession.getMapper(FwjdMapper.class);

Fwjd la=new Fwjd();

la.setJdid(1);

la.setJd("四狗子");

la.setQxid(4);

int users = fwjdMapper.modifyFwjd(la);

System.out.println(users);

sqlSession.commit();

sqlSession.close();

}

@Test

public void addUser() {

SqlSession sqlSession = sqlSessionFactory.openSession();

//相当于根据UserMapper接口和sql映射 创建一个UserMapperImpl代理类

FwjdMapper fwjdMapper = sqlSession.getMapper(FwjdMapper.class);

Fwjd la=new Fwjd();

la.setJd("四狗子");

la.setQxid(4);

int users = fwjdMapper.addFwjd(la);

System.out.println(la.getJdid());

sqlSession.commit();

sqlSession.close();

}

@Test

public void dropUser() {

SqlSession sqlSession = sqlSessionFactory.openSession();

//相当于根据UserMapper接口和sql映射 创建一个UserMapperImpl代理类

FwjdMapper fwjdMapper = sqlSession.getMapper(FwjdMapper.class);

int users = fwjdMapper.dropFwjd(5);

System.out.println(users);

sqlSession.commit();

sqlSession.close();

}

}