1. **配置MyBatis**

将MyBatis的配置存放xml文件中，名称可以任意取。以下案例中配置文件名称为sqlMapConfig.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>  <environments default=*"development"*>  <environment id=*"development"*>  <transactionManager type=*"JDBC"*/>  <dataSource type=*"POOLED"*>  <property name=*"driver"* value=*"oracle.jdbc.driver.OracleDriver"*/>  <property name=*"url"* value=*"jdbc:oracle:thin:@172.30.12.22:1521:fund"*/>  <property name=*"username"* value=*"cbpay"*/>  <property name=*"password"* value=*"cbpay"*/>  <!-- 最大连接数 -->  <property name= *"poolMaximumActiveConnections"* value=*"10"* />  <!-- 最大空闲连接数 -->  <property name= *"poolMaximumIdleConnections"* value=*"3"* />  <!-- 连接超时时间（如果某个链接执行了这么长的时间后仍未收回，MyBatis将会打印日志并重新尝试收回该链接） -->  <property name= *"poolTimeToWait"* value=*"20000"* />  <!-- 是否开启侦测查询，用于判断连接是否可用。如果开启则必须设定 poolPingQuery属性。-->  <property name=*"poolPingEnabled"* value=*"true"*/>  <!-- 用于设定侦测查询语句（应该使用最快的sql） -->  <property name=*"poolPingQuery"* value=*"select 1 from dual"* />  <!-- 用于设定侦测查询语句执行的时间间隔，默认为“0”，只有在开启侦查查询时才有效! -->  <property name=*"poolPingConnectionsNotUsedFor"* value=*"20000"* />  </dataSource>  </environment>  </environments>  <!—mapper注册 -->  <mappers>  <mapper resource=*"cn/itcast/mybatis/domain/User.xml"*/>  </mappers>  </configuration> |

1. **启动MyBatis**

DataSource 工具类(单例类)调用getInstance方法初进行始化。通过getSqlSession方法获取sqlSession的对象(使用sqlSession对象进行增删改查操作)。通过closeSqlSession方法关闭sqlSession释放资源。

DataSource工具类的代码如下：

|  |
| --- |
| package cn.itcast.datasource;  import java.io.IOException;  import java.io.InputStream;  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.apache.log4j.Logger;  public class DataSourceUtil {    static Logger logger = Logger.getLogger(DataSourceUtil.class);    private static DataSourceUtil dataSource ;    // 变量私有化  private static SqlSessionFactory sqlSessionFactory;  /\*\*  \* 私有的构造方法  \*/  private DataSourceUtil(){    String filePath = "";  String fileName = "sqlMapConfig.xml" ;  String resource = filePath + fileName;  try {  InputStream inputStream = Resources.getResourceAsStream(resource);  sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);  //打印配置信息用于确认配置是否正确  // Configuration configuration = sqlSessionFactory.getConfiguration();  // Environment environment = configuration.getEnvironment();  // DataSource dataSource =environment.getDataSource();  // logger.info("Environment = ["+dataSource +"] !");  //从environment中获取配置参数!    } catch (IOException e) {  logger.error("init database error , resource = ["+resource+"] !",e);  }  }  /\*\*  \* 获取DataSourceUtil对象  \* @return  \*/  public static DataSourceUtil getInstance() {  if (dataSource == null) {  synchronized(DataSourceUtil.class){  if(dataSource == null){  dataSource = new DataSourceUtil();  }  }  }  return dataSource;  }    /\*\*  \* 获取sqlsession的方法，  \* @param flag 是否自动提交事务  \* @return sqlsession 对象  \*/  public static SqlSession getSqlSession(boolean flag) {  if (sqlSessionFactory == null) {  logger.error("sqlSession is null ! please check init log ! ");  return null;  } else {  SqlSession sqlSession = null;  sqlSession = sqlSessionFactory.openSession(flag);  return sqlSession;  }  }    /\*\*  \* 获取sqlSession的方法，（不会自动提交）  \* @return sqlsesssion 的对象  \*/  public static SqlSession getSqlSession() {  if (sqlSessionFactory == null) {  logger.error("sqlSession is null ! please check init log ! ");  return null;  } else {  SqlSession sqlSession = null;  sqlSession = sqlSessionFactory.openSession();  return sqlSession;  }  }    /\*\*  \* 关闭sqlSession对象  \* @param sqlSession 带关闭的sqlsession实例  \*/  public static void closeSqlSession(SqlSession sqlSession) {  if (sqlSession != null) {  sqlSession.commit();  sqlSession.close();  }  }  } |

1. **Mapper问题**

Mapper用于用于映射sql语句。有两种实现方式xml文件，以及mapper接口。

比较：

* Mapper接口更加直观，可以直接调用方法，使用起来较为简单而且更加直观，但是使用该方式实现sql映射，对于复杂的配置存在一定的局限性，适用于映射关系比较简单的情况。使用方法如下：

mapper. getOne (Id),直接传入参数即可，

* Xml文件方式，此种方式远不如接口方式简单直观。但可以实现较为复杂的映射，使用方法如下：

sqlSession.selectOne("cn.itcast.mybatis.getOne", id);

其中cn.itcast.mybatis.getOne对应mapper. getOne方法，id值为传入参数；

案例如下：

* Mapper接口方式
* 新建mapper接口，如下所示

|  |
| --- |
| **package** cn.itcast.mybatis.domain;  **import** org.apache.ibatis.annotations.ResultType;  **import** org.apache.ibatis.annotations.Select;  **import** cn.itcast.domain.User;  **public** **interface** UserMapper {    @Select("SELECT \* FROM user\_c WHERE id = #{id}")  User getUserById(String id);  } |

* 在配置文件中注册此mapper

|  |
| --- |
| <mappers>  <mapper class=*"cn.itcast.mybatis.domain.UserMapper"*/>  </mappers> |

注：此处也可以导入整个包，如下所示：

|  |
| --- |
| <mappers>  <package name="org.mybatis.builder"/>  </mappers> |

* 在java代码中使用如下方式调用mapper

|  |
| --- |
| UserMapper userMapper = sqlSession.getMapper(UserMapper.**class**);  user = userMapper.getUserById(id); |

* Xml配置方式

|  |
| --- |
| <?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">  <!-- 命名空间用于查找 -->  <mapper namespace=*"cn.itcast.mybatis"*>    <!-- 替代，语句中的内容,简写，在多个查询返回的列相同时使用，可以减少重复工作 -->  <sql id=*"cols"*>  id,name,age,address  </sql>  <!-- 查询所有记录 -->  <select id=*"listAll"* resultType=*"cn.itcast.domain.User"*>  select <include refid=*"cols"*/> from user\_c  </select>  <!-- 查询一个用户 别名替代，简写 -->  <select id=*"getOne"* parameterType=*"string"* resultType=*"cn.itcast.domain.User"*>  select <include refid=*"cols"*/> from user\_c where id= #{id}  </select>    <!-- 插入一个用户 -->  <insert id=*"insertOne"* parameterType=*"cn.itcast.domain.User"* >  insert into user\_c( <include refid=*"cols"*/> ) values (#{id},#{name},#{age},#{address})  </insert>    <!-- 更新一个用户 -->  <insert id=*"updateOne"* parameterType=*"cn.itcast.domain.User"* >  update user\_c set name=#{name},age=#{age},address=#{address} where id=#{id}  </insert>    <!-- 删除一个用户 -->  <update id=*"deleteOne"* parameterType=*"cn.itcast.domain.User"* >  delete from user\_c where id= #{id}  </update>  </mapper> |

1. **连表查询**

对于MyBatis而言，连表查询分为一对一和一对多两种情况。

注：一对一和一对多都是针对，主表和关联表中数据的对应关系而言。例如一对一即主表中一条记录，在关联表中最多只有一条相关数据。

本案例中以xml配置的方式实现!

* 一对一

配置如下：

|  |
| --- |
| <?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">  <!-- 命名空间便于代码中查找 -->  <mapper namespace="cn.itcast.mybatis">  <!-- 查询一个用户 别名替代，简写 -->  <select id="getOne" parameterType="string" resultMap="User">  select a.id, a.name, a.age, a.address , b.rolecode from user\_c a , user\_roles b where a.id = b.id and a.id= #{id}  </select>  <resultMap type="cn.itcast.domain.User" id="User" autoMapping="true">  <association property="userRoles" javaType="cn.itcast.domain.UserRoles" >  <id property="id" column="id"/>  <result property="roleCode" column="rolecode"/>  </association>  </resultMap>  </mapper> |

注：

* select 标签表示次项配置为查询语句。
* id属性，即此查询对应的id值，在代码中使用如下：

sqlSession.selectOne("cn.itcast.mybatis.getOne", id);

其中cn.itcast.mybatis为命名空间，getOne为select标签的id值

* parameterType表示传入参数的类型，
* resultMap表示查询结果类型，次项配置中的User对应，下面配置中的resultMap标签配置的返回值类型（“User”即resultMap标签中的“id”的值）
* association标签用于配置复合类型，即User中存在除了基本类型外的另一个实体类（此例中为cn.itcast.domain.UserRoles）作为变量。
* property属性，是在User实体类中，该变量的名称。
* javaType，即此实体类。
* id标签，配置该实体类的唯一标识，可以为多项，使用逗号隔开即可。
* result标签，表示该实体类其他比变量，一个变量对应一条result标签配置
* 一对多

配置如下：

|  |
| --- |
| <?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">  <!--命名空间用于查找 -->  <mapper namespace="cn.itcast.mybatis">  <!-- 查询一个用户 别名替代，简写 -->  <select id="getOne" parameterType="string" resultMap="User">  select id,name,age,address from user\_c where id= #{id}  </select>  <resultMap type="cn.itcast.domain.User" id="User" autoMapping="true">  <collection property="list" javaType="ArrayList" resultMap="UserRoles" select="getUserRole" column="id"/>  </resultMap>  <resultMap type="cn.itcast.domain.UserRoles" id="UserRoles" autoMapping="true">  </resultMap>    <select id="getUserRole" parameterType="string" resultMap="UserRoles" >  select id,roleCode from user\_roles where id= #{id}  </select>  </mapper> |

注：以上案例中介绍过的标签和属性此处不做赘述。

* resultMap标签
* autoMapping属性表示自动映射，即返回结果后，MyBatis会根据查询结果和属性名和实体类的变量名自动匹配，如果匹配上则自动将信息注入实体类（通过set方法）
* collection标签，表示该项配置对应的实体类属性为集合类型。
* property即实体类中对应的变量名称。
* javaType表示实体类中对应变量的类型。
* resultMap属性，表示该集合类型中的子元素和类型（和resultMap标签中的id属性对应）。
* select属性 表示需要调用的子查询的id值。（调用子查询来获取相应的信息）
* column属性表示关联得列。

1. **事务控制**

1,事务控制的配置在SqlSession中完成。

* 获取不自动提交事务的sqlSession如下所示：

|  |
| --- |
| sqlSession = sqlSessionFactory.openSession(); |

* 获取自动提交事务的sqlSession如下所示：

|  |
| --- |
| sqlSession = sqlSessionFactory.openSession(true); |

注：也可以传入参数“false”获取不自动提交事务的sqlSession。

2,提交事务时使用commit方法，具体如下：

|  |
| --- |
| sqlSession.commit(); |

3，回滚事务使用rollbak方法，具体如下：

|  |
| --- |
| sqlSession.rollback(); |

注：2，事务提交；3，事务回滚必须是在事务不会自动提交的时候才有效。