**3、具体内容**

之前的所有内容都在本处进行总结，而且对于之前的一些概念不清楚的东西（代码会写）那么都可以不用去看了，把本次程序弄会了，一切就都会了，后面也就都会了。

**3.1、程序分层（理解）**

在一个完整的项目之中，对程序进行合理的分层，可以让开发变得更加的方便，也更加的具备层次感，每一层有每一层的开发人员，例如：可以简单的理解为美工 + 程序相分离。而实际上的分层操作，可以这样参考：

如果按照含金量来讲，首先把握住业务层是整个程序的实现关键，但是对于前台显示更加的重要。今天的主要任务是观察业务层和数据层的开发，而到了Java WEB之后，才开始实现显示层和控制层的开发。

在项目之中后台的建立直接有着重要的地位，但是不同层之间最为重要的连接组成部分就是接口，所以整个代码开发之中，对于后台代码就一定要有两个组成接口（业务层接口，给以后的控制层使用、数据层接口，给以后的业务层使用）。

· 数据层（数据访问层，Data Access Object）：指的是执行数据的具体操作，而现在的开发之中，大多数都是针对于数据库的开发，所以在数据层之中的主要任务是负责完成数据的CRUD，而在java之中，如果要想进行数据的CRUD实现，肯定使用java.sql.PreparedStatement接口；

· 业务层（业务对象，Business Object，BO，又或者将其称为Service，服务层），服务层的主要目的是根据业务需求进行数据层的操作，一个业务层要包含多个数据层的操作。

清楚了基本概念之后，那么新的问题就该出现了，如何去区分业务层或者是数据层？下面以玉史先生吃饭为例，说明一下。

如果说现在某一个项目业务非常复杂，可能分为若干个子业务，那么就还需要一个总的业务层操作。

**3.2、实例分析（重点）**

下面以emp数据表（empno、ename、job、hiredate、sal、comm，都是基本字段）为例分析一个操作，客户要求可以实现如下的几个功能：

· 【业务层】增加一个新雇员信息；

|- 〖数据层〗要根据增加的雇员编号查看此雇员是否存在；

|- 〖数据层〗如果雇员不存在则执行插入操作，如果存在则不插入；

· 【业务层】修改一个雇员的信息；

|- 〖数据层〗直接传入新的数据即可，如果没有修改返回的更新行数是0；

· 【业务层】删除一个雇员的信息；

|- 〖数据层〗直接传入要删除的雇员编号即可，如果没有此雇员信息返回的是0；

· 【业务层】根据编号查询一个雇员的信息；

|- 〖数据层〗返回一个雇员的完整信息；

· 【业务层】取得全部雇员的信息，要求可以实现模糊查询和分页显示，查询结果除了返回数据之外，还要求知道模糊或全部查询时所返回的全部数据量：

|- 〖数据层〗模糊或查询全部满足条件的雇员数据，多个数据；

|- 〖数据层〗使用COUNT()进行满足条件的数据统计。

**3.3、准备阶段（重点）**

**3.3.1、VO类：负责数据的传输与包装**

但是现在有一个最为严重的问题出现了，不同层之间（这些层除了数据层要操作SQL之外，那么其他层操作的数据都应该是对象），所以应该有一个负责传输的数据对象，这个对象可以称为Value Object（VO，POJO、TO、PO）。

**但是，现在对于简单Java类的开发原则也发生了一些变化：**

**· 类名称要和表名称保持一致；**

**· 为了日后类的操作方便，所有的简单Java类必须实现java.io.Serializable接口；**

**· 类中不允许出现任何的基本数据类型，只能使用包装类；**

**· 类之中的所有属性都必须封装，必须都编写setter、getter；**

**· 类之中一定要提供有无参构造方法。**

在DAO的开发之中，所有的名称都有严格规定，假设现在的项目的总包名称为：cn.mldn.oracle，那么现在这个VO类的保存包名称就应该是cn.mldn.oracle.vo。

**范例：**定义cn.mldn.oracle.vo.Emp类

|  |
| --- |
| **package** cn.mldn.oracle.vo;  **import** java.io.Serializable;  **import** java.util.Date;  @SuppressWarnings("serial")  **public** **class** Emp **implements** Serializable {  **private** Integer empno ;  **private** String ename ;  **private** String job ;  **private** Date hiredate ;  **private** Double sal ;  **private** Double comm ;  // setter、getter略，自己补充  } |

**3.3.2、DatabaseConnection类：负责数据库连接**

既然现在要完成数据层的开发，那么就一定需要数据库的连接与关闭操作，可是如果将数据库的连接和关闭都写在每一个数据层之中，这样代码过于重复，而且也不方便维护，那么为了方便起见，现在定义一个DatabaseConnection的类，这个类专门负责取得和关闭数据库连接。而这个类定义在cn.mldn.oracle.dbc包之中。

**范例：**定义cn.mldn.oracle.dbc.DatabaseConnection

|  |
| --- |
| **package** cn.mldn.oracle.dbc;  **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.SQLException;  /\*\*  \* 本类的主要功能是负责数据库的连接与关闭的  \* **@author** MLDN  \*/  **public** **class** DatabaseConnection {  **private** **static** **final** String *DBDRIVER* = "oracle.jdbc.driver.OracleDriver" ;  **private** **static** **final** String *DBURL* = "jdbc:oracle:thin:@localhost:1521:MLDN" ;  **private** **static** **final** String *DBUSER* = "scott" ;  **private** **static** **final** String *PASSWORD* = "tiger" ;  **private** Connection conn = **null** ; // 保存连接对象  /\*\*  \* 构造方法的主要目的是进行数据库连接，只要在程序之中实例化了DatabaseConnection对象  \* 那么就表示要进行数据库的连接操作了，所以在构造方法之中连接数据库  \* 在本构造方法之中，如果出现了异常，将直接输出异常信息，因为如果数据库连接都没有了，根本就无法操作  \*/  **public** DatabaseConnection() {  **try** {  Class.*forName*(*DBDRIVER*);  **this**.conn = DriverManager.*getConnection*(*DBURL*, *DBUSER*, *PASSWORD*);  } **catch** (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 取得一个数据库连接对象，这个对象在构造方法中取得  \* **@return** Connection接口对象  \*/  **public** Connection getConnection() {  **return** **this**.conn ;  }  /\*\*  \* 关闭连接，不管是否连接上，执行此操作都不会出错  \*/  **public** **void** close() {  **if** (**this**.conn != **null**) { // 取得了连接  **try** { // 关闭连接  **this**.conn.close() ;  } **catch** (SQLException e) {  e.printStackTrace();  }  }  }  } |

如果在实际的工作之中，按照DAO最早提出的标准，对于数据层的实现类还需要实现数据库的移植操作。即：对于数据库连接类应该变为一个专门负责连接的接口，就好象以下的形式一样：

|  |
| --- |
| **public** **interface** DatabaseConnection {  **public** Connection getConnection() ;  **public** **void** close() ;  } |

而后如果一个项目可能在Oracle或DB2下运行，那么针对于这两种数据库分别定义一个接口实现类，以对应两个不同的数据库连接。但是这种开发已经和现在的模式有些出入了，而且特别的麻烦，所以在本次为了和日后的开发可以更好的联系在一起，只是定义了一个类而已。

**3.4、开发数据层（重点）**

**3.4.1、定义IEmpDAO接口：数据层开发标准**

不同层之间的操作依靠的是接口，所以数据层的开发首先要定义出来的就是标准。那么既然是标准就需要定义的是一个接口，现在很明显针对的是emp表，所以这个接口的名称就应该为“表名称DAO”，即：EmpDAO，但是这里有一个问题了，接口和类的命名要求是一致的，所以为了从名称上区分出接口或者是类，则建议在接口名称前增加一个字母“I”，表示Interface的含义，即：emp这张实体表的操作标准的接口名称为：IEmpDAO，而且这个接口应该保存在cn.mldn.oracle.dao包之中。

那么对于这个接口的开发主要是针对于数据的两种操作（更新、查询），所以从开发标准上对于命名也有着严格的要求，而且必须遵守，基本标准如下：

· 更新操作：以“doXxx()”的方式命名，例如：doCreate()、doUpdate()、doRemvoe()；

· 查询操作，因为查询操作分为两类：

|- 数据查询：以“findXxx()”或“findByXxx()”为主，例如：findAll()、findById()、findByJob()；

|- 统计查询：以“getXxx()”或“getByXxx()”为主，例如：getAllCount()、getByJobCount()。

**范例：**编写IEmpDAO接口的操作标准

|  |
| --- |
| **package** cn.mldn.oracle.dao;  **import** java.util.List;  **import** cn.mldn.oracle.vo.Emp;  **public** **interface** IEmpDAO {  /\*\*  \* 执行数据的增加操作  \* **@param** vo 包含所要增加的数据的VO对象  \* **@return** 如果增加数据成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doCreate(Emp vo) **throws** Exception ;  /\*\*  \* 执行数据的更新操作  \* **@param** vo 包含了新数据的VO对象  \* **@return** 如果修改成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doUpdate(Emp vo) **throws** Exception ;  /\*\*  \* 删除一个雇员的信息  \* **@param** id 要删除的雇员编号  \* **@return** 如果删除成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doRemove(**int** id) **throws** Exception ;  /\*\*  \* 根据雇员编号查询一个雇员的完整信息  \* **@param** id 要查询的雇员编号  \* **@return** 如果没有指定的雇员编号，返回值为null，<br>  \* 如果有指定的雇员信息，则将所有的雇员信息包装到Emp实例化对象之中返回。  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** Emp findById(**int** id) **throws** Exception ;  /\*\*  \* 查询全部的雇员信息  \* **@return** 多个雇员信息使用List返回，如果List集合的size()长度为0，则表示没有数据返回  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** List<Emp> findAll() **throws** Exception ;  /\*\*  \* 分页显示所有雇员的信息，同时可以完成模糊查询  \* **@param** column 要模糊查询的字段名称  \* **@param** keyWord 要模糊查询的数据，如果为空字符串（isEmpty()判断为true，表示空字符串），则表示查询全部  \* **@param** currentPage 当前所在的页  \* **@param** lineSize 没页显示的记录长度  \* **@return** 多个雇员信息使用List返回，如果List集合的size()长度为0，则表示没有数据返回  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** List<Emp> findAll(String column, String keyWord, **int** currentPage,  **int** lineSize) **throws** Exception;  /\*\*  \* 统计模糊查询的数据结果，使用COUNT()函数进行统计  \* **@param** column 要模糊查询的字段名称  \* **@param** keyWord 要模糊查询的数据，如果为空字符串（isEmpty()判断为true，表示空字符串），则表示查询全部  \* **@return** 会根据数据量的多少返回数据的长度，如果没有数据返回0  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **int** getAllCount(String column, String keyWord) **throws** Exception;  } |

现在开发的标准只是满足于程序需求的提出需要。

**3.4.2、定义IEmpDAO接口的实现类**

既然在接口中已经定义了数据层的操作标准，那么对于实现类只需要遵循数据层的CRUD操作即可，但是对于DAO接口的实现类需要有明确的定义，要求将其定义在：cn.mldn.oracle.dao.impl包之中。

**范例：**定义EmpDAOImpl子类

· 现在有如下一种的子类实现接口方式：

|  |
| --- |
| @Override  **public** **boolean** doCreate(Emp vo) **throws** Exception {  DatabaseConnection dbc = **new** DatabaseConnection();  String sql = "INSERT INTO emp (empno,ename,job,hiredate,sal,comm) VALUES (?,?,?,?,?,?)";  PreparedStatement pstmt = dbc.getConnection().prepareStatement(sql);  pstmt.setInt(1, vo.getEmpno());  pstmt.setString(2, vo.getEname());  pstmt.setString(3, vo.getJob());  pstmt.setDate(4, **new** java.sql.Date(vo.getHiredate().getTime()));  pstmt.setDouble(5, vo.getSal());  pstmt.setDouble(6, vo.getComm());  **if** (pstmt.executeUpdate() > 0) {  **return** **true** ;  }  dbc.close() ;  **return** **false**;  } |

如果真的按照这种方式实现的程序，有两个重要问题：

· 对于数据层之中给出的若干方法，由服务层调用，一个服务层要执行N个数据层，那么每次执行的时候打开一次关闭一次数据库？

· 按照异常的处理机制，如果现在执行的过程之中出现了错误，那么顺着throws就结束调用了，数据库就再也无法关闭了。

按照之前的分析，一个业务要进行多个数据层操作，所以数据库连接与关闭交给业务层做最合适，而数据层只需要有一个Connection对象就可以操作了，它不需要关心这个对象是从那里来的，怎么来的，只关心能不能使用。

|  |
| --- |
| **package** cn.mldn.oracle.dao.impl;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.util.ArrayList;  **import** java.util.List;  **import** cn.mldn.oracle.dao.IEmpDAO;  **import** cn.mldn.oracle.vo.Emp;  **public** **class** EmpDAOImpl **implements** IEmpDAO {  **private** Connection conn;  **private** PreparedStatement pstmt;  **public** EmpDAOImpl(Connection conn) {  **this**.conn = conn;  }  @Override  **public** **boolean** doCreate(Emp vo) **throws** Exception {  String sql = "INSERT INTO emp (empno,ename,job,hiredate,sal,comm) VALUES (?,?,?,?,?,?)";  **this**.pstmt = **this**.conn.prepareStatement(sql);  **this**.pstmt.setInt(1, vo.getEmpno());  **this**.pstmt.setString(2, vo.getEname());  **this**.pstmt.setString(3, vo.getJob());  **this**.pstmt.setDate(4, **new** java.sql.Date(vo.getHiredate().getTime()));  **this**.pstmt.setDouble(5, vo.getSal());  **this**.pstmt.setDouble(6, vo.getComm());  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true**;  }  **return** **false**;  }  @Override  **public** **boolean** doUpdate(Emp vo) **throws** Exception {  String sql = "UPDATE emp SET ename=?,job=?,hiredate=?,sal=?,comm=? WHERE empno=?";  **this**.pstmt = **this**.conn.prepareStatement(sql);  **this**.pstmt.setString(1, vo.getEname());  **this**.pstmt.setString(2, vo.getJob());  **this**.pstmt.setDate(3, **new** java.sql.Date(vo.getHiredate().getTime()));  **this**.pstmt.setDouble(4, vo.getSal());  **this**.pstmt.setDouble(5, vo.getComm());  **this**.pstmt.setInt(6, vo.getEmpno());  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true**;  }  **return** **false**;  }  @Override  **public** **boolean** doRemove(**int** id) **throws** Exception {  String sql = "DELETE FROM emp WHERE empno=?";  **this**.pstmt = **this**.conn.prepareStatement(sql);  **this**.pstmt.setInt(1, id);  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true**;  }  **return** **false**;  }  @Override  **public** Emp findById(**int** id) **throws** Exception {  Emp emp = **null**;  String sql = "SELECT empno,ename,job,hiredate,sal,comm FROM emp WHERE empno=?";  **this**.pstmt = **this**.conn.prepareStatement(sql);  **this**.pstmt.setInt(1, id);  ResultSet rs = **this**.pstmt.executeQuery();  **if** (rs.next()) {  emp = **new** Emp();  emp.setEmpno(rs.getInt(1));  emp.setEname(rs.getString(2));  emp.setJob(rs.getString(3));  emp.setHiredate(rs.getDate(4));  emp.setSal(rs.getDouble(5));  emp.setComm(rs.getDouble(6));  }  **return** emp;  }  @Override  **public** List<Emp> findAll() **throws** Exception {  List<Emp> all = **new** ArrayList<Emp>();  String sql = "SELECT empno,ename,job,hiredate,sal,comm FROM emp";  **this**.pstmt = **this**.conn.prepareStatement(sql);  ResultSet rs = **this**.pstmt.executeQuery();  **while** (rs.next()) {  Emp emp = **new** Emp();  emp.setEmpno(rs.getInt(1));  emp.setEname(rs.getString(2));  emp.setJob(rs.getString(3));  emp.setHiredate(rs.getDate(4));  emp.setSal(rs.getDouble(5));  emp.setComm(rs.getDouble(6));  all.add(emp);  }  **return** all;  }  @Override  **public** List<Emp> findAll(String column, String keyWord, **int** currentPage,  **int** lineSize) **throws** Exception {  List<Emp> all = **new** ArrayList<Emp>();  String sql = "SELECT \* FROM ("  + "SELECT empno,ename,job,hiredate,sal,comm,ROWNUM rn FROM emp WHERE "  + column + " LIKE ? AND ROWNUM<=?) temp " + " WHERE temp.rn>? ";  **this**.pstmt = **this**.conn.prepareStatement(sql);  **this**.pstmt.setString(1, "%" + keyWord + "%");  **this**.pstmt.setInt(2, currentPage \* lineSize);  **this**.pstmt.setInt(3, (currentPage - 1) \* lineSize);  ResultSet rs = **this**.pstmt.executeQuery();  **while** (rs.next()) {  Emp emp = **new** Emp();  emp.setEmpno(rs.getInt(1));  emp.setEname(rs.getString(2));  emp.setJob(rs.getString(3));  emp.setHiredate(rs.getDate(4));  emp.setSal(rs.getDouble(5));  emp.setComm(rs.getDouble(6));  all.add(emp);  }  **return** all;  }  @Override  **public** **int** getAllCount(String column, String keyWord) **throws** Exception {  String sql = "SELECT COUNT(empno) FROM emp WHERE " + column + " LIKE ?";  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  **this**.pstmt.setString(1, "%" + keyWord + "%");  ResultSet rs = **this**.pstmt.executeQuery() ;  **if** (rs.next()) {  **return** rs.getInt(1) ;  }  **return** 0;  }  } |

**3.4.3、定义DAO工厂类**

由于不同层之间只能依靠接口取得对象，所以就一定需要定义工厂操作类，工厂类定义在cn.mldn.oracle.factory包之中，名称为DAOFactory。

**范例：**定义工厂类

|  |
| --- |
| **package** cn.mldn.oracle.factory;  **import** java.sql.Connection;  **import** cn.mldn.oracle.dao.IEmpDAO;  **import** cn.mldn.oracle.dao.impl.EmpDAOImpl;  **public** **class** DAOFactory {  **public** **static** IEmpDAO getIEmpDAOInstance(Connection conn) {  **return** **new** EmpDAOImpl(conn) ;  }  } |

**3.5、开发业务层（重点）**

**3.5.1、开发业务层标准**

业务层以后也是需要留给其他层进行调用的，所以业务层定义的时候也需要首先定义出操作标准，而这个标准也依然使用接口完成，对于业务层，接口命名要求：表名称 + Service，例如：IEmpService，表示操作Emp表的业务。

**范例：**在cn.mldn.oracle.service包中定义IEmpService接口

|  |
| --- |
| **package** cn.mldn.oracle.service;  **import** java.util.Map;  **import** cn.mldn.oracle.vo.Emp;  **public** **interface** IEmpService {  /\*\*  \* 调用数据库的增加操作，操作流程如下：<br>  \* <li>首先要使用IEmpDAO接口中的findById()方法，根据要增加的id查看指定的雇员信息是否存在；  \* <li>如果要增加的雇员信息不存在，则执行IEmpDAO接口的doCreate()方法，并将结果返回；  \* **@param** vo 包装数据的对象  \* **@return** 如果增加成功，返回true，如果雇员编号存在或者是增加失败，返回false  \* **@throws** Exception 有异常交给被调用处处理  \*/  **public** **boolean** insert(Emp vo) **throws** Exception;  /\*\*  \* 执行数据的更新操作，操作的时候直接调用IEmpDAO接口的doUpdate()方法，并将更新结果返回  \* **@param** vo 包装数据的对象  \* **@return** 如果修改成功，返回true，如果数据不存在或修改失败，返回false  \* **@throws** Exception 有异常交给被调用处处理  \*/  **public** **boolean** update(Emp vo) **throws** Exception ;  /\*\*  \* 执行数据的删除操作，删除操作的时候调用IEmpDAO接口的doRemove()方法  \* **@param** id 要删除雇员的id  \* **@return** 如果删除成功，返回true，如果数据不存在或删除失败，则返回false  \* **@throws** Exception 有异常交给被调用处处理  \*/  **public** **boolean** delete(**int** id) **throws** Exception ;  /\*\*  \* 根据雇员的编号取得全部的信息  \* **@param** id 雇员编号  \* **@return** 如果雇员存在则将数据包装为Emp对象返回，如果数据不存在则返回null  \* **@throws** Exception 有异常交给被调用处处理  \*/  **public** Emp get(**int** id) **throws** Exception ;  /\*\*  \* 查询全部或者是模糊查询全部数据，查询的同时可以返回满足此查询的数据量，在调用的时候需要执行以下操作：<br>  \* <li>查询全部的雇员信息：需要IEmpDAO接口的findAll()方法；  \* <li>查询满足条件的雇员数量：使用IEmpDAO接口的getAllCount()方法操作；  \* **@param** column 模糊查询的字段  \* **@param** keyWord 模糊查询的关键字  \* **@param** currentPage 当前所在页  \* **@param** lineSize 每页显示的数据长度  \* **@return** 由于在进行数据返回的时候，此方法要返回两类数据：List<Emp>、int，使用Map返回：<br>  \* <li>返回值1：key = allEmps，value = findAll()；  \* <li>返回值2：key = empCount，value = getAllCount()；  \* **@throws** Exception 有异常交给被调用处处理  \*/  **public** Map<String, Object> list(String column, String keyWord,  **int** currentPage, **int** lineSize) **throws** Exception;  } |

**3.5.2、定义业务层标准的实现类**

如果现在要想实现业务层的标准，必须有一个原则先把握住：一个业务层的方法操作要调用多个数据层，同时每个业务要处理数据库的打开和关闭。

**范例：**定义标准实现类 —— cn.mldn.oracle.service.impl.EmpServiceImpl

|  |
| --- |
| **package** cn.mldn.oracle.service.impl;  **import** java.sql.Connection;  **import** java.util.HashMap;  **import** java.util.Map;  **import** cn.mldn.oracle.dao.IEmpDAO;  **import** cn.mldn.oracle.dbc.DatabaseConnection;  **import** cn.mldn.oracle.factory.DAOFactory;  **import** cn.mldn.oracle.service.IEmpService;  **import** cn.mldn.oracle.vo.Emp;  **public** **class** EmpServiceImpl **implements** IEmpService {  **private** DatabaseConnection dbc = **new** DatabaseConnection() ;  @Override  **public** **boolean** insert(Emp vo) **throws** Exception {  **try** {  Connection conn = **this**.dbc.getConnection() ; // 取得连接  IEmpDAO dao = DAOFactory.*getIEmpDAOInstance*(conn) ; // 取得DAO接口对象  **if** (dao.findById(vo.getEmpno()) == **null**) { // 没有要查询的雇员信息  **return** dao.doCreate(vo) ; // 返回DAO的结果  }  **return** **false**; // 数据存在，直接返回false  } **catch** (Exception e) {  **throw** e;  } **finally** {  **this**.dbc.close();  }  }  @Override  **public** **boolean** update(Emp vo) **throws** Exception {  **try** {  **return** DAOFactory.*getIEmpDAOInstance*(**this**.dbc.getConnection())  .doUpdate(vo);  } **catch** (Exception e) {  **throw** e;  } **finally** {  **this**.dbc.close();  }  }  @Override  **public** **boolean** delete(**int** id) **throws** Exception {  **try** {  **return** DAOFactory.*getIEmpDAOInstance*(**this**.dbc.getConnection())  .doRemove(id);  } **catch** (Exception e) {  **throw** e;  } **finally** {  **this**.dbc.close();  }  }  @Override  **public** Emp get(**int** id) **throws** Exception {  **try** {  **return** DAOFactory.*getIEmpDAOInstance*(**this**.dbc.getConnection())  .findById(id);  } **catch** (Exception e) {  **throw** e;  } **finally** {  **this**.dbc.close();  }  }  @Override  **public** Map<String, Object> list(String column, String keyWord,  **int** currentPage, **int** lineSize) **throws** Exception {  **try** {  Map<String,Object> map = **new** HashMap<String,Object>() ;  map.put("allEmps",  DAOFactory.*getIEmpDAOInstance*(**this**.dbc.getConnection())  .findAll(column, keyWord, currentPage, lineSize));  map.put("empCount",  DAOFactory.*getIEmpDAOInstance*(**this**.dbc.getConnection())  .getAllCount(column, keyWord));  **return** map;  } **catch** (Exception e) {  **throw** e;  } **finally** {  **this**.dbc.close();  }  }  } |

**3.5.3、定义Service工厂类**

如果要取得IEmpService接口对象，一定也需要使用工厂类，避免耦合问题。

**范例：**定义cn.mldn.oracle.factory.ServiceFactory工厂类

|  |
| --- |
| **package** cn.mldn.oracle.factory;  **import** cn.mldn.oracle.service.IEmpService;  **import** cn.mldn.oracle.service.impl.EmpServiceImpl;  **public** **class** ServiceFactory {  **public** **static** IEmpService getIEmpServiceInstance() {  **return** **new** EmpServiceImpl() ;  }  } |

**3.6、定义测试类**

一切的程序完成之后，下面就需要编写测试程序，对于测试程序现在有两种方法完成：

· 方式一：可以直接编写主方法，自己根据它的返回值结果进行判断是否成功；

· 方式二：利用JUNIT完成，这样的做法标准，而且也方便日后调试。

如果要使用JUNIT则就需要建立一个个的TestCase（测试用例），而且现在再进行测试的时候，应该首先选择的是服务层接口，因为选择不是针对于接口测试，而是针对于方法测试，方法就可以不用自己去编写了。

**范例：**编写测试程序类

|  |
| --- |
| **package** cn.mldn.oracle.test;  **import** java.util.Date;  **import** java.util.Iterator;  **import** java.util.List;  **import** java.util.Map;  **import** junit.framework.TestCase;  **import** org.junit.Test;  **import** cn.mldn.oracle.factory.ServiceFactory;  **import** cn.mldn.oracle.vo.Emp;  **public** **class** IEmpServiceTest {  @Test  **public** **void** testInsert() {  Emp vo = **new** Emp();  vo.setEmpno(9988);  vo.setEname("张三");  vo.setJob("清洁工");  vo.setSal(300.0);  vo.setComm(200.0);  vo.setHiredate(**new** Date());  **try** {  TestCase.*assertTrue*(ServiceFactory.*getIEmpServiceInstance*().insert(  vo));  } **catch** (Exception e) {  e.printStackTrace();  }  }  @Test  **public** **void** testUpdate() {  Emp vo = **new** Emp();  vo.setEmpno(8888);  vo.setEname("张三");  vo.setJob("清洁工");  vo.setSal(1000.0);  vo.setComm(600.0);  vo.setHiredate(**new** Date());  **try** {  TestCase.*assertTrue*(ServiceFactory.*getIEmpServiceInstance*().update(  vo));  } **catch** (Exception e) {  e.printStackTrace();  }  }  @Test  **public** **void** testDelete() {  **try** {  TestCase.*assertTrue*(ServiceFactory.*getIEmpServiceInstance*()  .delete(8888));  } **catch** (Exception e) {  e.printStackTrace();  }  }  @Test  **public** **void** testGet() {  **try** {  Emp vo = ServiceFactory.*getIEmpServiceInstance*().get(7369);  TestCase.*assertNotNull*(vo);  System.*out*.println(vo.getEname());  } **catch** (Exception e) {  e.printStackTrace();  }  }  @Test  **public** **void** testList() {  **try** {  Map<String, Object> map = ServiceFactory.*getIEmpServiceInstance*()  .list("ename", "", 1, 5);  TestCase.*assertNotNull*(map);  System.*out*.println("总记录数：" + map.get("empCount"));  @SuppressWarnings("unchecked")  List<Emp> all = (List<Emp>) map.get("allEmps") ;  Iterator<Emp> iter = all.iterator() ;  **while** (iter.hasNext()) {  Emp emp = iter.next() ;  System.*out*.println(emp.getEname() + "，" + emp.getJob());  }  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

**3.7、完成dept操作**

完成了Emp操作之后，下面继续完成dept表的操作，那么对dept的操作现在有如下的要求：

· 【业务层】增加一个新部门；

|- 〖数据层〗判断增加的部门编号是否存在；

|- 〖数据层〗增加部门数据；

· 【业务层】修改一个部门信息；

|- 〖数据层〗调用修改操作；

· 【业务层】删除一个部门信息；

|- 〖数据层〗调用删除操作；

· 【业务层】根据部门编号取得一个部门的信息；、

|- 〖数据层〗调用根据id查询的操作；

· 【业务层】查询全部的部门信息；

|- 〖数据层〗查询全部。

1、 开发DatabaseConnection.java类，已经开发完成；

2、 开发Dept的vo类：

|  |
| --- |
| **package** cn.mldn.oracle.vo;  **import** java.io.Serializable;  @SuppressWarnings("serial")  **public** **class** Dept **implements** Serializable {  **private** Integer deptno ;  **private** String dname ;  **private** String loc;  } |

3、 开发IDeptDAO接口：

|  |
| --- |
| **package** cn.mldn.oracle.dao;  **import** java.util.List;  **import** cn.mldn.oracle.vo.Dept;  **public** **interface** IDeptDAO {  **public** **boolean** doCreate(Dept vo) **throws** Exception;  **public** **boolean** doUpdate(Dept vo) **throws** Exception;  **public** **boolean** doRemove(**int** id) **throws** Exception;  **public** Dept findById(**int** id) **throws** Exception;  **public** List<Dept> findAll() **throws** Exception;  } |

这个时候所编写的接口，第一反应发现除了参数不一样之外，和IEmpDAO一样，而且就算现在有几百张表，对于一些基本操作：插入数据、更新全部、删除数据、根据ID查询数据、查询全部数据、带分页查询、统计分页的数据量。没有必要重复编写，各个表不同的只有两块：VO类、ID类型。所以现在对于接口就必须重新设计了。

**范例：**定义一个公共的IDAO接口

|  |
| --- |
| **package** cn.mldn.oracle.dao;  **import** java.util.List;  /\*\*  \* 公共的DAO操作接口  \* **@author** MLDN  \* **@param** <K>要操作的数据表的主键类型；  \* **@param** <V>要操作的VO类型  \*/  **public** **interface** IDAO<K, V> {  /\*\*  \* 执行数据的增加操作  \* **@param** vo 包含所要增加的数据的VO对象  \* **@return** 如果增加数据成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doCreate(V vo) **throws** Exception ;  /\*\*  \* 执行数据的更新操作  \* **@param** vo 包含了新数据的VO对象  \* **@return** 如果修改成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doUpdate(V vo) **throws** Exception ;  /\*\*  \* 删除一个雇员的信息  \* **@param** id 要删除的雇员编号  \* **@return** 如果删除成功返回true，否则返回false  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **boolean** doRemove(K id) **throws** Exception ;  /\*\*  \* 根据雇员编号查询一个雇员的完整信息  \* **@param** id 要查询的雇员编号  \* **@return** 如果没有指定的雇员编号，返回值为null，<br>  \* 如果有指定的雇员信息，则将所有的雇员信息包装到Emp实例化对象之中返回。  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** V findById(K id) **throws** Exception ;  /\*\*  \* 查询全部的雇员信息  \* **@return** 多个雇员信息使用List返回，如果List集合的size()长度为0，则表示没有数据返回  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** List<V> findAll() **throws** Exception ;  /\*\*  \* 分页显示所有雇员的信息，同时可以完成模糊查询  \* **@param** column 要模糊查询的字段名称  \* **@param** keyWord 要模糊查询的数据，如果为空字符串（isEmpty()判断为true，表示空字符串），则表示查询全部  \* **@param** currentPage 当前所在的页  \* **@param** lineSize 没页显示的记录长度  \* **@return** 多个雇员信息使用List返回，如果List集合的size()长度为0，则表示没有数据返回  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** List<V> findAll(String column, String keyWord, **int** currentPage,  **int** lineSize) **throws** Exception;  /\*\*  \* 统计模糊查询的数据结果，使用COUNT()函数进行统计  \* **@param** column 要模糊查询的字段名称  \* **@param** keyWord 要模糊查询的数据，如果为空字符串（isEmpty()判断为true，表示空字符串），则表示查询全部  \* **@return** 会根据数据量的多少返回数据的长度，如果没有数据返回0  \* **@throws** Exception 操作之中出现了异常，返回给被调用处执行处理  \*/  **public** **int** getAllCount(String column, String keyWord) **throws** Exception;  } |

而每一张数据表，除了以上的基本功能之外，还会包括一些自己的独特功能，所以可以在子接口中完成。

**范例：**定义IDeptDAO接口

|  |
| --- |
| **package** cn.mldn.oracle.dao;  **import** cn.mldn.oracle.vo.Dept;  **public** **interface** IDeptDAO **extends** IDAO<Integer, Dept> {  } |

4、 开发DAO接口的实现类；

|  |
| --- |
| **package** cn.mldn.oracle.dao.impl;  **import** java.sql.Connection;  **import** java.sql.PreparedStatement;  **import** java.sql.ResultSet;  **import** java.util.ArrayList;  **import** java.util.List;  **import** cn.mldn.oracle.dao.IDeptDAO;  **import** cn.mldn.oracle.vo.Dept;  **public** **class** DeptDAOImpl **implements** IDeptDAO {  **private** Connection conn ;  **private** PreparedStatement pstmt ;  **public** DeptDAOImpl(Connection conn) {  **this**.conn = conn ;  }  @Override  **public** **boolean** doCreate(Dept vo) **throws** Exception {  String sql = "INSERT INTO dept(deptno,dname,loc) VALUES (?,?,?)" ;  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  **this**.pstmt.setInt(1, vo.getDeptno()) ;  **this**.pstmt.setString(2, vo.getDname()) ;  **this**.pstmt.setString(3, vo.getLoc()) ;  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true** ;  }  **return** **false**;  }  @Override  **public** **boolean** doUpdate(Dept vo) **throws** Exception {  String sql = "UPDATE dept SET dname=?,loc=? WHERE deptno=?" ;  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  **this**.pstmt.setString(1, vo.getDname()) ;  **this**.pstmt.setString(2, vo.getLoc()) ;  **this**.pstmt.setInt(3, vo.getDeptno()) ;  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true** ;  }  **return** **false**;  }  @Override  **public** **boolean** doRemove(Integer id) **throws** Exception {  String sql = "DELETE FROM dept WHERE deptno=?" ;  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  **this**.pstmt.setInt(1, id);  **if** (**this**.pstmt.executeUpdate() > 0) {  **return** **true** ;  }  **return** **false**;  }  @Override  **public** Dept findById(Integer id) **throws** Exception {  Dept dept = **null** ;  String sql = "SELECT deptno,dname,loc FROM dept WHERE deptno=?" ;  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  **this**.pstmt.setInt(1, id);  ResultSet rs = **this**.pstmt.executeQuery() ;  **if** (rs.next()) {  dept = **new** Dept() ;  dept.setDeptno(rs.getInt(1)) ;  dept.setDname(rs.getString(2)) ;  dept.setLoc(rs.getString(3)) ;  }  **return** dept;  }  @Override  **public** List<Dept> findAll() **throws** Exception {  List<Dept> all = **new** ArrayList<>() ;  String sql = "SELECT deptno,dname,loc FROM dept" ;  **this**.pstmt = **this**.conn.prepareStatement(sql) ;  ResultSet rs = **this**.pstmt.executeQuery() ;  **while** (rs.next()) {  Dept dept = **new** Dept() ;  dept.setDeptno(rs.getInt(1)) ;  dept.setDname(rs.getString(2)) ;  dept.setLoc(rs.getString(3)) ;  all.add(dept) ;  }  **return** all;  }  @Override  **public** List<Dept> findAll(String column, String keyWord, **int** currentPage,  **int** lineSize) **throws** Exception {  **throw** **new** Exception("此方法未实现！") ;  }  @Override  **public** **int** getAllCount(String column, String keyWord) **throws** Exception {  **throw** **new** Exception("此方法未实现！") ;  }  } |

5、 在DAOFactory类之中，增加新的方法，取得IDeptDAO接口实现类对象；

6、 开发服务层接口；

|  |
| --- |
| **package** cn.mldn.oracle.service;  **import** java.util.List;  **import** cn.mldn.oracle.vo.Dept;  **public** **interface** IDeptService {  **public** **boolean** insert(Dept vo) **throws** Exception;  **public** **boolean** update(Dept vo) **throws** Exception;  **public** **boolean** delete(**int** id) **throws** Exception;  **public** Dept get(**int** id) **throws** Exception;  **public** List<Dept> list() **throws** Exception;  } |

7、 开发服务层接口实现类；

8、 在ServiceFactory接口之中增加新的方法，可以取得IDeptService接口对象；

**3.8、使用mgr字段操作**

在emp表中的mgr字段，表示的是每一个雇员的领导，如果现在要想加入上这种操作关系，需要做如下的几步。

1、 在Emp类之中表示出领导的关系，增加一个mgr属性；

|  |
| --- |
| **private** Emp mgr ;  **public** **void** setMgr(Emp mgr) {  **this**.mgr = mgr;  }  **public** Emp getMgr() {  **return** mgr;  } |

2、 修改DAO实现类，因为现在操作数据的时候要考虑mgr字段了

**3.9、使用deptno字段操作**

Emp表中的deptno字段是一个每一个雇员所属的部门编号，所以在这之中就会发生如下的两类关系：

· 关系一：一个雇员属于一个部门；

· 关系二：一个部门有多个雇员。

1、 首先在Emp类之中增加一个Dept的操作

|  |
| --- |
| **private** Dept dept ;  **public** **void** setDept(Dept dept) {  **this**.dept = dept;  }  **public** Dept getDept() {  **return** dept;  } |

2、 修改EmpDAOImpl的实现子类上；

**4、总结**

1、 程序的分层操作一定要掌握；

2、 基本操作，要求对于单表的CRUD灵活编写，半小时写完一个；

3、 对于表之间的关系，必须会，能都会最好。

**5、作业**

1、 emp表的基本字段CRUD、分页，5遍；

2、 dept表的基本字段CRUD，5遍；

以上在下周一的时候交。

3、 挑字段写，以下在元旦之后交。

以下的字段必须全写，不是挑字段了。

作业要求录视频编写，每一个都要求有视频，有编写完的代码；

写的时候，包名称写上你自己姓名的拼音。

cn.mldn.fuyunsong.dao

所有的视频凡是以不能录象为理由的，或者是没有录象，后者是丢失的，或者是XX原因的一概算没有完成。

至少有2 ~ 3遍是可以在半个小时内写完的。