 INSERT、DELETE、UPDATE三种SQL语句是数据库技术的三大基本语句。在通常的web开发中对它的处理可以说是无处不在。如果简单的都用手工来构造这些SQL语句的话，一方面给我们的开发带来很大的工作量，另一方面系统灵活性受到很大的限制。那么能不能基于某种规则让系统自动从页面表单中取出元素构造出SQL语句呢? 首先让我们看看一般INSERT、DELETE、UPDATE三种语句的基本形式： INSERT INTO table_name (col_1,col_2,col_3,) VALUES (value_1,value_2,value_3 …) DELETE FROM table_name WHERE col_n=value_n UPDATE table_name SET col_1=value_1,col_2=value_2,col_3=value_3 WHERE col_x=value_x 我们知道，借用J2ee中的request.getParameterNames()方法可以读到表单中的所有元素的名称，有了元素名称借用request.getParameter(elementName)方法可以获取该元素的值。假设在开发中我们让页面元素的名称和底层数据库表的字段名一致。那么在这三种语句中col_n 和 value_n对我们来说就不是未知的，未知的数据就剩下了table_name,col_x和value_x 。现在如果我们写一个方法，传入request对象，再把table_name,col_x,value_x作为参数传入方法，那么我们可以轻松的自动构造SQL语句了。 但这样做还是有欠灵活，因为一方面每一次使用该方法我们都得人工的设置 table_name,col_x和value_x；另一方面别忘了sql语句中对于字符串的字段需要加单引号和替换字符串中间的单引

号，而整型、浮点型、系统函数(如now(),to_date()等数据库函数)等不需要做单引号的处理，这些如果没有好的解决的话，我们的方法将受到非常大的限制。要达到再进一步分离最好的办法就是在表单元素命名上面做文章，我们可以自己定义一套元素命名规则，对不同规则命名的元素做不同的处理--设我们定义元素命名规格如下： 1. table_name,col_x,value_x这类元素，为公共元素。我们规定这类元素名以c_k开头（c=common），我们限制table_name的元素名为c_table,col_x=value_x定义到一起，元素名定为c_where. 当然我们别忘了我们还需要一个元素表示构造什么样（INSERT、DELETE、UPDATE）的SQL语句。我们给这个元素命名c_genre，它的值被限制在INSERT、DELETE、UPDATE这三者之中 。 2. 对于表单中对应数据库字符串类型的元素，在SQL构造中需要做单引号的处理。这类元素我们暂且称他们为字符串型元素。字符串型元素我们规定其名为s_ 数据库表字段名 （s=String）。 3. 对于不需要做但引号处理的元素（如integer型、float型、数据库系统函数--如now()，to_date()等等）。我们暂且简单的统称这类元素为整型元素。对于整型元素我们限制其命名规则为i_ 数据库表字段名(i=Integer)。 基于上面的规格我们可以非常轻松写一个Javabean。代码如下： /*** @version: 1.1* @Time: 2005.03.02*/package com.river.page .import java.util.*.import javax.servlet.http.HttpServletRequest.public class PageUtil { private HttpServletRequest request = null . public PageUtil(){} public void init(HttpServletRequest _request){this.request = _request . } public void clear(){if(this.request != null){this.request = null . }}public

String get(String elementName){ if(request == null || request.getParameter(elementName) == null){return "". }else{return request.getParameter(elementName). }}public String get(HttpServletRequest _request,String elementName){ init(_request). return get(elementName).}public String getSQL(HttpServletRequest _request){ init(_request). return getSQL().}public String getSQL(){ String sqlstr = "". String c_table = get("c_table"). String c_genre = get("c_genre"). String c_where = get("c_where"). if(c_genre == null || c_genre.equals("")){return "the action is null/empty". } if(c_table == null || c_table.equals("")){return "unknow table/empty" . } if(c_genre.equalsIgnoreCase("INSERT")){java.util.Enumeration arg_names = request.getParameterNames().String colstr = "",valstr = "".String arg_name,pre_name,end_name .while(arg_names.hasMoreElements()){ arg_name = String.valueOf(arg_names.nextElement()). if(arg_name.length() continue. } pre_name = arg_name.substring(0,2). end_name = arg_name.substring(2). if(pre_name.equalsIgnoreCase("i_")){colstr = colstr "," end_name.if(get(arg_name).equals("")){ valstr = valstr ",NULL".}else{ valstr = valstr "," String.valueOf(get(arg_name)).} }else if(pre_name.equalsIgnoreCase("s_")){colstr = colstr "," end_name.if(get(arg_name).equals("")){ valstr = valstr ",NULL".}else{ valstr = valstr "," get(arg_name).replaceAll("","") "". }} } if(!colstr.equals("")){colstr = colstr.substring(1).valstr = valstr.substring(1). } sqlstr = "INSERT INTO " c_table " (" colstr ") VALUES (" valstr ")". return sqlstr.}else

if(c_genre.equalsIgnoreCase("UPDATE")){ java.util.Enumeration arg_names = request.getParameterNames(). String colstr = "". String arg_name,pre_name,end_name . while(arg_names.hasMoreElements()){arg_name = String.valueOf(arg_names.nextElement()).trim().if(arg_name.length() continue.}pre_name = arg_name.substring(0,2).end_name = arg_name.substring(2).if(pre_name.equalsIgnoreCase("i_")){ if(get(arg_name).equals("")){colstr = "," end_name "=NULL". }else{colstr = "," end_name "=" get(arg_name).} }else if(pre_name.equalsIgnoreCase("s_")){if(get(arg_name).equals("")){ colstr = "," end_name "=" get(arg_name).}else{ colstr = "," end_name "=" get(arg_name).replaceAll("","") "".} }}if(!colstr.equals("")){ colstr = colstr.substring(1).}sqlstr = "UPDATE " c_table " SET " colstr.if(!c_where.equals("")){ sqlstr = " WHERE " c_where.}return sqlstr.}else if(c_genre.equalsIgnoreCase("DELETE")){sqlstr = "DELETE FROM " c_table.if(c_where != null amp. !c_where.equals("")){ sqlstr = " WHERE " c_where.}}else{ com.river.debug.Debug.show("unknow action type : " c_genre). return null.}return sqlstr.}public String toString(){ return "version 1.0, date 2005.03.02, author river".}} 这样我们就可以根据页面元素的命名来指导SQL语句的生成。这样做有很多的明显的好处： 1、减少编码工作，对于元素很多表单，用不着我们去写一大堆的代码，不用去担心哪个元素落下了，元素名有没有些错，单引号有没有处理。 2、通用、稳定、易于维护，javabean固有的优点，就不用太多的说明了。 3、分离表层

的表单内容与逻辑层SQL语句的构造。设想一下，如果我们数据库表结构有调整时，那么我们只要修改一下表单就好了，根本就不用理 原来写好的逻辑处理。附带着再说一句，设想如果我们再写一个类自动执行SQL，那么对于一些基本的增、删、改操作都可以映射到同一个action里面来处理，且不是很爽？ 当然，这样做的缺点也是有的。那就是有一定的性能损耗。特别是碰到表单元素非常多时。但是对于那些不是很“ 苛刻 ”的项目这点损耗是值得的。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com