

Weaver Software

中国上海耀华支路泛微软件大厦

邮政编码：200122

电话：+86 21 50942228

传真：+86 21 50942278

e-cology开发代码性能与安全技术规范

**仅限阅读 请勿传播**

当您阅读本方案时，即表示您同意不传播本方案的所有内容

**版权说明**

本手册以及相关的计算机软件程序仅用于为泛微协同管理平台最终用户提供信息，上海泛微网络科技股份有限公司有权。本手册是上海泛微网络科技股份有限公司的专有信息，并受中华人民共和国版权法和国际公约的保护。

未得到上海泛微网络科技股份有限公司明确的书面许可，不得为任何目的、以任何形式或手段辅助或传播本手册的任何部分内容。

本手册内容若有变动，恕不另行通知。本手册中例子中所引用的公司、人名和数据若非特殊说明，均属虚构。

本手册以及本手册所提及的任何产品的使用均受适应的最终用户许可协议限制。本手册由上海泛微网络科技股份有限公司制作。本手册中提及的所有商标、商标名称、服务标志及徽标均归其各自公司所有。

目录

[第1章 性能规范 5](#_Toc441507467)

[一、 RecordSet 5](#_Toc441507468)

[1.1.1、使用示例 5](#_Toc441507469)

[1.1.2、注意事项 6](#_Toc441507470)

[二、 禁止 RecordSet 结果集长期共享使用 8](#_Toc441507471)

[三、 RecordSetTrans 8](#_Toc441507472)

[1.3.1、使用示例 8](#_Toc441507473)

[1.3.2、注意事项 9](#_Toc441507474)

[四、 ConnStatement 9](#_Toc441507475)

[1.4.1、使用示例 10](#_Toc441507476)

[1.4.2、注意事项 11](#_Toc441507477)

[五、 如何使用缓存提高系统效率 11](#_Toc441507478)

[六、 System.out 20](#_Toc441507479)

[七、 while循环 20](#_Toc441507480)

[八、 添加必要的数据库索引 21](#_Toc441507481)

[九、 SQL语句禁止用子查询获取列 21](#_Toc441507482)

[十、 SQL语句避免频繁调用 22](#_Toc441507483)

[十一、 String与StringBugffer 22](#_Toc441507484)

[十二、 集合 22](#_Toc441507485)

[十三、 对象 23](#_Toc441507486)

[十四、 同步 23](#_Toc441507487)

[第2章 安全规范 23](#_Toc441507488)

[一、 禁止直接通过参数传递SQL 23](#_Toc441507489)

[二、 使用RecordSet和RecordSetTrans类进行SQL调用规范 24](#_Toc441507490)

1. 性能规范

## RecordSet

RecordSet是数据库操作类，无事务控制，使用完毕会自动释放连接；

|  |  |
| --- | --- |
| **方法** | **描述** |
| executeSql(String sql) | 执行sql脚本 |
|  |  |
|  |  |
|  |  |
|  |  |

### 1.1.1、使用示例

|  |
| --- |
| 简单的SQL语句可以使用该类简化操作，如select语句等。  示例：  查找员工姓名：  RecordSet rs = new RecordSet();  rs.executeSql("select lastname from hrmresource where id=10");  if(rs.next())lastname = rs.getString("lastname");  查找某员工所有的文档：  rs.executeSql("select \* from docdetail where doccreaterid=10");  while(rs.next()){  lastname = rs.getString("docsubject");  } |

### 1.1.2、注意事项

* 该对象执行sql后，会保留对象集，后续仍然可以使用该对象提取数据；
* 该对象能够复用，用完之后其他地方也能继续使用。尤其当使用循环的时候，不要在循环里面创建新的对象，避免垃圾回收频繁，引起性能问题。

|  |
| --- |
| 如，千万不要出现如下代码：  for(int i=0;i<10;i++){  rs = new RecordSet();  ...  }  这种情况可以使用如下方式：  rs = new RecordSet();  for(int i=0;i<10;i++){ ...} |

* 如果循环内有另外的SQL语句需要执行，那么需要使用另外的对象。

|  |
| --- |
| 千万不要出现以下代码：  RecordSet rs = new RecordSet();  rs.executeSql("select \* from hrmresource");  while(rs.next()){  rs.executeSql("select \* from docdetail where doccreaterid="+rs.getString("id"));  //此时，rs的值已经被改变，外层循环就是错误的！  }  这种情况应该使用以下方式：  RecordSet rs = new RecordSet();  RecordSet rs1 = new RecordSet();  rs.executeSql("select \* from hrmresource");  while(rs.next()){  rs1.executeSql("select \* from docdetail where doccreaterid="+rs.getString("id"));  } |

* 这个类不是线程安全的！ **如果多线程访问会产生连接泄漏！**所以 只能在 java方法里面 new 出来使用，不能作为公用变量，比如 类变量或者静态变量;

（**一种特殊情况： servlet 的类变量中绝对不允许定义 RecordSet、RecordSetTrans、ConnStatement，因为 servlet 不是线程安全的**）

* 这个类（以及RecordSetTrans、ConnStatement） 会存储select的结果集，直到被下次select的结果集覆盖，因此一定要控制select出的内容大小， 如果select的字段很多，其中还有 varchar、text等长文本字段， 记录数 超过1w 就需要特别小心了，如果是几百万，一定会导致系统宕机了， 如果只是select 一列（id） 那么也不要超过 1千w级别 否则也会出问题。

## 禁止 RecordSet 结果集长期共享使用

**原因**：RecordSet是线程不安全的，**必须仅仅在方法里面 new出来，同时在方法里面使用**，禁止定义为类级别或者静态变量，做长期的共享使用

（RecordSet、RecordSetTrans、ConnStatement 都有类似要求）

## RecordSetTrans

RecordSetTrans是数据库操作类，有事物控制

|  |  |
| --- | --- |
| **方法** | **描述** |
| executeSql(String sql) | 执行sql脚本 |
|  |  |
|  |  |
|  |  |
|  |  |

### 1.3.1、使用示例

需要使用事物控制的操作需要使用该类来处理，如需要同步更新多个表的数据等。

示例：

删除某个员工，并且删除其创建的文档：

|  |
| --- |
| RecordSetTrans rst = new RecordSetTrans();  try{  rst.setAutoCommit(false);  rst.executeSql("delete from docdetail where id=10");  rst.executeSql("delete from hrmresource where id=10");  rst.commit();  }catch(Exception e){  rst.rollback();  } |

### 1.3.2、注意事项

* 事物控制容易引起死锁问题，所以如非必要，尽量使用RecordSet或者Connstatement代替;
* 如果事物中需要额外的查询操作，请另外新建一个RecordSet对象处理。

## ConnStatement

ConnStatement：无事务控制，需手动关闭连接

|  |  |
| --- | --- |
| **方法** | **描述** |
| executeSql(String sql) | 执行sql脚本 |
|  |  |
|  |  |
|  |  |
|  |  |

### 1.4.1、使用示例

当一条sql语句要被循环多次执行，且只是参数值不一样时，可以使用该类操作，如循环更新多条记录的相同字段。

示例：

更新指定书签的顺序：

|  |
| --- |
| ConnStatement stat = new ConnStatement();  RecordSet rs = new RecordSet();  rs.executeSql("select \* from workflow\_showdoc");  try{  String sql = "update workflow\_showdoc set fieldorder = ? where id=?";  stat.setStementSql(sql);  int i=0;  while(rs.next()){  stat.setInt(1,i);  stat.setString(rs.getString("id"));  stat.executeQuery();  }  }catch(Exception e){  }finally{  stat.close();  } |

### 1.4.2、注意事项

* 使用该类需要特别注意的是，使用完后一定要在finally中手动关闭连接，不然会导致连接泄露，严重影响系统的性能及稳定性！
* 这个类不是线程安全的！ **如果多线程访问会产生连接泄漏！**所以 只能在 java方法里面 new 出来使用，不能作为公用变量，比如 **类变量或者静态变量**;

## 如何使用缓存提高系统效率

对于一些常用的信息，比如人员的名称，如果每次访问的时候都查询数据库来获取，将极大的影响系统的性能。我们可以将这些信息放到系统的缓存中，在用到的时候，从系统缓存中获取。（使用缓存是典型的用空间换时间的系统优化方式）

系统提供了 weaver.general.StaticObj 类来保存这些信息。weaver.general.StaticObj 类使用 HashTable 来管理缓存信息，其主要的方法如下：

StaticObj将数据库中常用的数据放入缓存中，以加快系统的反应速度。客户程序可以调用getInstance()方法访问本类的唯一实例。

StaticObj有两层缓存，分别为 mainKey - value 和 mainKey - subkey – value，由 mainKey 可以取得第一层的对象，由 mainKey - subkey 可以取得第二层的值，在第二层的值存在的情况下，第一层的值为 Hashtable。

|  |
| --- |
| /\*\*  \* 返回唯一实例.如果是第一次调用此方法,则创建实例  \*  \* @return StaticObj 唯一实例  \*/  public static StaticObj getInstance() |

|  |
| --- |
| /\*\*  \* 返回唯一实例.如果是第一次调用此方法,则创建实例  \*  \* @return StaticObj 唯一实例  \*/  public static StaticObj getInstance() |

|  |
| --- |
| /\*\*  \* 获取 mainKey 对应的第一层对象值  \*  \* @param key 主键名称  \* @return Object 对于的第一层对象值  \*/  public Object getObject(String key) |

|  |
| --- |
| /\*\*  \* 设置 mainKey 对应的第一层对象值  \*  \* @param key 主键名称  \* @param obj mainKey 对应的第一层对象值  \*/  public void putObject(String key, Object obj) |

|  |
| --- |
| /\*\*  \* 从缓存中清除 mainKey 对应的第一层对象值  \*  \* @param key 主键名称  \*/  public void removeObject(String key) |

|  |
| --- |
| /\*\*  \* 获取 mainKey - subkey 对应的第二层对象值  \*  \* @param HashKey 主键名称  \* @param RecKey 子主键名称  \* @return Object 对于的第二层对象值  \*/  public Object getRecordFromObj (String HashKey, String RecKey) |

|  |
| --- |
| /\*\*  \* 设置 mainKey - subkey 对应的第二层对象值  \*  \* @param HashKey 主键名称  \* @param RecKey 子主键名称  \* @param obj mainKey - subkey对于的第二层对象值  \*/  public void putRecordToObj (String HashKey, String RecKey, Object obj) |

|  |
| --- |
| /\*\*  \* 从缓存中清除 mainKey - subkey 对应的第二层对象值  \*  \* @param HashKey 主键名称  \* @param RecKey 子主键名称  \*/  public void removeRecordFromObj (String HashKey, String RecKey) |

|  |
| --- |
| /\*\*  \* 从缓存中清除 所有对象值  \*  \*/  public void clearRecord() |

运用这些方法，我们可以构造所需要的缓存类，使用这些缓存类来对其它模块和功能提供缓存信息的接口，这些缓存类在系统中的命名为 XXXComInfo，其中XXX为相应缓存信息的名称。参考如下的人力资源缓存类：weaver.hrm.resource.ResourceComInfo。

ResourceComInfo 类主要提供人力资源信息中常用的一些信息的缓存，比如人员名称，所在部门。对于这些信息，都需要一个键值。常见的键值是这些信息在数据库表中的Primary Key，比如人力资源ID，其它模块和功能在关联人力资源信息的时候，只记录人力资源的ID，当需要获取人力资源的其它信息，如人员名称，所在部门的时候，从ResourceComInfo类提供的统一接口中获取。

weaver.hrm.resource.ResourceComInfo 的代码：细节参考 VSS 里的代码

public class ResourceComInfo extends BaseBean {

|  |
| --- |
| /\* 1 \*/  private ArrayList ids = null; // 保存人力资源键值 ID 队列  private ArrayList loginids = null; // 保存人力资源登录名 队列  …………  …………  //用来同步初始化过程，防止多个线程同时进行初始化，导致的内存溢出  **private** **static** Object ***syncObj*** = **new** Object();    /\*\*  \* 人力资源缓存信息接口类构造方法  \* 构造方法中将获取公共缓存类的唯一实例，并调用getResourceInfo 方法获取缓存信息，  \* 同时赋值人力资源记录的数量  \*  \*/  public ResourceComInfo() throws Exception{  staticobj = StaticObj.getInstance();  getResourceInfo() ;  array\_size = ids.size();  }  /\*\*  \* 获取人力资源缓存信息方法  \* 检查是否有人力资源缓存信息，如果没有，将调用setResourceInfo 方法从数据库中获取人力资源信息并放入缓存中  \* 将缓存中的信息赋予相应的队列  \*  \*/  private void getResourceInfo() throws Exception{  **synchronized**(*syncObj*){ |

|  |
| --- |
| /\* 2 \*/  if(staticobj.getObject("ResourceInfo") == null)  setResourceInfo();  // 将缓存中的信息赋予相应的队列  ids = (ArrayList)(staticobj.getRecordFromObj("ResourceInfo", "ids"));  loginids = (ArrayList)(staticobj.getRecordFromObj("ResourceInfo", "loginids"));  …………  …………  }  }  /\*\*  \* 从数据库获取人力资源信息并放入缓存方法  \* 从数据库获取所需要的人力资源信息 （该类所提供的人力资源信息）  \* 将获取的信息放入缓存中  \*  \*/  private void setResourceInfo() throws Exception{  // 生成队列实例  ids = new **FastArrayList**();// 请看 注释6 的解释 标注001  loginids = new ArrayList();  lastnames = new ArrayList();  …………  …………    // 查询数据库并赋值队列  RecordSet rt = new RecordSet() ;  rt.executeProc("HrmResource\_SelectAll","") ;  while(rt.next()){  ids.add(Util.null2String(rt.getString("id")));  loginids.add(Util.null2String(rt.getString("loginid")));  …………  …………  } |

|  |
| --- |
| /\* 3 \*/  // 将获取的信息放入缓存中, 缓存中的一级键值为 "ResourceInfo" ， 二级键值为人力资源对应的队列名称  staticobj.putRecordToObj("ResourceInfo", "ids", ids);  staticobj.putRecordToObj("ResourceInfo", "loginids", loginids);  …………  …………  }  …………  …………  } |

以上代码需要注意的几个地方：

注释1：这里所定义的队列，用于依次存放人力资源的所需信息。所需信息是根据我们的需要来确定的，但是必须明确一点，只有那些经常用到的信息采放到缓存中，否则将会造成缓存的臃余。

注释2：这里的指针和方法用于一些程序遍历人力资源的所有信息。比如如下在jsp页面中的应用：

|  |
| --- |
| <select name="resourceinfo">  <%  while(ResourceComInfo.next()) {  %>  <option value="<%= ResourceComInfo .getResourceid()%>"><%=ResourceComInfo .getLastname ()%></option>  <%  }  %>  </select> |

注释3：缓存中的一级键值设置为相应的类名，根据这个键值来判断缓存中是否有相应的信息，或者清除缓存中的该信息。需要注意的是，缓存中的键值信息必须唯一，不能重复！

注释4：在不是必须的时候，更新人力资源的时候一定要使用更新指定的缓存的方式，而不要把缓存清空，等待下一次使用到该缓存的时候在重新初始化该缓存。

注释5：对数据更新频率不高，使用频率确相对较高数据请尽量使用缓存的方式存取数据，而不是直接从数据库中读取。

**注释6：当前的 XXXComInfo的实现， 根据 id 获取相关列信息的方法 getXXX(id) 的性能并不是 map 级别的性能**， 使用当前的XXXComInfo的实现方式， 这个方法耗时跟数据的多少有关系，数据达到几万以后，性能就会下降，如果再碰到内外嵌套循环，那么性能会有严重问题，请使用的时候多加注意，当前有新的实现方式 来改善这个问题，使用weaver. general.FastArrayList（**搜索：标注001**） 来改善 ArrayList 查找操作的性能，就可以达到 Map 级别的访问性能

## System.out

正式代码中严禁使用 System.out 进行日志的输出，原因是这个语句的性能非常低，一旦调用的非常频繁， 就会成为性能瓶颈。

## while循环

如果系统中有 while(条件) 循环踢， 条件的地方 或者在 循环体内部，一定要有一个退出条件，比如计数到 10000 次后 强制退出

如果没有这个退出条件，一旦代码出现预料外的情况，很容易导致死循环，从而导致系统宕机

例外： 想 RecordSet.next() 这种就不用这样写了，这个类的next方法是非常安全的，以上指的是自己写的逻辑中 比较复杂的 (条件)，比如：

|  |
| --- |
| while（条件1&&条件2&&（条件3||条件4）……）{  ……  } |

## 添加必要的数据库索引

当你的功能中包含sql操作，同时 操作的表中的数据量非常大是 百万计以上，一定要注意添加适当的索引，否则很容易引起系统的性能问题， 如果拿不准，可以把sql 提交给你的上级来共同研究应该怎么加索引，不能仅仅完成功能就完事。

## SQL语句禁止用子查询获取列

比如：

|  |
| --- |
| select id, name, age, **(select flag from table2 where xxxx ) as flag** from table1 |

推荐的做法是：

|  |
| --- |
| select id, name, age, **flagId** from table1 |

然后通过另外的sql 由flagId 获取到 flag

原因，列里的子查询，有时候会扰乱数据库的 执行计划，导致数据库找到的执行计划不是最优的。

## SQL语句避免频繁调用

比如：超过50次/秒调用、在一个while循环里执行sql，而循环次数可能很大

原因：及时单个sql本身执行很快毫秒，但是一段执行次数太多，也会严重影响性能，比如单条sql执行 100ms 那么 50次 就是5秒，这个 操作就会很慢

正确方式：应该通过一条sql语句，把所有的数据都预先取出来（大概也就比100ms多一点而已的时间）

## String与StringBugffer

不要使用如下String初始化方法： String str = new String(“abcdef”);

这将产生两个对象，应当直接赋值： String str = “abcdef”;

在处理可变 String 的时候要尽量使用 StringBuffer 类，StringBuffer 类是构成 String

类的基础。String 类将 StringBuffer 类封装了起来，（以花费更多时间为代价）为开发人员提供了一个安全的接口。当我们在构造字符串的时候，我们应该用 StringBuffer 来实现大部分的工作，当工作完成后将 StringBuffer 对象再转换为需要的 String 对象。比如：如果有一个字符串必须不断地在其后添加许多字符来完成构造，那么我们应该使用 StringBuffer 对象和她的 append() 方法。如果我们用 String 对象代替 StringBuffer 对象的话，将会花费许多不必要的创建和释放对象的 CPU 时间。

## 集合

避免使用Vector和HashTable等旧的集合实现，这些实现的存在仅是为了与旧的系统兼容，而且由于这些实现是同步的，故而在大量操作时会带来不必要的性能损失，使用ArrayList代替Vector，使用HashMap代替HashTable。 若却是需要使用同步集合类，当使用如下方式获得同步集合实例： Map map = Collections.synchronizedMap(new HashMap());

由于数组、ArrayList与Vector之间的性能差异巨大，在能使用数组时不要使用ArrayList，尽量避免使用Vector。

## 对象

* 避免在循环中频繁构建和释放对象。
* 不再使用的对象应及时销毁。
* 如无必要，不要序列化对象。

## 同步

* 在不需要同步操作时避免使用同步操作类，如能使用ArrayList时不要使用Vector。
* 尽量少用同步方法，避免使用太多的 synchronized 关键字。
* 尽量将同步最小化，即将同步作用到最需要的地方，避免大块的同步块或方法等。

1. 安全规范

## 禁止直接通过参数传递SQL

|  |
| --- |
| 错误写法：  <input type="hidden" name="sqlwhere" value="where id in (1,2,3,4)"/>  /workflow/MultiRequestBrowser.jsp?sqlwhere=where id in (1,2,3,4) |

两种方式解决传输sql的问题：

* 针对java固定的sql参数（即无需通过页面上的输入参数动态生成的），可以采用如下方式加密处理：

|  |
| --- |
| <jsp:useBean id="xssUtil" class="weaver.filter.XssUtil">  <input type="hidden" name="sqlwhere" value="<%=xssUtil.put("where id in (1,2,3,4)")%>"/>  /workflow/MultiRequestBrowser.jsp?sqlwhere=<%=xssUtil.put("where id in (1,2,3,4)")%> |

* 针对需要用js动态拼接生成的sql，需要通过传递单独的参数到服务器端进行处理，不能用js拼好sql直接传输到服务器端。

## 使用RecordSet和RecordSetTrans类进行SQL调用规范

目前RecordSet和RecordSetTrans提供的execute和executeSql方法采用SQL拼接方法来执行SQL，这样会产生SQL注入安全威胁。

现在提供以下两个方法来替代execute和executeSql。

|  |  |  |
| --- | --- | --- |
| **方法名** | **参数** | **说明（示例）** |
| executeQuery(String sql,Object… params) | sql：待执行的SQL语句  params：用于替换sql中的?的参数，可以是0个或多个 | 该方法用于查询，RecordSet中执行完毕后会返回true|false，RecordSetTrans执行失败会抛出异常。同时会保留结果集供使用。遍历结果集的方法和executeSql执行后遍历一样。  示例：   * rs.executeQuery(“select id,loginid from hrmresource where lastname like ?”,”%王%”) * rs.executeQuery(“select id,loginid from hrmresource where lastname like ? and seclevel >= ?”,”%王%”,30) |
| executeUpdate(String sql,Object… params) | sql：待执行的SQL语句  params：用于替换sql中的?的参数，可以是0个或多个 | 该方法用于更新记录，RecordSet中执行完毕后会返回true|false，RecordSetTrans执行失败会抛出异常。  示例：   * rs.executeUpdate(“insert into hrmresource(loginid,lastname) values(?,?)”,”wangxin”,”王鑫”) * rs.executeUpdate(“delete from hrmresource where id=?”,id) |

请大家以后写SQL语句时，使用以上两个方法来替代原来的拼接SQL的方式，以避免SQL渗透威胁。