持久化工具：Memory介绍

[1 概述 1](#_Toc418720535)

[1.1 连接、语句和结果集 2](#_Toc418720536)

[1.2 获取结果集 2](#_Toc418720537)

[1.3 连接的管理 3](#_Toc418720538)

[1.4 语句预处理 3](#_Toc418720539)

[2 使用 4](#_Toc418720540)

[2.1 命令与查询 4](#_Toc418720541)

[2.1.1 查询(query) 4](#_Toc418720542)

[2.1.2 命令(update) 4](#_Toc418720543)

[2.2 增删改查(CRUD) 5](#_Toc418720544)

[2.2.1 新增(create) 5](#_Toc418720545)

[2.2.2 查询(read) 5](#_Toc418720546)

[2.2.3 更新(update) 5](#_Toc418720547)

[2.2.4 删除(delete) 6](#_Toc418720548)

[2.3 其它 6](#_Toc418720549)

[2.3.1 分页 6](#_Toc418720550)

[2.3.2 IN语句 6](#_Toc418720551)

[2.3.3 事务 6](#_Toc418720552)

[3 多余的废话 6](#_Toc418720553)

[3.1 为什么不用链式写法？ 6](#_Toc418720554)

[3.2 为什么不用XML或Annotation配置? 7](#_Toc418720555)

[3.3 为什么只用PreparedStatement? 8](#_Toc418720556)

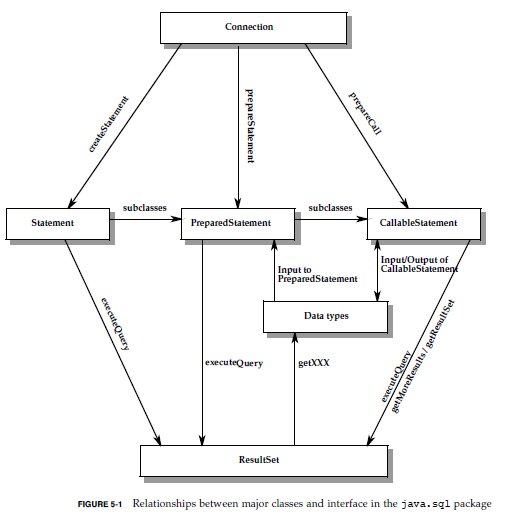
[3.4 能不能把运行时的SQL语句打印出来？ 8](#_Toc418720557)

[3.5 也说ORM 9](#_Toc418720558)

[4 参考文献 9](#_Toc418720559)

# 概述

## 连接、语句和结果集



<http://download.oracle.com/otndocs/jcp/jdbc-4_1-mrel-spec/index.html>

从JDBC的规范上看，其对数据访问层有相当简洁的抽象：1、连接(connection) 2、语句(statement)、3结果集(result set)。我们对数据库做的事情无非：连接数据库，执行语句，拿到结果。

因此，持久化的工具的目的就不言自明了：进一步简化连接的管理、语句的执行、结果集提取等操作。下面从获取结果集、管理连接、语句预处理等3方面逐一阐述工具做了哪些事情。

这里提一句，Memory在设计与实现上，都借鉴了Dbutils，其相对于hibernate, mybatis这些庞然大物，已经是一个极其小巧的工具。但是Memory的类和接口更少（不超过10个），体积更小（只有二十几K），数目和体积都约为dbutils的1/3，却添加了非常实用的功能：

* 将简单的POJO对象直接持久化到数据库中；
* 打印运行时出错的SQL语句，其可以直接拷贝到数据库客户端上进行调试；
* 直截了当的分页查询。

## 获取结果集

获取结果集，就是把ResultSet转换为目标数据结构，这里使用T（泛型）泛指各种数据结构。我们定义一个接口类来表示这件事情：

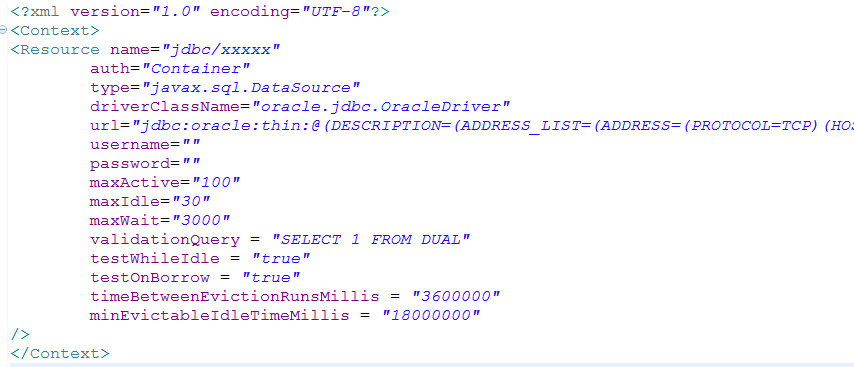
C:\Users\Lenovo\AppData\Roaming\Tencent\Users\345120259\QQ\WinTemp\RichOle\[]1F)N_3EG8]9HJ2AP0`R}7.png

在实际应用中，结果集是某张表的一行或多行数据时，常使用BeanHandler、BeanListHandler或JSONObjectHandler、JSONArrayHandler进行处理，结果集是某一列的一行或多行数据时，使用ColumnHandler、ColumnListHandler进行处理。

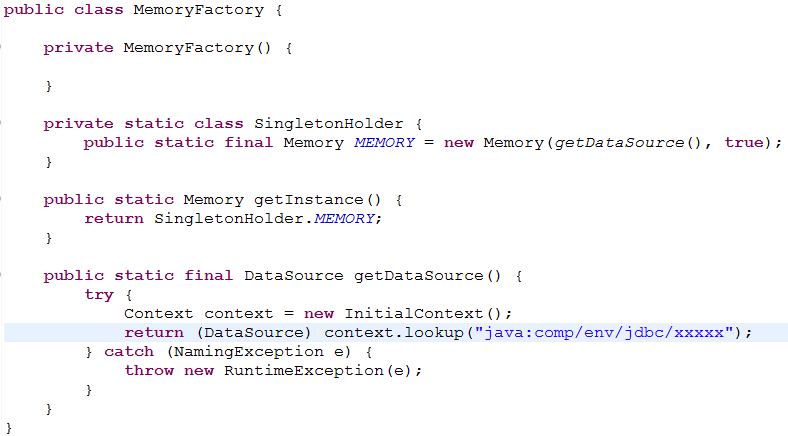
## 连接的管理

将连接的交给外部的数据源(DataSource)进行统一管理。比如使用Tomcat容器自带的数据源。

在Tomcat的context.xml文件配置数据源xxxxx：



在代码中实例化（采用懒加载单例模式）该数据源：



## 语句预处理

与ResultSetHanlder相互呼应，提供了PreparedStatementHanlder类，这个类提供语句（PreparedStatment）一些辅助性的方法，比如生成运行时的SQL语句、调整日期格式、简化分页语句写法等。这个类在应用中不会直接用到。其作用将隐藏在最重要的一个类Memory之中（与这个工具命名相同）。

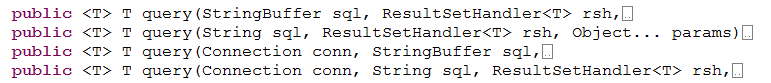
# 使用

上章从结果集提取、连接管理、语句处理等3个角度介绍了这个工具，本章介绍的Memory类就是对3者的集成，分3节描述Memory开放的API。

## 命令与查询

对数据库所有的操作，可分为两类：命令与查询。命令即更新数据，可进一步分为新增、删除与编辑。

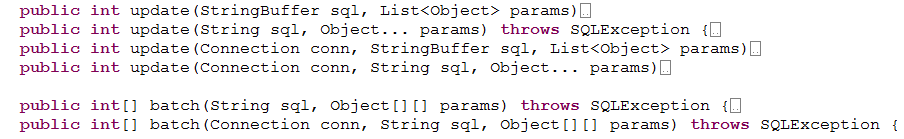
### 查询(query)



从接口定义可以看出，查询(query)方法，返回结果集，参数名也相似，只是数据结构不同而已：StringBuffer和List一组，String和Array（变长参数）一组，没有传递Connection参数，则表明连接在memory内部管理；有传递Connection参数，则表明连接交给外部程序管理。

在这个层面使用API，就是写SQL语句，几乎没有任何限制，唯一的限制就是在使用BeanHandler与BeanListHandler时，**Bean的字段与Table的字段要存在相互匹配，Bean的字段命名风格是驼峰式，Table的字段命名是下划线连接**。

### 命令(update)



相对于查询(query)方法，更新(update)方法，没有结果集处理器(ResultSetHandler)的参数以及结果集转化为的对象。

但更新有批量更新(batch)的方法，提供批量执行sql语句的功能。

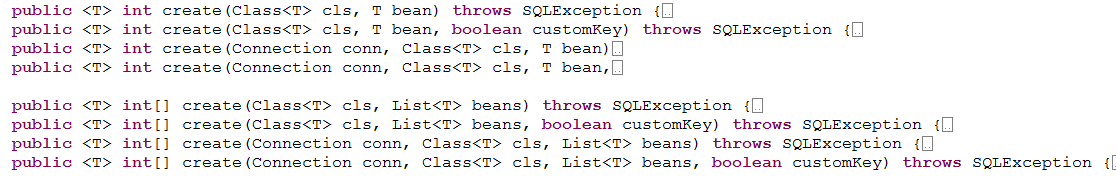
## 增删改查(CRUD)

增删改查，英文缩写为CRUD，这个大家都非常熟悉，使用Create, read, update, delete来做作为接口名称，这样记忆和理解成本最低。

Lifesinger在jQuery 为什么优秀兼谈库与框架的设计一文中，提到：**在类库界，解决了 What，解决了定位问题后，基本上已经决定了生死存亡。** 至于 How，也重要但往往不是关键。https://github.com/lifesinger/lifesinger.github.com/issues/114

本人对此深以为然，所以Memory工具在接口方法名称、类名等的使用上相当节制（数量尽量少），这点也不同于别的持久化工具。

### 新增(create)



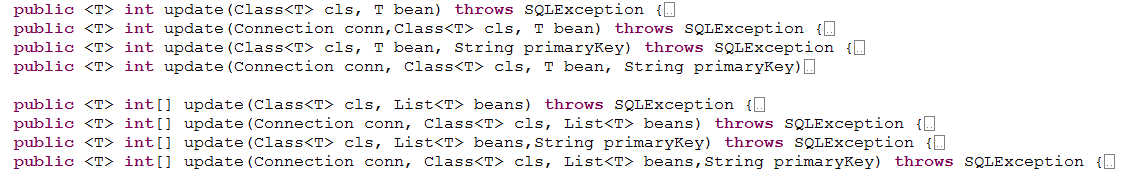
这些接口可持久化新增的一个对象或多个对象时。customkey这个参数表示主键的值是否使用自定的值。如果不是使用自定义的值，则采用序列(oracle)或自增主键(mysql)，此时主键的名称必须是ID。

### 查询(read)



根据主键（主键名必须为ID）读取一条记录，并转化为对象。

### 更新(update)



这些接口可持久化更新的一个对象或多个对象时。primaryKey这个参数指定主键名称，默认是ID。

### 删除(delete)



根据主键（主键名必须为ID）删除一条记录。

## 其它

Memory的API在SQL语句操作层面分为：命令与查询（2.1节），在对象操作层面分为：增删改查（2.2节）。查询有一些常用的辅助性操作，比如分页和IN语句；在对事务有要求的场合，memory提供获取连接的接口，并将连接交给应用自行控制。

### 分页



分页查询几乎是必不可少的，但是oracle的分页查询语句写起来相当复杂（3重嵌套），mysql分页查询虽然简单，但是其参数limit offset, n也不够直观。

分页查询，即在问如果每页pageSize条记录，那么第pageNo页的记录是什么。分页查询接口(pager)封装了oracle和mysql的查询语句，并提供了pageSize和pageNo两个直观的参数。

### IN语句



IN语句在查询时也比较常用，占位符?必须与参数的个数相匹配，手工拼接容易出错；当参数个数是动态变化时，占位符的拼写更是繁琐，因此对IN语句做了一个简单的封装，以保持代码的简洁。

### 事务



可以从memory取出一条连接，然后设置连接为非自动提交，进行事务操作与回滚。

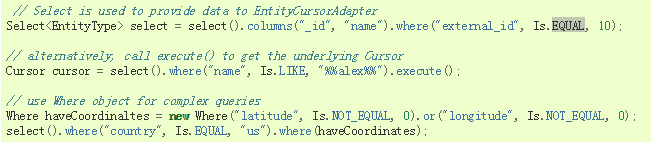
# 多余的废话

## 为什么不用链式写法？

不少持久化的库或框架，喜欢使用链式写法来写SQL语句。但是殊不知链式的写法在Jquery很自然，在SQL中却是生搬硬套，不得其法。**SQL是数据库领域的专用语言(DSL)，用其本来的写法来表达是最自然的**。

比如：

http://droidparts.org/orm.html#many-to-many



又比如：

C:\Users\Lenovo\AppData\Roaming\Tencent\Users\345120259\QQ\WinTemp\RichOle\R4IK$S)$VCHG}8{ZEO7X`LD.png

这些库的设计与Hibernate的Criterion多多少少有些相似，把SQL简单明了的写法改成所谓面向对象的链式写法。关系和对象变得扭曲（Object-Relational Impedance Mismatch），让人几乎看不到SQL本身的简洁和链式写法（builder pattern）的优雅，一举两“失”。

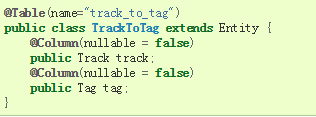
## 为什么不用XML或Annotation配置?

只要我们约定了表名与类名、列名与字段名的命名规则，并严格遵循，何须在再去了解XML和annotation配置的写法，再去写XML和Annotation维护映射关系呢？。少了这些额外的东西，代码的可维护性和可读性是不是也大大提高了呢。

试举一些持久化框架的做法：

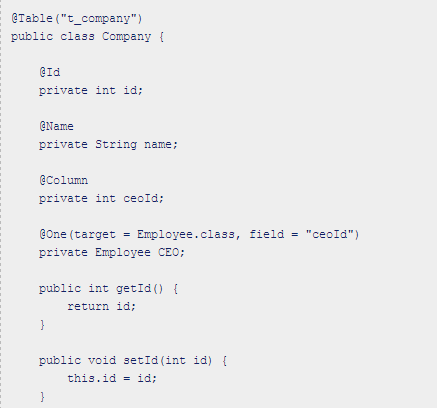
比如：

http://droidparts.org/orm.html#many-to-many



又比如：

<http://www.nutzam.com/core/dao/dynamic_table_name.html>



XML繁琐冗长的配置，比如Ibatis或Hibernate，就不拷贝举例了。

## 为什么只用PreparedStatement?

Statement和CallableStatement只在极少的场景，比如复杂的数据导入导出，可能用到。

但在绝多大多数场景，PreparedStatment相对Statement更高效、更安全，代码的可读性更好；而CallableStatment，是把业务逻辑隐藏在SQL的存储过程，而不是显化在代码之中，理解代码将变得更困难，可读性也不如PreparedStatement。

## 能不能把运行时的SQL语句打印出来？

在开发过程，SQL语句有可能写错，如果能把运行时出错的SQL语句直接打印出来，那对排错非常方便，因为其可以直接拷贝到数据库客户端进行调试。

在<https://www.ibm.com/developerworks/cn/java/j-loggable/>这篇文章中，作者也希望有一种方法，它使我们能够获得查询字符串，并用实际的参数值替换参数占位符，最终他提出了一种解决方案，使用修饰器模式(decorator)扩展PreparedStatement，新增一个有日志功能的LoggableStatment的类。这当然是很不错的解决方案。

在memory工具，没有新增扩展类，只是在PrparedStatementHandler中，提供一个print方法，将SQL语句中的占位符替换为实际的参数，并在发生SQL Exception时，将其打印出来。

## 也说ORM

在开源中国可以搜到数百个ORM框架或类库。可见ORM曾经、也许现在还是，让不少攻城狮和程序猿，趋之若鹜。当然也有人对其反思，有一篇文章《[为什么我说ORM是一种反模式](http://www.nowamagic.net/librarys/veda/detail/2217)》，就提出不同的看法。

文章的中文链接：

<http://www.nowamagic.net/librarys/veda/detail/2217>

文章的英文连接：

https://github.com/brettwooldridge/SansOrm/wiki/ORM-is-an-anti-pattern

ORM，通俗讲，就是把一种问题转化为另一种问题进行解决。但是数据库的问题，比如关联查询、分页、排序，能在OOP中得以完美的解决吗？OOP恐怕心有余而力不足。而这些问题却是关系数据库最擅长的问题域。

把关系数据库擅长解决的问题转化给不擅长处理这类问题的OOP去解决，这不是很糊涂吗？OOP的方法论，应当控制一下自己的野心，专注于自己擅长的领域，比如代码的组织与管理、界面开发的应用等等。

当然ORM也不是一无是处，把一条数据（结果集）自动转化为一个对象，以便于业务代码的处理还是有益处的。但要把所有的关系操作映射为对象的操作（比如外键关系映射为继承），或者反之（比如将继承映射为外键关系），必定是事倍功半、得不偿失。

# 参考文献

<http://commons.apache.org/proper/commons-dbutils/>

<http://www.nowamagic.net/librarys/veda/detail/2217>

https://github.com/brettwooldridge/SansOrm/wiki/ORM-is-an-anti-pattern

<http://segmentfault.com/a/1190000000378827>

<http://www.oschina.net/project/tag/126/orm>

<http://www.nutzam.com/core/dao/annotations.html>

<https://github.com/lifesinger/lifesinger.github.com/issues/114>

https://github.com/brettwooldridge/SansOrm/wiki/ORM-is-an-anti-pattern

<http://www.oschina.net/translate/secrets-of-awesome-javascript-api-design>

<https://github.com/lifesinger/lifesinger.github.com/issues/114>

<http://download.oracle.com/otndocs/jcp/jdbc-4_1-mrel-spec/index.html>

<http://en.wikipedia.org/wiki/Singleton_pattern#Lazy_initialization>