# **对象和数据库的天然阻抗**

<https://www.jdon.com/mda/oo-reltaion2.html>

[板桥里人](http://www.jdon.com/aboutme.htm) http://www.jdon.com 2008/4/14（转载请保留）

在“[面向对象建模与数据库建模两种分析设计方法的比较](https://www.jdon.com/mda/oo_relation.html" \t "https://www.jdon.com/mda/_blank)”一文中我们比较了在对需求分析时两种方法的不同，所谓数据库建模分析，就是项目一开始就根据需求建立数据库模型，如数据表结构和字段等，这种错误现象大量普遍存在我们国内项目实践中，从每年大量招聘启示中就可见一斑：招聘数据库建模人员，招聘Java面向对象程序员。这些说明软件业一边在大量使用Java/.NET/Ruby on Rails这样OO语言同时，还在同时使用与OO体系抵触的围绕关系数据库的分析设计方法。

　　为进一步说明OO和关系数据库是属于两个不同世界观，存在天然矛盾，就象有神论和无神论，就象水与火那样属于截然相反的两种编程知识。正好最近TheServerSide刊登一篇大谈对象数据库ODBMS的文章：When to use an Embedded ODBMS（以下简称ODBMS一文）  
<http://www.theserverside.com/tt/articles/article.tss?l=EmbeddedODBMS>

　　这篇文章不但谈到了OO和关系数据库天然阻抗；谈到了ODBMS和RDBMS区别，也谈到了ODBMS和O/R Mapping如Hibernate的区别，甚至谈到了ODBMS在非C/S如嵌入式方面的优点。这篇文章我看最重要价值就是它用大量篇幅阐述了对象和关系数据库存在的天然矛盾，我这里就将这部分篇幅大意转载这里，可以说是经过我咀嚼后反馈的结果，版权思想还是属于这篇文章，我这里只是用中国人更易于接受方式把这个天然矛盾转述出来：

****对象和关系数据库累赘转换****

　　在一个面向对象系统中，数据存在于对象中，在关系数据库中，数据存在于数据表的记录中。  
在关系数据库世界中，我们必须掌握表结构、表记录等概念，这里面没有任何对象概念，当我们在OO语言中使用关系数据库时，因为在OO世界中是大量对象，所以必须将数据在这两个不同世界之间转换传输。

　　 这就导致了大量垃圾临时对象，增加了应用系统的复杂性。这也是我们很多人使用了Java等这样的OO系统后，反而觉得开发效率并没有OO宣传那么高的原因， 说白了，OO思想没有完全占领应用系统，因为有关系数据库盘踞着数据库这个最后堡垒负隅顽抗。

　　下面看看我们程序如何为了让OO语言和关系数据库捆绑在一起工作，如何弥补两个世界的裂缝，如何在他们之间和稀泥。 为了将对象保存到关系数据库中，我们必须将对象中的数据解散开来，再将它们组装到SQL中，然后执行SQL。  
一个类如下：

|  |
| --- |
| public class Course { 　　public string name; 　　public int courseID; 　　int deptID; 　　public int creditHours; } |

将上述对象保存到关系数据库的SQL语句如下：

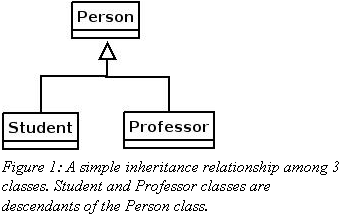
|  |
| --- |
| PreparedStatement insertStatement = connection.prepareStatement( "INSERT INTO Courses (name, courseID, deptID, creditHourse) " + "VALUES (?, ?, ?, ?)"); insertStatement.setString(1, course.name); insertStatement.setInt(2, course.courseID); insertStatement.setInt(3, course.deprtID); insertStatement.setInt(4, course.creditHours); insertStatement.executeUpdate(); |

上述在两个世界之间做的翻译工作与复杂性取决于对象Course这样的复杂性。如果Course字段有十几个，那么这个保存SQL将更复杂。

　　 更重要的是，万一有一个字段发生变化，更改量就很大。我们不但要修改对象，而且还要修改数据表结构，有过数据库编程经验的人知道，如果表中有数据，表结构变动带来的问题可能就象噩梦一样，所以，我们程序员经常以这个需要更改表结构拒绝一些软件的维护和拓展，这其实更是错上加错；在ODBMS一文中提出了对象数据库解决方案：只要更改类结构，其他都有ODBMS搞定；使用ORM如Hibernate也只多一个步骤：更改映射配置。

****继承关系的尴尬实现****

　　对象和关系数据库的不匹配还表现在关系数据库难以实现对象世界中的继承关系，如下图：



　　这是一个典型的对象世界表达客观世界的类图，学生和教授都是人，都具备人一些共性，当然他们之间也有区别，所以我们用上述继承关系来表达这样一个情况。

　　那么如果这样继承关系的类图如何保存到关系数据库中呢？有两种方式：one-class-per-table（一个类对应一个表）和one-object-per-row（一个对象对应一个表行）。

　　在one-class-per-table方式中，Student对象放在Student表中，Professor放在Professor表中。但是Student和Professor实际具有共性Person，这实际上将Person这个共性对象中数据分割成两个表保存，从语义上所：也就是将一个对象分割成两个部分了；而且当你要获取这个对象时，需要两次Select。同样道理增删改查都要两次。

　　如果按照one-object-per-row，将这Student和Professor放到一个表中，就会产生空白字段。Student对象中数据保存到Student对象对应的字段中，那么Professor对象对应的字段就是空的，反之也然。

　　这些都反映了对象和关系数据库天然不匹配，两者根本就无法搭配在一起工作，只有一方作出妥协，大部分情况是对象作出妥协，这样，一个OO语言Java/.NET系统就做成了一个数据库中心系统，根本无法享受OO语言带来快捷方便，可维护性强，由于Java出现比较早，更多程序员将项目失败归结于Java语言本身，转而使用数据库思维去做.NET，去做Ruby On Rails，还是会碰到上面这些问题，没有碰到，只能说明他们系统中就没有对象概念，解决方式很简单很可笑：矛盾双方，消灭一方不久解决矛盾了。

****类的复杂关系实现****

　　下面我们再看看ODBMS一文中列举的类复杂关系如何用关系数据库实现保存:

|  |
| --- |
| public class Department { 　　　　private Professor[] professors; ... } |

　　这个Department类很明确告诉我们这样一个意思，在一个Department中，存在很多Professor，大白话就是：一个部门里有很多教授，这个类就自然准确地表达这个意思，这也体现了使用OO表达需求的自然性和方便性。

　　那么我们下面看看，如果我们使用关系数据库来保存Department这个对象，很显然，这里需要使用关系数据库的表外键，通过表外键来表达Department表和Professor之间1:N关系，关键问题是：这个外键一般是设计在Professor表中，这就造成语义上的误解。

　　对象Department表达的需求含义是：  
　　1. 我是一个部门对象，在我里面包含很多教授对象。

　　当这个对象在关系数据库，由于外键在Professor表中，那么意思就是：  
　　2. 我是一个教授对象，这里有一个我所属的部门对象。

　　前后对比发现，其实完全是两者不同表达方式，这已经属于拷贝走样了，当我们从关系数据库中获得Department 对象时，得按照关系数据库规则绕一个弯来获得完整的Department 对象。也就是说：每次获得一个对象Department，我们得将这个需求意思翻译成关系数据库的表达方式，这种内耗式的翻译不但容易出错，也带来了程序员开发上的负担。万一不留神，翻译出错，问题就严重了。

　　当然，如果我们使用ODBMS，就不必做这些费力不讨好的翻译转换，也不再绕着弯子编程，只要直接告诉ODBMS或ORM框架如Hibernate，就能够获得一个真正对象意义上的Department。

　　以上是TSS的ODBMS一文大量篇幅阐述了对象和关系数据库矛盾的方面，所以，我们才认为：如果你使用对象语言，如Java/.NET/Ruby On Rails等，那么就必须坚持OO思想和方法，从程序中杜绝关系数据库对软件的影响，将关系数据库只看成是活动对象的“冬眠”(英文Hibernate)地方，这也就是ORM框架Hibernate的本义所在。