面向对象建模与数据库建模的比较Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E9_9D_A2_E5_90_91_E5_AF_B9_E8_c104_644565.htm 我们知道：一个软件从无到有需要经过如下几个阶段：分析、设计、编程、调试、部署和运行。 编程阶段我们通常使用Java/.NET这样面向对象语言工具，可以带来很多设计上的好处，但是也存在一个奇怪的现象：很多程序员虽然在使用OO语言，但是却在code非OO的代码，最终导致系统性能降低或失败，这个现象在Java语言尤其显得突出，难怪有些人就把问题归结于Java语言本身，睡不着觉怪床歪，又为了面子问题，说自己转向.NET，实际上是在 回避自己的问题和弱点。 那么，这些人的问题和弱点体现在什么地方呢？从上面软件生产过程来看，每个阶段都对前面有所依赖，在编程阶段出问题，追根溯源，问题无疑出在分析和设计阶段，分析设计作为一个软件产生的龙头，有着映射实际需求世界到计算机世界这样一个拷贝任务，如何做到拷贝不走样，是衡量映射方法好坏与否的主要判断标准。 目前，将需求从客观现实世界映射到计算机软件世界主要有两种方式：传统数据库分析设计和面向对象建模( object-oriented class model)， 当前软件主要潮流无疑是面向对象占据主流，虽然它可能不是唯一最好最简单的解决方案，但是它是最普通，也是最恰当的。 也就是说：在分析设计阶段，采取围绕什么为核心(是对象还是数据表为核心)的分析方法决定了后面编码阶段的编程特点，如果以数据表为核心进行分析设计，也就是根据需求首先得到数据表名和字段，然后培训程序员学会SQL语句如何操作这些数据表

，那么程序员为实现数据表的前后顺序操作，必然会将代码写成过程式的风格。 相反，如果分析设计首先根据需求得出对象模型(class Model)，那么程序员使用对象语言，再加上框架辅助，就很顺理成章走上OO编程风格。至于OO代码相比传统过程编码的好处不是本文重点，可参考J道(jdon.com)相关讨论，扩展性和维护性好，开发越深入开发速度越快无疑是OO系统主要优点。 本文重点主要是比较OO建模和数据表建模两者特点，这两者我们已经发现属于两个不同方向，也就是说，属于两个完全不同的领域，在J道其他文章里我们 其实已经把这两个领域上升为不同的学科，数据表建模属于数学范畴思维；而OO建模属于哲学思维。 下面我们看看面向对象的Class Model和Database Model是如何来表达客观世界的，也就是他们在表达需求上有些什么不同？ 面向对象模型(Class Model) 类代表一个对象类型，类在代码运行阶段将被创建为一个个对象实例， 每个类由两个部分组成：属性和行为，属性通常是一些数据状态值，也就是说：类将数据封装隐藏在自己内部了， 访问这些数据属性必须通过类公开的方法，或者接口。 别小看这样一个小小包装，却决定了以后代码的维护性和扩展性，打个比喻，日常生活中我们经常用各种盒子和袋子包装一些东西，这样做就是为了方便这些东西的携带或储藏，小到生活，大到客观世界每个地方，都是包装分类的影子，无论大小公司都是一个封装，行政部分单位划分，仓库物流更需要包装，我们从来不会因为嫌麻烦而不愿意引入一个似乎多余的盒子或袋子，那么有什么理由不在我们赖之生存的软件中（靠编软件吃饭）引入封装概念呢？ 这里可以再深入想像一下：不愿意用盒子和袋子的携带东

西大部分是一些急脾气的毛头小伙子，而偏偏这些小伙子又从事 软件工作，看来软件的非对象化是注定的，只是一个玩笑。 类的方法行为也有多种类型，如公开 私有等，我们可以设计一些方法为公开接口，而将另外一些行为隐藏起来，这样一个看似简单灵活的选择，却能够应付我们日后频繁的修改，软件不修改就不叫软件，软件修改了就崩溃是业务软件，专业的软件是抗修改的，而且能够极其方便快速地被修改。这些都依靠接口公开和隐藏这样一个简单魔术。 类的关系 我们不能只用一个一个单独的类来表达客观世界，因为客观世界存在千丝万缕的各种关系，在计算机领域无疑我们使用类的关系来表达映射这些关系。这里我们只探讨类在建模方法上的关系，而不是UML中类的通用关系。 类在建模上主要有如下几个关系： 类与类关系经常是这样：一个类包含一个类(构造性structural)，或者借助另外一个类达到某个功能（功能性），在对需求建模分析中，构造性的这种关系，也称为关联(Association)是我们关注重点，当然这种关系很显然表达的是一种静态的结构，比如电脑包含屏幕，他们之间的关系就是一种关联。本文来源:百考试题网 聚合(Aggregation)是一种表格式样的关联，表示一个类包含多项子类，这种关系是一种整体与部分的关系。 一个汽车有四个轮子，四个轮子是汽车的部分。 组成(Composition)是一种更强烈的聚合关系，一个对象实际是由其子对象组成，子对象也唯一属于父对象。 继承也是类建模中经常用到的关系，继承可以将一些数据属性抽象到父类中，避免重复，如入库单和出库单有 很多属性是差不多的，唯一不动的就是入库和出库的行为，那么我们可以抽象一个库单为父类，使用继承关系分别 表达入库单

和出库单。 在Evans DDD中，提到通过访问聚合根来遍历导航关联对象，这样做的好处很明显保证了对象的从属性，非常符合我们日常生活逻辑，比如，你要得到盒子里面的东西，必须首先得到盒子，然后经过一些准备如打开盒子，才能得到盒子里面的东西，假设一下，如果没有这样封装导航关系，盒子和东西都是可以透明并行得到，你想得到东西就能够直接获得，而不必经过打开盒子这一关，这样的访问方式首先怪诞，其次是不安全，如果盒子和东西放在数据表中，就会发生 这种情况。 数据库模型(Database Model 传统E-R模型 ) 好了，下面我们谈论关系数据表模型，以前我们朴素的分析设计都是根据需求直接建立数据表的方式来进行的，为什么称为朴素，是因为我们好像只有数据结构 算法方面的知识，也认为只有这样做才叫做软件。 那么既然这条路能够走出来，我们看看这个领域是如何映射客观世界的。 数据表由于技术提供庞大数据存储和可靠的数据访问，正在不断从技术领域走向社会领域，很多不懂计算机的人 也知道需要建立数据库来管理一些事务，但是不代表我们就必须围绕数据库的分析设计。 数据表是类似前面的“ 类 ”，也是一种表达客观世界的基本单元，表有多列字段，表的字段是保存数据的，每个字段有数据类型。注意，这里没有数据的封装和公开，表的字段是赤裸的，只要有数据库访问权限，任何人都可以访问，没有结构层次关系，都是扁平并列的，如果你想在数据表字段之间试图看出客观世界中的层次和封装，那就错了，在拷贝不走样这个条件下， 这个映射方法至少把这个信息拷贝丢了。百考试题论坛 数据表也有一些行为，这些行为是基于实体的一些规则： 约束(Constraints) 能够保证不同表

字段之间的关系完整安全性，保证数据库的数据安全。 触发器(Triggers)提供了实体在修改 新增和删除之前或之后的一些附加行为， 存储过程(Database stored procedures)提供数据专有的脚本性语言，存储过程象一个数学公式虽然具有抽象简洁美学，但是这种简洁是闷葫芦美学，不是大众美学，只有公式存储过程发明者自己了解精通，别人无法插手，软件不是科学，不是比谁智商高，科研水平高，软件是人机工程，更讲究集体，讲究别人是否方便与你协同扩展软件。 关系数据表的遍历访问是通过列字段遍历或表join等方式实现，SQL语句是这样标准语言，只要会写SQL语句，就能访问那些失去层次，失去客观世界特征的苍白的数据，这样的系统能够多少真实反映客观需求，是有问号的？SQL语句是否方便修改，是否经得起频繁修改而不出错，都是有疑问的地方，是否 SQL语句越复杂，修改越快，或者另外一个程序员能够很快修改不是自己写的SQL语句，这些都是问题所在。 数据表关系 数据表的关系主要是通过外健或专门关联表来表达的，这种关系虽然可以反映1:1或1:N这样关系，但是无法表达关系的性质，是紧密组成关系式的关联，还是无关紧要的普通关系，正因为如此，使用数据表分析设计时，我们会有蜘蛛网的关系表，这些关系由于在后期无法分辨性质，无法进行整理，增加了系统复杂性。 更重要的是：分析就是对一个可能陌生领域进行探寻，如果使用数据表的分析设计方法，那么我们实际就是在陌生领域中寻找数据表这样一个形式，那么有可能产生误判断，将一个实则是表达关系的东东误认为是一个实体表，因为关系表必然带来关系，这样，就必然产生蜘蛛网式的数据表模型，将简单问题复杂化。 总结 要谈方法

，这个世界其实只存在两种：一是将复杂问题简单化的方法；一个是将简单问题复杂化的方法。 你使用什么样的方法，你就有什么样的世界观，就是什么样的人，但是对于软件这个领域，你只能选择前者。百考试题－全国最大教育类网站(100test.com) 因为方法的不同，软件路线也就存在下面几个路线：完全面向对象类建模路线(J道网站和笔者一直致力于这种路线的推介)；一种是对象和关系数据库混合型，还有一种就是过去的完全关系数据库类型软件（如Foxpro/VB/Delphi等）。 编辑特别推荐: 指点一下：到底该不该去考JAVA认证? Java面试题：第一锦 Java面试题：第二锦 Java面试题：第三锦 Java面试题：第四锦 Java面试题：第五锦 一个有趣的编程:程序员的爱情故事 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