目录参考：<http://www.open-open.com/lib/view/open1450427478266.html>

PO(persistant object) 持久对象

在 o/r 映射（数据模型是对象模型-object，存储模型是关系模型-relational，个人理解就是将对象存储到数据库实现持久化）的时候出现的概念，如果没有 o/r 映射，就没有这个概念存在了。通常对应数据模型 ( 数据库 ), 本身还有部分业务逻辑的处理。可以看成是与数据库中的表相映射的 java 对象。最简单的 PO 就是对应数据库中某个表中的一条记录，多个记录可以用 PO 的集合。 PO 中应该不包含任何对数据库的操作。

百度百科：

O/R映射层是持久层的一个特例，它的数据模型是对象模型（Object），存储模型是关系模型（Relational），cmp和Hibernate是对象模型到关系模型之间转换的两种不同实现。

由于对象模型和关系模型应用广泛，所以很多人错误的认为数据持久层就是对象模型到关系型数据库的转换，其实持久化的范围更广一些。

其中，数据持久化就是将内存中的数据模型转换为存储模型，以及将存储模型转换为内存中数据模型的统称。数据模型可以是任何数据或对象模型，存储模型可以是关系模型、XML、二进制等。

持久层（Persistence Layer），即专注于实现数据持久化应用领域的某个特定系统的一个逻辑层面，将数据使用者和数据实体相关联。

# [PO对象的三种状态。](http://www.cnblogs.com/zhengchubin/archive/2012/10/09/2716704.html)

(save,persist,update,saveOrUpdate,merge,flush,lock)及对象的三种状态  
hibernate的保存  
hibernate对于对象的保存提供了太多的方法，他们之间有很多不同，这里细说一下，以便区别。  
一、预备知识  
在所有之前，说明一下，对于hibernate，它的对象有三种状态，transient、persistent、detached  
下边是常见的翻译办法：  
transient：瞬态或者自由态  
(new DeptPo(1,”行政部”,20,”行政相关”)，该po的实例和session没有关联，该po的实例处于transient)  
persistent：持久化状态  
(和数据库中记录项影射的Po实例，它的状态是persistent, 通过get和load等得到的对象都是persistent)  
detached：脱管状态或者游离态  
    (1)当通过get或load方法得到的po对象它们都处于persistent,但如果执行delete(po)时(但不能执行事务),该po状态就处于detached, (表示和session脱离关联),因delete而变成游离态可以通过save或saveOrUpdate()变成持久态  
(2)当把session关闭时，session缓存中的persistent的po对象也变成detached  
因关闭session而变成游离态的可以通过lock、save、update变成持久态  
持久态实例可以通过调用 delete()变成脱管状态。  
通过get()或load()方法得到的实例都是持久化状态的。  
脱管状态的实例可以通过调用lock()或者replicate()进行持久化。  
   
save()和persist()将会引发SQL的INSERT，delete()会引发SQLDELETE，  
而update()或merge()会引发SQL UPDATE。对持久化（persistent）实例的修改在刷新提交的时候会被检测到，它也会引起SQL UPDATE。  
saveOrUpdate()或者replicate()会引发SQLINSERT或者UPDATE  
二、save 和update区别  
把这一对放在第一位的原因是因为这一对是最常用的。  
save的作用是把一个新的对象保存  
update是把一个脱管状态的对象或自由态对象（一定要和一个记录对应）更新到数据库  
   
三、update 和saveOrUpdate区别  
这个是比较好理解的，顾名思义，saveOrUpdate基本上就是合成了save和update,而update只是update;引用hibernate reference中的一段话来解释他们的使用场合和区别  
通常下面的场景会使用update()或saveOrUpdate()：   
程序在第一个session中加载对象,接着把session关闭   
该对象被传递到表现层   
对象发生了一些改动   
该对象被返回到业务逻辑层最终到持久层   
程序创建第二session调用第二个session的update()方法持久这些改动   
   
saveOrUpdate(po)做下面的事:   
如果该po对象已经在本session中持久化了，在本session中执行saveOrUpdate不做任何事   
如果savaOrUpdate(新po)与另一个与本session关联的po对象拥有相同的持久化标识(identifier)，抛出一个异常   
org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session: [org.itfuture.www.po.Xtyhb#5]  
saveOrUpdate如果对象没有持久化标识(identifier)属性，对其调用save() ，否则update() 这个对象   
   
四、persist和save区别  
这个是最迷离的一对，表面上看起来使用哪个都行，在hibernate reference文档中也没有明确的区分他们.  
这里给出一个明确的区分。（可以跟进src看一下，虽然实现步骤类似，但是还是有细微的差别）  
主要内容区别：  
1，persist把一个瞬态的实例持久化，但是并"不保证"标识符(identifier主键对应的属性)被立刻填入到持久化实例中，标识符的填入可能被推迟到flush的时候。  
   
2，save, 把一个瞬态的实例持久化标识符，及时的产生,它要返回标识符，所以它会立即执行Sql insert  
   
五、saveOrUpdate,merge和update区别  
比较update和merge  
update的作用上边说了，这里说一下merge的  
如果session中存在相同持久化标识(identifier)的实例，用用户给出的对象覆盖session已有的持久实例   
(1)当我们使用update的时候，执行完成后，会抛出异常   
(2)但当我们使用merge的时候，把处理自由态的po对象A的属性copy到session当中处于持久态的po的属性中，执行完成后原来是持久状态还是持久态，而我们提供的A还是自由态  
   
六、flush和update区别  
这两个的区别好理解  
update操作的是在自由态或脱管状态(因session的关闭而处于脱管状态)的对象//updateSQL  
而flush是操作的在持久状态的对象。  
默认情况下，一个持久状态的对象的改动（包含set容器）是不需要update的，只要你更改了对象的值，等待hibernate flush就自动更新或保存到数据库了。hibernate flush发生在以下几种情况中：  
1，调用某些查询的和手动flush(),session的关闭、SessionFactory关闭结合   
get()一个对象，把对象的属性进行改变,把资源关闭。  
2，transaction commit的时候（包含了flush）  
   
七、lock和update区别  
update是把一个已经更改过的脱管状态的对象变成持久状态  
lock是把一个没有更改过的脱管状态的对象变成持久状态(针对的是因Session的关闭而处于脱管状态的po对象(2)，不能针对因delete而处于脱管状态的po对象)  
对应更改一个记录的内容，两个的操作不同：  
update的操作步骤是：  
(1)属性改动后的脱管的对象的修改->调用update  
lock的操作步骤是：  
(2)调用lock把未修改的对象从脱管状态变成持久状态-->更改持久状态的对象的内容-->等待flush或者手动flush  
八、clear和evcit的区别  
  clear完整的清除session缓存  
   evcit(obj)把某个持久化对象从session的缓存中清空。

转载请加上http://www.cnblogs.com/zhengchubin/

DO（Domain Object）领域对象-领域模型里面的对象

DO(Domain Object)领域对象，也叫做业对象；是对业务角色和业务实体之间应该如何联系和协作以执行业务的一种抽象；**个人感觉这个就是一个抽象，可以代表以下的各种XO**。因为以下的XO都代表了不同的业务中的抽象。

就是从现实世界中抽象出来的有形或无形的业务实体（各个领域中的对象，然后分析各个对象之间的关系，从而组合为一个业务解决系统）。

百度百科：

[领域模型](http://baike.baidu.com/view/757895.htm)是对领域内的概念类或现实世界中对象的可视化表示。又称概念模型、领域对象模型、分析对象模型。它专注于分析问题领域本身，发掘重要的业务领域概念，并建立业务领域概念之间的关系。

## 概念

[编辑](javascript:;)

业务对象模型（也叫领域模型 domain model）是描述业务用例实现的对象模型。它是对业务角色和业务实体之间应该如何联系和协作以执行业务的一种抽象。业务对象模型从业务角色内部的观点定义了业务[用例](http://baike.baidu.com/view/706238.htm)。该模型为产生预期效果确定了业务人员以及他们处理和使用的对象（“业务类和对象”）之间应该具有的静态和动态关系。它注重业务中承担的角色及其当前职责。这些模型类的对象组合在一起可以执行所有的业务用例。

## 核心元素

[编辑](javascript:;)

业务角色显示了一个人承担的一系列职责。业务实体表示使用或产生的可交付工件、资源和事件。业务[用例](http://baike.baidu.com/view/706238.htm)实现显示了协作的业务角色和业务实体如何执行某个工作流程。使用以下几种图来记录业务用例实现：图显示参与的业务角色和业务实体。活动图，其中泳道显示业务角色的职责，而对象流显示如何在[工作流程](http://baike.baidu.com/view/3992935.htm)中使用业务实体。序列图描述业务角色和业务主角之间交互的详细情况，并显示如何在业务用例执行过程中访问业务实体。

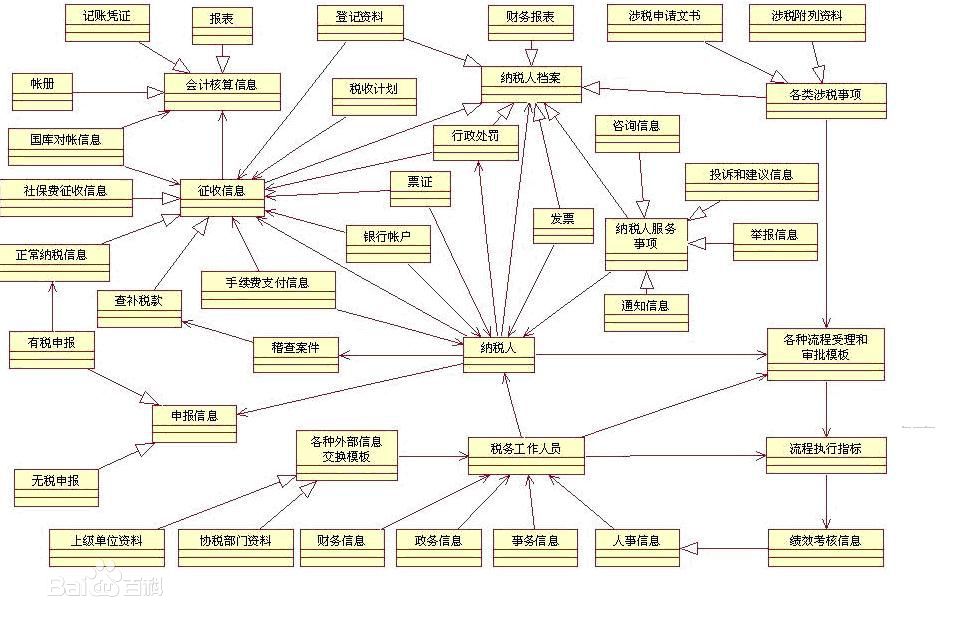
业务对象模型将结构（各个组成部分）的概念和行为（组成部分之间的协作）的概念结合了起来。

它是一个纽带工件，用于对业务关系进行清晰的表述，表述方式与软件开发人员的思考方式类似，同时仍保留一些纯粹的业务内容。将我们所知道的有关业务的信息按照对象、属性和职责进行了合并。

它探索业务领域知识的本质，所采用的方式使我们能够从对业务问题的思考转变到对软件应用程序的思考上来。

它是一种确定需求的方法，使需求能够为待建信息系统使用，并得到该系统的支持。（对于接项目的组织就是由这个来构建结构：首先分析你项目里的实体有多少实体 然后根据实体之间的关系去绘制UML图 在根据UML来建立数据库）

确定业务对象定义、对象间关系、对象名称和对象间关系名称的流程使我们能够以一种能被业务领域专家理解和验证的精确方式来表达业务领域知识。



如何定义领域模型（概念模型）

参考：<http://blog.csdn.net/chz_cslg/article/details/23957619>

作为OOAD（Object Oriented Analysis Design，[面向对象](http://baike.baidu.com/view/125370.htm)的分析和设计，[面向对象分析与设计](http://baike.baidu.com/view/1488767.htm)）中的第二步，概念模型的识别显得比用例识别更加困难。

### ****为什么要创建领域模型****

**降低与OO建模之间的表示差异**。

领域层软件类的名称要源于领域模型中的名称，以使对象具有源于领域的信息和职责。

打个比方，你可以用一长串0和1来表示“薪水册”，可是这种软件表示与我们脑中的薪水册领域模型之间存在巨大的差异，这将影响我们对软件的理解和修改。而OO建模则可以减小这一差异。

### ****如何创建领域模型****

那又如何创建领域模型呢？

某一领域的模型，也可以说是某个领域对于一个问题的一种特定看待方式

可以通过以下四个步骤：

1. 寻找概念类
2. 将其绘制为UML（统一建模语言或标准建模语言）类图中的类
3. 添加关联
4. 添加属性

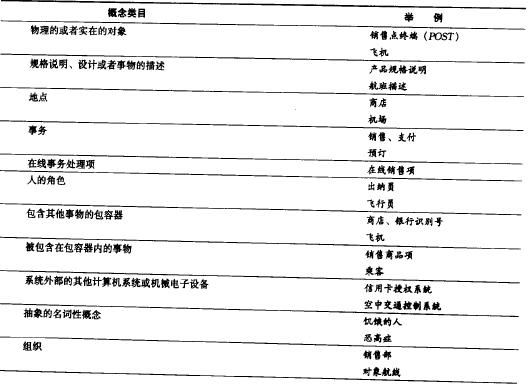
#### ****如何找到概念类****

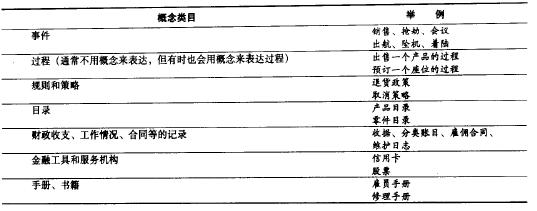
1）重用和修改现有的模型。

这是**首要**、**最佳**且**最简单**的办法。可以从已发布的领域模型和书籍中获得。

2）使用分类列表

示例：





（图二）

3）确定名词列表

在对**领域的文本性描述**中识别名词和名词短语，将其作为候选的概念类或属性。

缺点：自然语言的不精确性，不同名词短语可能表示同一概念类或属性，此外可能还有歧义。

建议与**概念类分类列表**一同使用。

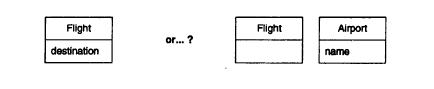
在实践中，在发现概念类时，一般直接为其**绘制UML类图**。

了解了几种识别概念类的方法，我们接下来讨论过程中可能出现的问题。

**1）最常见错误：把应该是概念类的事物表示为属性。**

准则：**如果我们认为某概念类X不是现实世界中的数字或文本，那么X可能是概念类而不是属性**。

考虑一下航空预定领域。destination应该作为Flight的属性，还是作为单独的概念类Airport?



（图三）

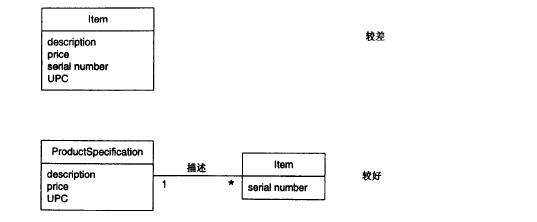
在现实世界里，目的地机场不会被看作是数字或文本，而是一占据大规模空间的事物。因此，Airport应该是个概念，而不是属性。

**2）缺少描述类**

描述类包含描述其他事物的信息。例如，ProductDescription记录Item的价格、图片和文字描述。

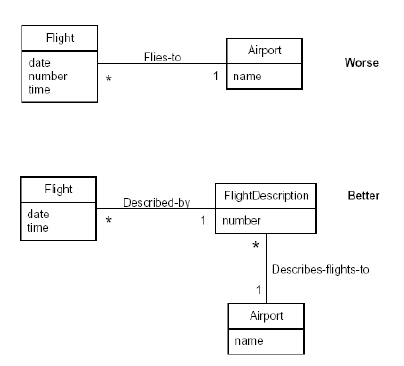
**为什么**使用描述类？

假设：雀巢咖啡，大受欢迎，销售一空。这就意味着雀巢咖啡的所有Item实例都从计算机存储器中被删除。这时如果有人问：雀巢咖啡多少钱一盒？那将没法回答。因为价格是记录在实例上的，而这些实例都已经被删除。由此可以看出，需要其他事物来记录雀巢咖啡的**描述**（规格说明）。



（图四）

又比如，虹桥机场每天有很多航班，同一架飞机有多个航线，但其一架飞机只有一个飞机号码。如果仅仅有两个概念类Flight和Airport，假设有一天天降大雾，所有航班取消，我们删除了所有Flight记录。这时如果我们想知道是否存在号码为12345的一架飞机，那我们将无法查到。由此可以看出，我们还需要一个描述类FlightDescription。



**何时需要？**

1. 需要有关商品或服务的描述，独立于任何商品或服务的现有实例。

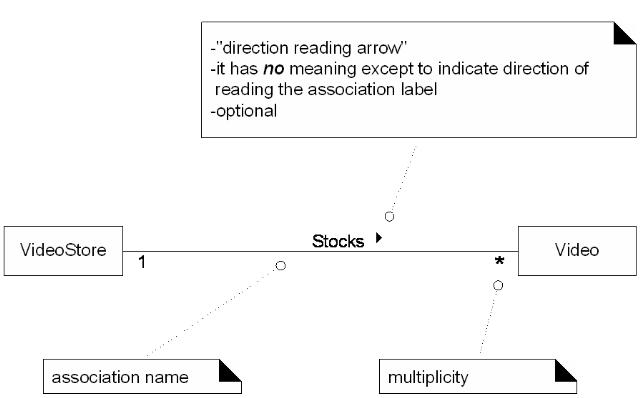
2. 删除其所描述事物（如Item）的实例后，导致信息丢失，而这些信息是需要维护的，但是被错误地与所删除的事物关联起来。

3. 减少冗余或重复信息。

#### ****关联****

**什么是关联？**

关联是类（类的实例）之间的关系，表示有意义和值得关注的连接。



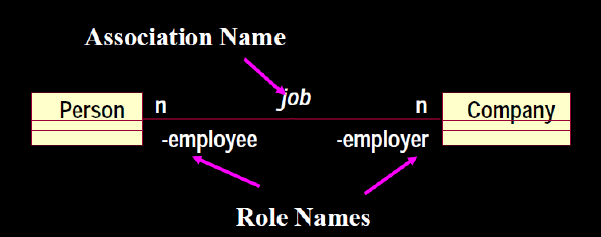
（图五）

**关联名**

关联名来描述一个关联的本质。

**角色（Role）**

用来描述离角色名较近一端的类在另一端的类面前所呈现的面貌。



**何时表示关联？**

1. 如果存在需要保持一段时间的关系，将这种语义表示为关联（**“需要记住”**的关联）。

2. 从常见关联列表中派生的关联。

准则：**要避免在领域模型中加入太多的关联**。

在领域建模过程中，关联不是关于数据流、[**数据库**](http://lib.csdn.net/base/mysql)外键联系、实例变量或软件方案中的对象连接的语句；关联声明的是针对现实领域**从纯概念角度**看有意义的关系。

**在UML中如何对关联命名**

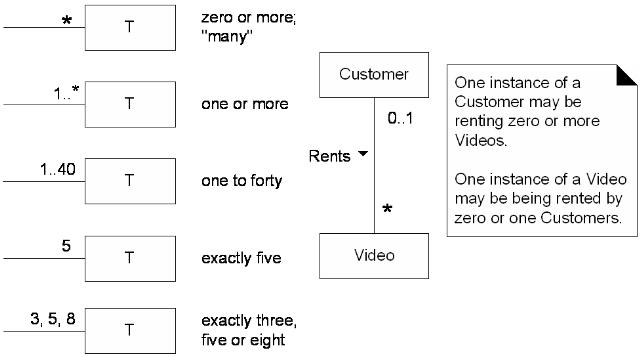
以“类名－动词短语－类名”的格式为关联命名，其中的动词短语构成了可读的和有意义的顺序。

图五对应于：**VideoStore Stocks Video**

**多重性**

多重性定义了类A有多少个实例可以和类B的一个实例关联。（见图五）

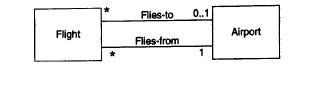
多重性的值表示在**特定时刻**（而不是在某个时间跨度内）有效关联的实例数量。



（图六）

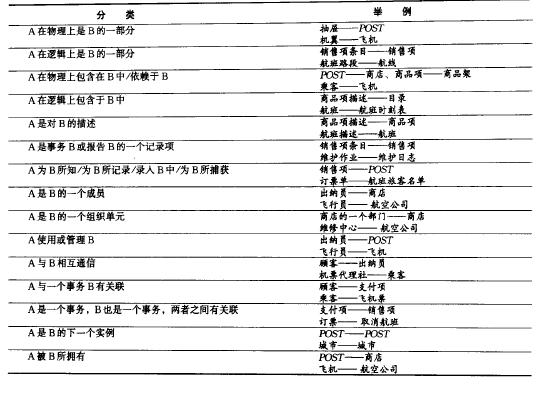
多重性是和语境有关的。

两个类之间的多重关联：

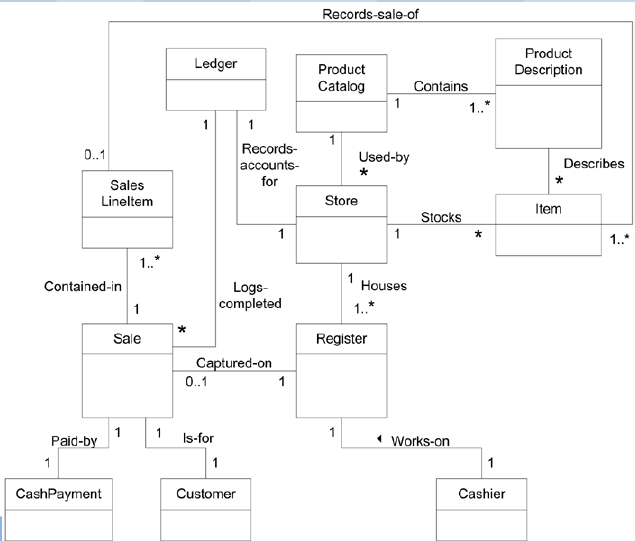


（图七）

**常见关联列表**



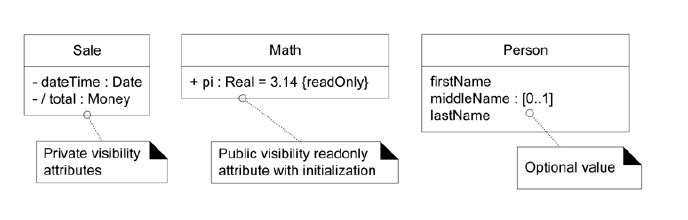
（图八）



一个例子

#### ****属性****

在UML中，一个属性的完整语法是：visibility name : type multiplicity = default {property-string}，即可见的名称：类型 = 默认值 {性质}。

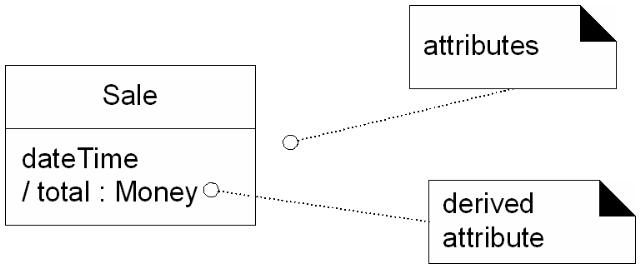


**什么是属性？**

属性是对象的逻辑数据值。

**何时展示属性？**

当需求（例如，用例）建议或暗示**需要记住**信息时，引入属性。



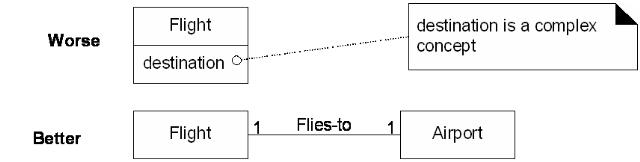
（图九）

**导出属性**（derived attribute）：可以由其他信息导出的属性。

准则：**大部分属性类型应该是“简单”数据类型**，例如数字和布尔。通常，**属性的类型不应该是复杂的领域概念**，例如Sale或AirPort。

准则：**领域模型中属性的类型更应该是数据类型**。

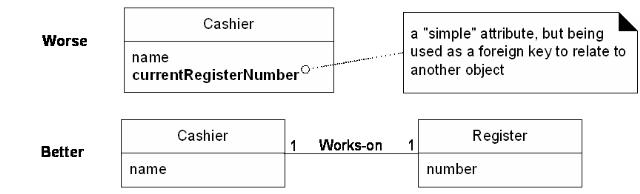
准则：**通过关联而不是属性来表示概念类之间的关系**。



（图十）

**领域模型是概念透视图，不是软件透视图。在设计模型中，属性可以是任何类型。**

准则：**任何属性都不表示外键**。



（图十一）

### ****结论****

没有所谓唯一正确的领域模型。所有模型都是对我们试图要理解的领域的近似。领域模型主要是在特定群体中用于理解和沟通的工具。有效的领域模型捕获了当前需求语境下的本质抽象和理解领域所需要的信息，并且可以帮助人们理解领域的概念、术语和关系。

TO(Transfer Object) ，数据传输对象

在应用程序不同 tie( 关系 ) 之间传输的对象

DTO（Data Transfer Object）数据传输对象

这个概念来源于J2EE的设计模式，原来的目的是为了EJB的分布式应用提供粗粒度的数据实体，以减少分布式调用的次数，从而提高分布式调用的性能和降低网络负载，但在这里，可以泛指用于展示层与服务层之间的数据传输对象—也可以将展示层的数据转换为一个对象，这也可以称为DTO。如json数据转换为video对象

VO(value object) 值对象（简单的理解为json数据）

通常用于业务层之间的数据传递（如json数据、xml数据），和 PO （持久层数据如数据库或者内存数据）一样也是仅仅包含数据而已。但应是抽象出的业务对象 , 可以和表对应 , 也可以不 , 这根据业务的需要 。用 new 关键字创建，由 GC 回收的。

BO(business object) 业务对象

从业务模型的角度看 , 见 UML 元件领域模型中的领域对象。封装业务逻辑的 java 对象 , 通过调用 DAO 方法 , 结合 PO,VO 进行业务操作。 business object: 业务对象 主要作用是把业务逻辑封装为一个对象。这个对象可以包括一个或多个其它的对象。 比如一个简历，有教育经历、工作经历、社会关系等等。 我们可以把教育经历对应一个 PO （类似一张表），工作经历对应一个 PO（类似一张表） ，社会关系对应一个 PO 。 建立一个对应简历的 BO 对象处理简历，每个 BO 包含这些 PO 。 这样处理业务逻辑时，我们就可以针对 BO 去处理。

POJO(plain ordinary java object) 简单无规则 java 对象

纯的传统意义的 java 对象。就是说在一些 Object/Relation Mapping 工具中，能够做到维护数据库表记录的 persisent object 完全是一个符合 Java Bean 规范的纯 Java 对象，没有增加别的属性和方法。我的理解就是最基本的 Java Bean ，只有属性字段及 setter 和 getter 方法！。

DAO(data access object) 数据访问对象

是一个 sun 的一个标准 j2ee 设计模式， 这个模式中有个接口就是 DAO ，它负责持久层的操作。为业务层提供接口。此对象用于访问数据库。通常和 PO 结合使用， DAO 中包含了各种数据库的操作方法。通过它的方法 , 结合 PO 对数据库进行相关的操作。夹在业务逻辑与数据库资源中间。配合 VO, 提供数据库的 CRUD 操作.