## 思想

UI层和Data层不变

UI层与Service层之间加一个Controller，只是类，没有接口，持有一个或多个Service

逻辑层分为两部分：应用层和领域层（领域驱动模型）

应用层就是之前讲的Service，持有同层的Service和下层的DAO。

应用层的命名是xxService

应用层没有数据，只有方法，负责转发在DAO和Controller之间转发。

而领域层的命名可以是xxDomain，比如OrderDomain

持有数据，和VO/PO数据一致。并且具有方法，是业务逻辑方法。

可以将之前设计的Service分隔，与DAO和Controller交互的方法保留，其他的都是逻辑方法，分配给一个或多个Domain来处理。Domain是一个类，不是接口。

VO/PO传到逻辑层，逻辑层new一个Domain来承载VO和PO的数据，并将VO/PO丢弃

<http://www.blogjava.net/AndersLin/archive/2006/06/15/53086.html>

<http://verran.iteye.com/blog/58035>

Domain Object包含了这些需要被持久化的业务数据，同时还包含了与之相关所有业务操作，并且有自己的继承体系

Domain Object关心的是自己，只处理和自己状态有关的逻辑。   
Domain Service则处理Domain Object之间的关系，以及Domain Object和外部的交互。

domain object表示领域组件，如果把他们相关的业务逻辑都移到万能的上帝Service组件里，domain object的意义也就不是很重要了。  
这对于OO而言也是不可取的。白痴domain object没有任何意义，业务功能让他们和service组件绑定到了一齐，要复用他们，就要复用他们所有的。  
这造成Service 层和domain object层耦合太大，是有弊端的。  
我们可以看到这样的情况下，Service中的重复代码很多，复用性不高。

Service组件从某种程度上来说是面向过程的。和use case中的用例有一定的对应关系。当然粒度会有不同。  
所以对于OO来说，service这一层代表了不好的味道，要尽量的薄。它只不过是起到一种代理facade，调度组件的作用。  
对于domain object来说，必须包含维护他们自身及相互关系的业务逻辑。如果将这部分逻辑放到service层去了，就是转移了相应的逻辑，造成Service中 的重复代码增多，复用性降低。而domain就是对service的进一步改进使他更OO而不是一个流程这样他就有点过程性了!

* VIEW Controller（每个用例1+个）-->**就是我们所说的Controller(持有一个或多个Service)**
  + 控制界面间跳转
  + 将界面交互转给Logic Controller
* Logic Controller（每个用例1+个，平层调用1+个）**-->Service（持有DAO，对应着一个或多个Domain）**
  + 接受界面层调用，分解、转发给领域类**-->Domain（负责具体业务逻辑）**
  + 必要的时候调用Data层controller
* Data DAO（controller，每个内聚实体类群一个）**-->DAO**
  + 接收logic调用
  + 维护持久化数据
* **明确上一层controller与下一层controller的持有引用关系**
  + 可以在启动时设置关联或者建立Factory创建并设置关联
  + 设置之后可以直接调用

**表示层（Presentation）**

　　该层的主要职责是通过用户界面向用户显示数据信息，同时解释用户的命令，并把用户的请求发送到应用层。

**应用层（Application）**

　　应用层主要用于协调不同领域对象之间的动作或领域模型与基础结构层组件之间的工作，以完成一个特定的、明确的系统任务。

　　有一个需求是保存一个订单后，需要发送一封Email给用户。保存订单是业务，业务代码在领域层中；而发送Email则应是基础设施层的功能，那么就需要使用应用层来协调领域对象和基础设施对象之间的动作。

**领域模型层（Domain Model）**

　　将业务逻辑高度内聚到领域层，所以领域层是整个系统的核心，它只与实际业务相关，不应关心任何技术细节，尽可能的做到与持久化无关。

* **实体**

　　实体在领域模型中非常重要，由标识来区分的对象称为实体（即标识必须唯一）。

* **值对象**

　　和实体不同，值对象没有标识，不需要跟踪值对象的状态，而且值对象非常容易创建和丢弃。值对象是不可变的，用一个构造器创建，所有属性都定义为 只读。如果其中一个属性需要修改，那么就需要重新创建一个新的值对象来进行整体替换。值对象的相等性比较是通过各个属性值的比较来完成的。

* **领域层服务**

　　在设计领域模型时，有些业务行为不适合放在任何一个领域对象中，或者该业务行为需要联合多个领域实体才能完成，那么我们就把其放在对应的领域服务中。领域层服务负责和领域中的实体对象、值对象以及其他领域层对象交互。

* **聚合（Aggregate）**

　　聚合通常定义一组关联的对象，以及对象和关系之间的边界，作为一个数据更改的单元处理。每个聚合只有一个聚合根（实体对象）。聚合根可以引用其他聚合根，聚合内的对象可以引用聚合内的另一个对象，但是聚合边界外的任何对象不能绕过聚合根对象访问聚合内的对象。

* **仓储（Repository）**

　　为每一个聚合根对象创建一个仓储，表示该种类型的所有对象为一个概念的对象集合，对仓储的访问通过类似集合的接口。仓储的要点是让开发人员将精力聚焦在领域模型逻辑上，并将真实的数据访问隐藏在仓储接口后面，这就是之前说的领域模型与持久化无关。

* **工厂（Factory）**

　　很多时候，构造一个聚合以及其所有的关系、约束、规则等比较复杂，让一个实体对象自己负责对象的创建就会使得代码变得混乱。此时就需要有一个工厂，能够知道如何构建这些类型的对象，并统一进行创建。