|  |  |
| --- | --- |
| |  | | --- | | 软件开发要干什么：  1.反映真实世界要自动化的业务流程  2.解决现实问题  领域Domain  1.Domain特指软件关注的领域  2.在不能充分了解业务领域的情况下是不可能做出一个好的软件  **领域建模**  http://www.uml.org.cn/mxdx/images/2014090341.png  http://www.uml.org.cn/mxdx/images/2014090342.jpg  http://www.uml.org.cn/mxdx/images/2014090343.jpg  http://www.uml.org.cn/mxdx/images/2014090344.jpg  **领域模型驱动设计**  1.分层架构  2.实体  3.值对象  4.服务  5.模块  6.聚合  7.工厂  8.资源库  **分层架构：**  **http://www.uml.org.cn/mxdx/images/2014090345.jpg**  1.将领域模型相关的代码集中到一个层中，把它从用户界面、应用和基础设施代码中分隔开来  2.释放领域对象的显示自己、保存自己、管理应用任务等职责，让它专注于展现领域模型  3.复杂的程序切分成层  4.层中采用内聚的设计  5.层仅依赖于它底下的那层  http://www.uml.org.cn/mxdx/images/2014090346.jpg  **实体entity：**  有一类对象拥有唯一标识符  1.能够跨越系统的生命周期甚至能超越软件系统的一系列的延续性和标识符  2.这样的对象称为实体。  **值对象-value Object**  1.对某个对象是什么不感兴趣，只关心它拥有的属性  2.用来描述领域的特殊方面、且没有标识符的一个对象，叫做值对象  3.能被简单的创建和丢弃，生命周期中不会被持久化  4.值对象可以被共享，值对象应该不可变  **服务-service（比webservice更细粒度服务描述）**  1.领域中的一些动词，代表了领域中的一个重要的行为，却不属于任何对象  1)服务执行的操作涉及一个领域概念，这个领域概念通常不属于一个实体或者值对象  2)被执行的操作涉及到领域中的其他的对象  3)操作是无状态的  2.服务对象不再拥有内置的状态  3.服务对象担当重要的协调功能  4.开发通用语言时，领域中的主要概念被引入到语言中，语言中的名词很容易被映射成对象。  语言中对应那些名词的动词变成那些对象的行为。但是有些领域中的动作，它们是一些动词，看上去却不属于任何对象。它们代表了领域中的一个重要的行为，所以不能忽略它们或者简单的把它们合并到某个实体或者值对象中。给一个对象增加这样的行为会破坏这个对象，让它看上去拥有了本该属于它的功能。  **模块**  1.将相关领域模型提炼分类，分而治之  2.将高关联度的模型分组到一个模块以提供尽可能大的内聚（以能完整完成任务为准）  3.分层是水平划分  4.模块是垂直划分(Domain内部)  参考架构概述  1.领域驱动设计(DomainDriven Design)有一个官方的sample工程，名为DDDSample  2.官网：http://dddsample.sourceforge.net/  3.该工程给出了一种实践领域驱动设计的参考架构  http://www.uml.org.cn/mxdx/images/2014090347.jpg  http://www.uml.org.cn/mxdx/images/2014090348.jpg  http://www.uml.org.cn/mxdx/images/2014090349.jpg  http://www.uml.org.cn/mxdx/images/20140903410.jpg  **架构概述**  http://www.uml.org.cn/mxdx/images/20140903411.jpg  **详细架构**  http://www.uml.org.cn/mxdx/images/20140903412.png  **架构详解:Interfaces-接口层**  http://www.uml.org.cn/mxdx/images/20140903413.png  1.该层包含与其他系统进行交互的接口与通信设施，在多数应用里  2.可能提供包括WebServices、RMI或Rest等在内的一种或多种通信接口  3.该层主要由Facade、DTO和Assembler三类组件构成，三类组件均是典型的J2EE模式  **DTO**  http://www.uml.org.cn/mxdx/images/20140903414.png  1.DTO- DataTransfer Object（数据传输对象），也常被称作VO-ValueObject(值对象)  2.DTO设计之初是为了将细粒度的领域对象包装为粗粒度的数据结构，减少网络通信并简化调用接口  **DTO 作用**  1.减少网络流量  2.简化远程对象和远程接口  3.传输更多的数据减少远程调用次数  4.避免将领域状态跨层次传递  5.由于同步和版本控制增加了复杂性  **DTO 应用时序图**  http://www.uml.org.cn/mxdx/images/20140903415.png  Assembler  http://www.uml.org.cn/mxdx/images/20140903416.png  1.DTO与领域对象之间的相互转换工作多由Assembler承担  2.因此Assembler几乎总是同DTO一起出现。  **Assembler 实现方案**  **http://www.uml.org.cn/mxdx/images/20140903417.png**  **Fa?ade**  http://www.uml.org.cn/mxdx/images/20140903418.png  1.实践Facade的过程中最难把握的问题就是Facade的粒度问题。  2.传统的Service均以实体为单位进行组织，而Facade应该具有更粗粒度的组织依据，较为合适的粒度依据有：  3.一个高度内聚的模块一个Facade  4.或者是一个“聚合”(特指领域驱动设计)一个Facade.  **Facade 实现方案**  **http://www.uml.org.cn/mxdx/images/20140903419.png**  **Facade 应用时序图**  **http://www.uml.org.cn/mxdx/images/20140903420.png**  **Service**  **http://www.uml.org.cn/mxdx/images/20140903421.jpg**  1.Service会与多种组件进行交互  2.这些组件包括：  1)其他的Service  2)领域对象  3)Repository  4)DAO  **Service 应用时序图**  **http://www.uml.org.cn/mxdx/images/20140903422.png**  **Domain-领域层**  http://www.uml.org.cn/mxdx/images/20140903423.png  1.Domain层是整个系统的核心层，该层维护一个使用面向对象技术实现的领域模型，几乎全部的业务逻辑会在该层实现  2.Domain层包含:  1)Entity（实体）  2)ValueObject(值对象)  3)Domain Event（领域事件）  4)Repository（仓储）等  **Infrastructure-基础设施层**  1.基础设施层nfrastructure为Interfaces、Application和Domain三层提供支撑  2.所有与具体平台、框架相关的实现会在Infrastructure中提供，避免三层特别是Domain层掺杂进这些实现，从而“污染”领域模型  3.Infrastructure中最常见的一类设施是对象持久化的具体实现  http://www.uml.org.cn/mxdx/images/20140903424.png  **“传统”架构-贫血领域模型**  http://www.uml.org.cn/mxdx/images/20140903425.jpg  DDD && SOA  1.DDD 领域模型驱动设计  2.SOA 面向服务的架构 | |