实战DDD(Domain-DrivenDesign领域驱动设计)[1] PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/141/2021_2022__E5_AE_9E_E6_88_98DDD__c29_141323.htm 2004年著名建模专家Eric Evans发表了他最具影响力的著名书籍：Domain-Driven Design Tackling Complexity in the Heart of Software（中文译名：领域驱动设计 2006年3月清华出版社译本，或称 Domain Driven-Design architecture [Evans DDD]）。 Martin Fowler作序说；“ 希望本书是一本非常有影响力的书籍,..... Eric最值得我尊敬的一个方面是他敢于讨论还未取得成功的事情 ”，其实，时值今年2006年，DDD开发框架已经层出不穷（如RoR、RIFE、JdonFramework等），我们项目软件包结构都变成了这样：xxx.model.xxx.service，DDD思想已经遍地开花，不能再说不成功了。 DDD是告诉我们如何做好业务层！并以领域驱动设计思想来选择和合适的框架，本文以基于JdonFramework开发的JiveJdon3.0说明DDD方法的实战应用。 首先必须认识到：领域建模是一种艺术的技术，不是数学的技术，它是用来解决复杂软件快速应付变化的解决之道（快速适应需求变化的软件复用）。 我们知道软件的产生过程是：分析、设计、编程、测试、部署。过去，分析领域和软件设计是分裂的，分析人员从领域中收集基本概念；而设计必须指明一组能北项目中适应编程工具构造的组件，这些组件必须能够在目标环境中有效执行，并能够正确解决应用程序出现的问题。 模型驱动设计(Model-Driven Design)抛弃了分裂分析模型与设计的做法，使用单一的模型来满足这两方面的要求。这就是领域模型。 单一的领域模型同时满足分析原

型和软件设计，如果一个模型实现时不实用，重新寻找新模型。如果模型没有忠实表达领域关键概念时，也必须重新寻找新的模型。 建模和设计成为单个迭代循环。将领域模型和设计紧密联系。因此，建模专家必须懂设计，会编程。 分层架构 最初层次只分为三层：表现层、业务层和持久层；DDD其实告诉我们如何让实现业务层！ 一位网友曾经请教层次的职责，对服务Service提出疑问。根据Eric的理论，业务层将细分为两个层次：应用层和领域层。它们的定义是：应用层：定义软件可以完成的工作，并且指挥具有丰富含义的领域对象来解决问题，保持精练；不包括业务规则或知识，无业务情况的状态； 领域层：负责表示业务概念、业务状态的信息和业务规则，是业务软件核心。 层次之间必须清晰分离，每个层都是内聚的，并且只依赖它的下层，为了实现各层的最大解耦，Ioc模式和Ioc容器是目前最好的选择
，JdonFramework使用基于PicoContainer的Ioc容器实现了各层的松耦合； Eric特别指出：那种将业务逻辑交由业务界面处理的快速UI方式是旁门左道。希望象C/S结构那样可视化拖拖图形就完成的软件开发是一种错误的方向，开发时快速，难于维护和扩展，虽然使用J2EE技术，其实是一种伪多层技术。可惜，有很多国人在疯狂开发这类工具，大有不撞南墙不低头之势，并且疯狂误导很多非专业人士，可悲可叹！如果对这段言论持不同意见，建议你购买"领域驱动设计"这本译书，见P53页。 领域模型种类 传统模型分为两种：实体（Entity）和值对象（Value Object），现在服务（Service）成为第三种模型元素。 实体（Entity）定义：通过一系列连续性（continuity）和标识（identity ID）来定义；个人认为它和分

析领域的四色原型中的PPT原型非常类似，可以看成是PPT原型延续。