当前版本:v1.0.0

当你获取到此文档即代表你同意此链接声明的条例，如果不同意请删除此文档。
往后本人翻译的所有文档链接都会分享到这里。所有文档的版本都会按照语义化版本的约定，即【主版本.次版本.修订号】，实际情况不需要修订号。如果文档有错误请指出，发邮件到我邮箱，mailto:806846483@qq.com，邮件主题请按照如下格式：
邮件主体：文件名-版本-页码
邮件正文：指出是什么错，如果让你翻译，你会怎么翻译，如果有参考链接（官方的链接？），请一并附上。
如果是实践结果和文档描述不相符合，请描述清楚你的实践过程，我会尽可能再次重试你的实践过程。
另外，需要队友一起翻译技术文档，希望组个翻译社，看完一个文档不够屌，来一起翻译和探讨吧，独乐乐不如众乐乐，希望你有时间并且能坚持下去。有想法的可以加本人微信。

领域驱动设计-什么是领域驱动设计和怎么使用它

进一步扩展前面我们讨论的面向对象分析和设计（OOAD），这篇文章讨论领域驱动设计(DDD)，DDD 是建立在面向对象分析设计上开发软件的一种方法。

通过这篇文章我们解释什么是领域驱动设计，在现代开发周期中如何实现，使用 DDD 的优点和缺点。

什么是领域

定义 DDD 之前我们首先必须要说明在开发中"领域"的含义。领域在字典中的解释是：“活动或者知识的范围”，更深层次的来讲，软件工程中领域指的是软件应用的地方。
换句话说,在软件开发中,领域指的是与"应用程序逻辑范围的知识和活动"

另一个在软件开发中常使用的术语是领域层或领域逻辑，对于开发者来说，说成是业务逻辑或许应该会更加熟悉。应用程序业务逻辑指的是业务对如何与其他业务对象交互（如何生成对象，修改相关数据）这种更高级别的规则。

什么是领域驱动设计

最先介绍领域驱动设计的是在程序员 **Eric Evans** 2004 年出版的《领域驱动设计:复杂软件核心复杂应对之道（Domain-Driven Design: Tackling Complexity in the Heart of Software）》中，领域驱动设计是领域概念的扩展和应用，并且将它应用在软件开发中。它的目标是将软件相关部分连接到不断发展的模型中，以此更容易创建复杂的应用，DDD 关注三个核心点：

.关注核心领域和核心领域逻辑。

.在领域模型中进行复杂性设计。

.与领域专家紧密合作，以此改进应用模型和解决新出现的领域问题。

Evans 的《领域驱动设计：复杂软件核心复杂性解决之道》一书中定义了几个常用的术语，在实践 DDD 和讨论 DDD 的时候非常有用。

.CONTEXT(上下文)：单词或者语句出现的环境，它决定了单词或语句的含义。关于模型的语句只能在上下文中理解。

.MODEL(模型)：一个系统的抽象，用于描述领域某个方面，并且能够用于解决领域相关的问题。

.UBIQUITOUS LANGUAGE(统一语言)：与领域模型相关的结构化语言，用于将团队成员的活动与软件连接起来。

.BOUNDED CONTEXT(上下文边界)：模型定义范围和适用范围的描述（比如，子系统，特定团队的工作）。

构建块

领域驱动设计同样也定义了几个连接领域模型的高层次概念，以此来修改，创建领域模型。

.ENTITY（实体）：连续状态变化的对象，而不是传统使用属性来定义的对象。

.VALUE OBJECT（值对象）：一个不可变有属性的对象，但是它没有唯一的标识符。

.DOMAIN EVENT（领域事件）：系统内记录与模型活动相关的分离事件的对象，系统内所有的事件都应该能够被跟踪，一个领域事件仅被领域专家关心的事件类型创建。

.AGGREGATE（聚合）：根据组边界定义值对象和实体的聚合，而不是允许单个实体或者值对象执行它自己所有的动作，聚合的对象都有一个统一的根对象（在书籍中写的是选择一个实体作为根），这样,外部对象不再直接访问聚合内部的单个对象或者实体，而是直接访问单一的聚合根对象，并且使用这个对象将指令传递给对应的分组。这个实践和我们在设计模式编码相关联

.SERVICE（服务）：本质上是来说，一个服务就是一个操作或者业务逻辑的组合，这样就表明了它在对象领域中不适用。换句话来说，如果某些功能必须存在且不能和实体或者值对象相关联，它可以定义为服务。

.REPOSITORIES（仓库）：不要和常见的版本控制仓库相混淆，仓库在 DDD 里面的意思就是一个服务，它提供一个全局接口来访问特定聚合内部所有的实体类和值对象。应该包括创建，修改，删除聚合内部对象的方法。然而，通过使用仓库服务来构造数据查询的目的是删除业务逻辑对象模型中的数据查询方法。

.FACTORIES（工厂）：正如我们在设计模式文章里面讨论的那样，DDD 建议使用工厂来创建复杂对象和聚合，保证客户端不用知道对象内部组成（透明性）。

同样，DDD 也着重强调越来越流行的持续集成实践，它要求所有开发团队使用同一个仓库共享代码，并且每天推送代码到仓库。在每天结束的时候自动检查代码仓库完，运行单元测试，回归测试等过程。这样就可以快速检测出潜在存在的问题并在下一次提交代码的时解决这个问题。

领域驱动设计优点

.沟通简单：团队成员使用与领域模型相关的统一语言来沟通会更加容易。通常来说，在讨论应用程序时 DDD 使用更少的技术行业术语，因为在先前建立的统一语言定义了更简单的术语来指代哪些更具有技术型的术语。

.提高灵活性：DDD 基于面向对象分析和设计相关的概念，几乎领域模型内的任何东西都基于对象，因此十分便于分模块。这就可以对各个组件，整个系统作出持续性的修改。

.在接口上强调领域：DDD 是围绕领域概念和领域专家建议进行构建的实践活动，与哪些首先强调 UI/UX 的应用程序不同，DDD 总是会生成适合当前领域的应用程序。虽然需要明显的平衡，但是聚焦于 DDD 意味者能够产生一个与该领域用户有共鸣的产品。

领域驱动设计的缺点

.需要精力充沛的领域专家：即使有最精通技术的开发人员，如果团队内没有一个知道应用程序使用领域相关的领域专家，那也是没有意义的。在某些情形下，领域驱动设计需要一个或多个外部人员在整个软件开发生命周期中扮演领域专家的角色。

.鼓励迭代实践：虽然许多人觉得这是一个优势，不可否认，DDD 实践强烈依赖连续迭代和持续集成来构建易于修改的项目。某些团队在实践这个的时候可能会遇到问题，特别是那些过去经验与不太灵活的开发模型有关，比如瀑布模型。

.不适用偏向技术型的项目：DDD 适用于领域复杂的应用(业务逻辑复杂)，它不适用于边界复杂的领域，类似这种领域都有高的技术复杂性。DDD 着重强调需要领域专家生成正确的统一语言并且一起写出项目的领域模型，但是领域专家来极难把握具有极高技术复杂性的项目，因此可能导致全体团队成员没有完全理解技术上的要求和限制。

[Domain-Driven Design – What is it and how do you use it?]