什么是DDD

题目难度：★★★

知识点标签：领域驱动模型

学习时长：20分钟

题目描述

了解DDD吗，简单描述一下

解题思路

> 面试官问题可以从几个方面来回答：什么是领域驱动模型，与传统软件开发模式的不同

什么是DDD(领域驱动模型)

2004年Eric Evans 发表《领域驱动设计——软件核心复杂性应对之道》（Domain-Driven Design – Tackling Complexity in the Heart of Software），简称Evans DDD，领域驱动设计思想进入软件开发者的视野。领域驱动设计分为两个阶段：

- 1、以一种领域专家、设计人员、开发人员都能理解的通用语言作为相互交流的工具，在交流的过程中发现领域概念，然后将这些概念设计成一个领域模型；
- 2、由领域模型驱动软件设计，用代码来实现该领域模型；

简单地说，软件开发不是一蹴而就的事情，我们不可能在不了解产品（或行业领域）的前提下进行软件开发，在开发前，通常需要进行大量的业务知识梳理，而后到达软件设计的层面，最后才是开发。而在业务知识梳理的过程中，我们必然会形成某个领域知识，根据领域知识来一步步驱动软件设计，就是领域驱动设计的基本概念。而领域驱动设计的核心就在于建立正确的领域驱动模型。

传统软件开发与贫血模型

传统开发四层架构：View，Service，Dao，Model

在传统模型中，对象是数据的载体，只有简单的getter/setter方法，没有行为。以数据为中心，以数据库ER设计作驱动。分层架构在这种开发模式下，可以理解为是对数据移动、处理和实现的过程。

业务逻辑都是写在Service中的，实体类充其量只是个数据载体，没有任何行为，是一种贫血模型。**简单的业务系统采用这种贫血模型和过程化设计是没有问题的，**但在业务逻辑复杂了，业务逻辑、状态会散落到在大量方法中，原本的代码意图会渐渐不明确，**我们将这种情况称为由贫血症引起的失忆症**。

传统架构的特点：

- a. 以数据库为中心
- b. 贫血模型
- c. 业务逻辑散落在大量的方法中
- d. 当系统越来越复杂时，开发时间指数增长，维护成本很高

DDD设计思想

采用DDD的设计思想，业务逻辑不再集中在几个大型的类上，而是由大量相对小的领域对象(类)组成，这些类具备自己的状态和行为，每个类是相对完整的独立体，并与现实领域的业务对象映射。领域模型就是由这样许多的细粒度的类组成。

建立领域知识(Build Domain Model)

说了这么多领域模型的概念，到底什么是领域模型呢？作为一个软件开发者，我们很难在对一个领域不了解的情况下着手开发，所以我们首先需要和领域专家沟通，建立领域只是。以飞机航行为例子：

现要为航空公司开发一款能够为飞机提供导航，保证无路线冲突监控软件。那我们应该从哪里开始下手呢？根据DDD的思路，我们第一步是`建立领域知识`：作为平时管理和维护机场飞行秩序的工作人员来说，他们自然就是这个领域的专家，我们第一个目标就是与他们沟通，也许我们并不能从中获取所有想要的知识，但至少可以筛选出主要的内容和元素。你可能会听到诸如起飞，着陆，飞行冲突，延误等领域名词，让们从一个简单的例子开始：

- 起点－>飞机－>终点

这个模型很直接，但有点过于简单，因为我们无法看出飞机在空中做了什么，也无法得知飞机怎么从起点到的终点，那么如此似乎会好些：

- 飞机－>路线－>起点/终点

  既然点构成线，那何不：

- 飞机－>路线－>points（含起点，终点）

  这个过程，是我们不断建立领域知识的过程，其中的重点就是寻找领域专家频繁沟通，从中提炼必要领域元素。

通用语言(Ubiquitous Language)

上面的例子的确看起来简单，但过程并非容易：我们（开发人员）和领域专家在沟通的过程中是存在天然屏障的：我们满脑子都是类，方法，设计模式，算法，继承，封装，多态，如何面向对象等等；这些领域专家是不懂的，他们只知道飞机故障，经纬度，航班路线等专业术语。

所以，在建立领域知识的时候，我们（开发人员和领域专家）必须要交换知识，知识的范围范围涉及领域模型的各个元素，如果一方对模型的描述令对方感到困惑，那么应该立刻换一种描述方式，直到双方都能够接受并且理解为止。在这一过程中，就需要建立一种通用语言，作为开发人员和领域专家的沟通桥梁。

可如何形成这种通用语言呢？其实答案并不唯一，确切的说也没有什么标准答案。

(a) UML

利用UML可以清晰的表现类，并且展示它们之间的关系。但是一旦聚合关系复杂，UML叶子节点将会变的十分庞大，可能就没有那么直观易懂了。最重要的是，它无法精确的描述类的行为。为了弥补这种缺陷，可以为具体的行为部分补充必要说明（可以是标签或者文档），但这往往又很耗时，而且更新维护起来十分不便。

(b) 文档/绘图

文档耗时很长，可能不久就要变化，为模型从一开 始到它达到比较稳定的状态会发生很多次变化， 可能在完成之前它们就已经作废了。对于复杂系统，绘图容易混乱。

(c) 伪代码

极限编程推荐这么做，但是使用难度大

总结

领域驱动设计的核心是**领域模型**，这一方法论可以通俗的理解为先找到业务中的领域模型，以领域模型为中心驱动项目的开发。而领域模型的设计精髓**在于面向对象分析，在于对事物的抽象能力，**一个领域驱动架构师必然是一个面向对象分析的大师。

在面向对象编程中讲究封装，讲究设计低耦合，高内聚的类。而对于一个软件工程来讲，仅仅只靠类的设计是不够的，我们需要把紧密联系在一起的业务设计为一个领域模型，让领域模型内部隐藏一些细节，这样一来领域模型和领域模型之间的关系就会变得简单。这一思想有效的降低了复杂的业务之间千丝万缕的耦合关系。