01 | DDD小传：领域驱动设计为什么这么火？

2022-12-06 钟敬 来自北京

《手把手教你落地DDD》

讲述：钟敬

时长 18:14　大小 16.65M

你好，我是钟敬。

今天咱们正式开始学习领域驱动设计（DDD）。

虽然 DDD 在这几年越来越流行，但是对于它的一些基本问题，业界仍然有很多不同看法。

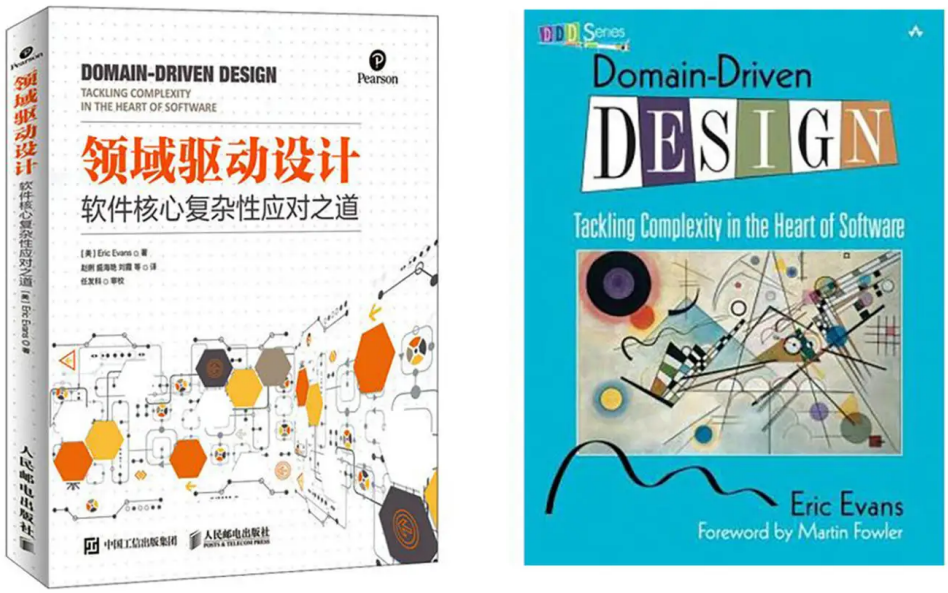
有人说，DDD 是划时代的创新；但也有人说，DDD 只是新瓶装旧酒，毫无新意；有人认为，DDD 仅仅是为了开发微服务；也有人认为，DDD 只是一种面向对象编程的方法；还有人以为，DDD 只是少数高手的专利。

这些看法都是片面的，说明还有不少人并没有真正理解 DDD 的基本概念。对基本概念缺乏了解，必然会影响对 DDD 的理解和学习。

所以，这节课咱们先聊聊 DDD 是什么，DDD 的来源，DDD 解决了什么问题以及 DDD 在这几年流行起来的原因。了解了这些，相信你对上面的问题就会有一个正确的认识，也会为后续的学习打下基础。

DDD 说的是什么？

2003 年，Eric Evans 写了《领域驱动设计：软件核心复杂性应对之道》一书，正式提出了这种方法。领域驱动设计的英文是 Domain-Driven Design，所以简称 DDD。下面就是这本书的中英文封面：



《领域驱动设计：软件核心复杂性应对之道》的中英文封面

按照作者自己的说法，“DDD 是一种开发复杂软件的方法”。

为了把意思说得更透彻，我把作者的原话做一个扩展：“DDD 是一种开发**复杂**软件的**系统化**的**方法学**和**思想**”。咱们下面掰一下这句话。

首先，到底**什么是方法学？**

方法学的英文名是 methodology。这么说吧，假如你 Java 代码写得特别溜，那么可以说你掌握了面向对象的**编程方法**；假如你还很熟悉面向对象的设计原则，掌握很多设计模式，那可以说你懂面向对象的**设计方法**；假如你能为业务概念构建领域模型，那么你就懂了面向对象的**分**

析方法。面向对象的分析、设计、编码三种方法融会贯通，成为一个有机的整体，这个叫面向对象的方法学。

我们发现，很多小伙伴都能熟练掌握编程方法，其中一些小伙伴，能掌握设计方法，但掌握分析方法的，就很少了。而分析方法，或者说领域建模的方法，正是 DDD 的重点。关于什么是领域建模，咱们在后面的课程中还会详细讲。

刚才说，DDD 是**系统化**的方法学。那么**什么是系统化呢**？

咱们用数学举个例子。古代数学家祖冲之花了一辈子时间，用“割圆术”把圆周率算到小数点后 7 位，在当时已经是世界领先了。而现在的大学生，只要掌握了微积分，用一天的时间算出来的圆周率，就比祖冲之一辈子算出来的还准。

那么是不是可以说，现在随便一个大学生，智商都比祖冲之高呢？这可不见得。之所以我们算得比祖冲之准，是因为前辈们总结了微积分这套方法，我们站在了巨人的肩膀上才能做到。微积分，就是解决高等数学问题的系统化的方法。

所以，系统化方法的作用在于，提供了一套相对容易的步骤，能够使我们这些中等智商的人，也能做到原来高智商的人才能做到的事情，从而让你能够省出时间和脑力，来探索更复杂的问题。在软件开发领域，DDD 就是这样一套系统化的方法学。

不过，据说有人问过 Evans 本人：“DDD 是一种方法学吗？” Evans 说，DDD 不只是一种方法学，更重要的是背后的一套开发软件的思想和哲学。那么，**DDD 中又蕴含着哪些思想和哲学呢？**

归纳一下，大概有构建知识、分而治之、抓大放小、统一语言、抽象化、可视化、协作以及演进等等。凡是说到思想，都是一些大词儿，听起来都对，但是感觉好像没什么用，关键是不知道怎么落地。

而 DDD 恰恰是通过总结了一套系统化的方法学，能够把这些大词落地。具体做法，我们会在后续的课程中讲解。

DDD 的来源

为了让你进一步理解 DDD 的含义，我们再了解一下 DDD 的来源。

按照作者在原书中的说法，DDD 是来自面向对象的方法学和敏捷软件开发。DDD 对它们进行了总结和提炼，使之更容易学习和实践。

业界有一句话 “DDD 就是 OO Done right”。OO 就是面向对象，也就是说把面向对象做对了，就是 DDD。也可以反过来说，面向对象本来就是“领域驱动”的。

不过，面向对象方法学从 80 年代兴起，到了 2000 年前后已经走向成熟。那么问题就来了：既然有了面向对象，我们为什么还要提 DDD 呢？这是因为传统的面向对象方法学仍然存在一些问题。

传统面向对象方法学的问题

早期面向对象的成功，主要是在几个特定的领域，比如计算机语言、图形用户界面、办公自动化软件等等，但在企业应用方面还没有取得成功。所谓企业应用，包括像银行的贷款系统、保险公司的理赔系统、电信公司的计费系统等等。

那个时候的面向对象方法学还不能很好地应用于企业应用，大体上有以下几个原因。

第一个原因是，很多开发人员走了一条只重技术不重业务的弯路。企业应用是用来解决业务问题的，所以我们应该首先把业务研究清楚，再通过技术手段来实现。但很多开发人员把主要精力放到技术的研究上，比如语言、框架、工具等等。以为把技术学会了，自然就能把系统开发好。

重技术、不重业务的思想造成了业务和技术人员之间难以相互理解，技术人员难以真正满足业务需求，如果连需求的方向都搞错了，那么技术再纯熟也会南辕北辙。这样的弯路走了十几年，早期面向对象方法学的大好形势也就烟消云散了。

第二个原因是，围绕业务进行开发的方法本身就不好学。面向对象方法学主要是围绕领域建模开展的。领域建模这个东西，看老师傅做的时候好像挺简单，新手一上去，怎么做都不对。

这是因为，领域建模是一种“手艺”。凡是手艺，都不是看看书、学学理论就能掌握的，而是要经过实践中的磨炼。很多小伙伴听了 DDD 的课，也看了 DDD 的书，但是没法在实践中真正解决复杂的问题，就是因为没掌握这门手艺。

第三个原因是，早期面向对象方法学主要考虑的是建模技术，很少考虑协作问题。历史上很多伟大的软件开始时都是个人作品，比如 UNIX 和 C 语言。作者在一开始是写给自己用的，所以并不存在协作问题。但是企业应用则不同，多数都是团队作战。即使只有一线开发人员，也免不了和需求方打交道。所以协作变得很重要。

最后一个原因是难以适应变化。企业应用的需求往往变化频繁，很多变化根本无法预料。传统的面向对象方法学也很少讨论怎样应对变化的需求。

DDD 的解决之道

可以说，DDD 正是为了解决上面这些问题而提出的。首先你从“领域驱动设计”这个名字就可以看出来。“领域”指的就是软件系统要解决的业务问题，也可以叫“业务领域”。用领域来驱动设计，就是说要从业务出发进行系统的设计。强调这个原则，就是希望把开发者从只重技术的弯路上拉回来。

要搞清业务，就要学会领域建模。前面也说了，这并不好学，那怎么办？DDD 采用了“模式”的方法。

我们都知道，老专家做事比新手做得好，说明专家心里一定有些新手不知道的东西。这些东西以前不容易讲清楚，后来有人把专家心里的这些高招都进行了系统化的梳理，每一条都是解决特定难题的通用的解决方案，这就是所谓“模式”。

计算机界最有名的是设计模式，后来又有分析模式、架构模式等等。读这些讲模式的书，就好像同时有好几个大师在给你讲课一样。通过学习模式，原来不容易掌握的东西就变得相对容易了。

Eric Evans 正是对面向对象方法学和敏捷软件开发方法进行了提炼，总结出了一套围绕领域建模进行软件开发的模式，一共有四十多个。这些模式成为《领域驱动设计》这本书的主体。

除了模式，书里还有一套相关的原则和实践。其中最基础的模式包括：模型驱动设计，实体、值对象等等。这些在后续的课程中，咱们都会讲清楚。

另一方面，领域驱动设计非常强调业务人员和技术人员要一起协作进行领域建模，在这个过程中提炼领域知识。和协作密切相关的模式有通用语言、模型驱动设计、限界上下文等等。实际上，这几个模式贯穿了整个 DDD。

此外，DDD 提出了所谓“柔性设计”的概念，使得模型和系统可以随着需求的变化而演进。什么意思呢？书里打了一个比方：这就好比一副新的皮手套，开始时整副手套都很僵硬；戴得久了，关节处就会自然变得柔软，而其他部分还是比较硬的。

同样，在软件设计中，也不是一开始就把所有地方都设计得很灵活，而是先进行“足够的”和“整洁的”设计。

随着业务变化，将变化频繁的部分重构得越来越灵活，而不常变化的部分则保持不变。也就是说，模型中的哪些部分需要设计得灵活，是自然演进形成的，这样就避免了“过度设计”。这个过程就是柔性设计。而这个重构的过程，也是不断加深领域知识理解的过程。

关于协作和演进，正是 DDD 的来源之一，敏捷软件开发所解决的重点问题。

从“沉寂”到“爆火”

看到这里，你是不是觉得 DDD 还不错呀？不过，《领域驱动设计》这本书其实是 2003 年写的，到现在将近 20 年了。直到最近几年，才真正开始普及起来。

你可能会问了，既然 DDD 这么好，为什么之前不火，现在才火起来呢？

其实，在 DDD 刚出现的时候，很多企业软件还不太复杂。一些复杂的软件，变化也不像现在这样频繁。甚至还有一些企业，干脆每隔四五年把原来的系统推翻重建一次。

另一方面，当时一些新兴的产业，例如互联网，还处在跑马圈地、野蛮生长的阶段。这时关注的是系统快速上线，抢占市场，至于软件质量好不好，容不容易维护，暂时不是考虑的重点。

在这种情况下，DDD 就成了一种“屠龙术”。杀龙是一项比较困难的技术，但是就算学会了，全世界也找不到几条龙，所以屠龙术就显得没有什么用处。DDD 就面临着这样的尴尬，或者说，从整个业界来看，必要性还不够强。

另一方面，DDD 普及的一些前提条件也还没准备好。

首先是敏捷软件开发刚刚出现不久，还不普及。如果没有迭代开发、持续重构、测试驱动、持续集成等敏捷实践的支持，构建良好的领域模型并在代码上落地是很困难的。

其次是配套的开发框架还不成熟。那时 J2EE 还被认为是企业应用事实上的标准，而基于这种框架开发程序，是很难和 DDD 的领域模型相衔接的。2004 年，Spring 发布了 1.0 版，从技术上基本解决了 EJB 的问题，理论上可以比较好地支持 DDD。然而 Spring 的真正普及，还要假以时日。

那么 DDD 为什么在这几年又火起来了呢？

首先是，数字化时代的到来，使 DDD 变得非常有必要。

数字化时代，技术逐渐成为企业核心竞争力的主要因素，无论业务还是系统都变得更加复杂。因此，如何将业务和技术融为一体，就成了很多企业的主要问题，而这正是 DDD 的主要优势。

行业竞争的加剧也要求系统具有更好的用户体验、更高的质量、更快地满足变化的需求。这些问题很难解决，必须引入系统化的方法。再说了，云计算、微服务等新技术架构的产生，也需要方法学的支持。可以说，“恶龙”已经遍地都是，“屠龙术”终于有了用武之地。

第二，DDD 普及的道路已经铺好，这项技术逐渐变得可行。

现在，敏捷软件开发已经普及。迭代、演进、协作等思想已经深入人心。DevOps 技术应用得也日益广泛。而且 Spring boot 等轻量级框架已经得到广泛使用。这些框架支持了领域模型与具体技术的关注点分离，使开发人员从技术细节中解放出来，将更多的精力投入到领域逻辑本身的分析和设计。

再者，相关的架构实践也已经研究得比较透彻，像整洁架构、事件驱动架构以及 CQRS 等等，都有力地支持了 DDD 的落地实施。DDD 本身也在不断完善，比如补充了像领域事件等新的模式，出现了事件风暴等新的实践。

可以说是天时地利人和皆备，DDD 终于咸鱼翻身、星火燎原了。

总结

好，现在我们来总结一下。

今天我们介绍了 DDD 的概念、内容，以及近年来兴起的原因。总的来说，DDD 是一种开发**复杂**软件的**系统化**的**方法学**和**思想。**

DDD 建立在面向对象方法学和敏捷软件开发方法之上，一方面保留了面向对象的精华，另一方面又弥补了早期方法的不足。

DDD 从面向对象和敏捷中提炼出了一套原则、模式和实践，使面向对象方法学在企业应用中更加容易学习和掌握。

DDD 的核心是领域建模。领域模型是浓缩的领域知识。此外，DDD 还重视业务与技术人员的沟通，以及如何应对变化。

最后，我想说，数字化时代为软件开发带来了新的挑战。如何实现业技融合，如何应对复杂多变的需求，如何防止架构和代码的腐化等问题，需要新的解决办法。而 DDD 正是顺应了时代的要求，才日益普及起来。

思考题

下面有两道思考题：

1. 有人说 DDD 的目的就是为了开发微服务，你同意这种说法吗，为什么？

2. 有人说 DDD 是创新，有人说不是，你的看法是怎样的呢？

欢迎在评论区分享你的想法。下一节课，我们将从实战中体会 DDD 的魅力，开启第一个迭代。

精选留言 (18)

写留言

业余草 置顶

2022-12-06 来自广东

尽信DDD不如不用DDD。

DDD不能算一门完善的理论，它是出自工程师之手，目前还只能算是经验的总结和模式的梳理（当然，DDD思想也在不断完善中，例如“事件风暴建模法”是后面才引入到DDD核心内容中的），它能解决的问题是针对业务复杂性如何优化软件的架构。

对于性能问题以及代码细节上的结构问题，它并不能提供足够的帮助。所以当我们在应用DDD的时候，如果发现它并没有把我们的代码变成“完美的”、“理想化”的代码，反而造成了一些麻烦，不妨审视一下，我们是否已经过渡依赖了DDD，对它寄予了过大的期望。

反之，如果以平常心来对待DDD，和Eric Evans一样，把它当成一套模式的集合，发现哪个模式对我们有用就拿来用一下，不能用也不要勉强(DDD的限界上下文、聚合、实体、值对象等等，都可以认为是一种模式)，以这种拿来主义的心态去使用DDD，反而可能会渐入佳境，也不会造成太大负担。

作者回复: “尽信DDD不如不用DDD”这句说的很棒。提到模式，说明您肯定是看过原书的。事件风暴虽然在课里面也讲得比较细，但可能还算不上DDD的核心。关于DDD对代码的影响，确实不会特别细，但可能比一般同学想象的要“细”。DDD和整洁代码结合，就非常好了。等课程讲到代码的时候，我们再探讨。

共 2 条评论 >

11

escray 神 置顶

2022-12-06 来自广东

其实我有一点怀疑 DDD 现在很”火“这个说法，在 Google trends 上可以看到 DDD 的热度不及 2004 年高峰时的一半，另外就是似乎在国内相对热度比较高。

还是打算认真的学习一下 DDD，看能否在理解业务和编写代码上有所帮助。

DDD 的目的显然不是为了开发微服务，可能两者背后有一些相同的想法，DDD 出现的早，而微服务大概要晚十多年的样子。DDD 是”开发复杂软件的系统化方法学和思想“，微服务显

然不是。

我觉的 DDD 可以算是创新，但是更有价值的可能是通过 DDD 的方法论建立起来的领域模型。

作者回复: 是的，DDD是方法学层面，微服务是架构风格层面。DDD抽象于具体的架构风格。DDD可以帮助设计微服务，但不仅仅用于微服务。领域模型确实是DDD中最有价值的部分之一。

1

骆驼、 置顶
2022-12-07 来自广东

1、我觉得DDD的出现是为了应对业务需求多变的场景，它将业务划分为不同领域，同一类业务形成一个领域聚集在一起，不同业务相互隔离，当需求变更时，只需要变动业务对应的领域部分即可，一般都是结合设计模式进行开发，保证系统的可维护性，另外在代码层面上体现在尽可能的减少代码之间的耦合，避免牵一发而动全身。而微服务在我看来是一种系统架构设计，他的出现是主要为了解决单体应用无法支撑高并发的业务场景，但是微服务如何划分一直没有一个明确的概念，而DDD领域划分的思想就与微服务的划分相一致，这就导致DDD是微服务划分的一种重要手段。但是这里面如果DDD领域划分的过于细致，就会导致微服务的数量就会增多，对于一般企业而言没有那么多的资源去维护这种情况，我也不知道该咋解决。
2、对于我来说DDD应该算是创新，毕竟工作5年了，之前接触过的项目一直采用的是MVC模式开发，很多地方一开始设计的可能很好，随着开发时间的推移，到后来却越来越难维护，有的时候改一行代码，其他的地方就出问题了，耦合性太强，结果不得不导致重新写代码了。而DDD就明确要求各领域之间相互隔离，不能直接引用，就自然而然的解耦了

作者回复: 很多要点您都说得不错。有一点要澄清一下，作为微服务划分依据的，是“限界上下文”而不是领域。等到后面学到限界上下文，我再说说它和领域、子域的关系。
关于您指出的微服务粒度过细的问题，确实很常见，这一点也会在后面讲限界上下文的时候讨论。现在先说一点，就是微服务的粒度可以参考团队的“认知边界”。您可以先琢磨琢磨。

Jxin 置顶
2022-12-06 来自广东

1.不同意。时间线上先后倒置，所以至少 Evans 不会是为了微服务而ddd；时代在变化，ddd的“目的”也可能变化（也许就是因为解决了微服务拆分这个问题才火起来的呢，哪怕跟初衷不一样了，但价值高呀）。但我不这么认为，因为哪怕是现在，也不是所有公司都适合引入微服务(技术基础不具备/组织结构不匹配/业务特性不需要), 但大部分业务领域（特殊规则非事件流的还是搞不定，比如特定的复杂计算公式）都可以用ddd来提炼和构建认知，两者间是解偶松散的。所以能够支撑微服务拆分可能是ddd被挖坟的起因，但不是它能站住脚的全部因素，ddd的目的就是为了开发为服务总归有些片面。

2.定个调，软件设计的创新就是提高长期迭代的效率。ddd目前能看到一些成果（至少咨询公司能卖了不是），所以能说它是一种创新。但这个成果不多，因为没有大杀器级别的软件产品来证实（仅靠咨询公司的瓶颈），所以创新得可能没有那么突出。

作者回复: 1.关于微服务的回答很到位呀。微服务是一种架构风格（或者说架构模式）；DDD是方法学层面，抽象于具体的架构风格。微服务固然用得上，单体也用得上，无服务器架构也成，或者将来才出现的某种架构风格，说不定也行。
2.你先给出了“创新”的评价标准，然后再评价DDD，说明了思维的严谨。等我们听听更多同学的意见，再归纳一下。

关于“挖坟”嘛，广东人更喜欢说“咸鱼翻生” 😊

老狗
2022-12-06 来自广东

1. DDD 是一系列相互关联的模式的“聚合” 应对复杂的业务场景，的确有助于微服务的划分，但是不限于微服务，复杂业务的系统都合适
2. DDD更多的是一种总结性创新，更很多操作和模式在DDD书前就有，但是有两样创新，一方面是成体系的把这些模式聚合起来，另一方面是的确也总结了诸如统一语言之类的新模式

作者回复: 正解！关于创新的问题，等多听点同学们的看法，再聊细一点。

共 2 条评论 >


2

aoe
2022-12-07 来自广东

《领域驱动设计：软件核心复杂性应对之道》开始没几页作者就说：好的模型是改出来的。
看到这里我想到了 TDD，不然没人敢改代码
从编码实现的角度看：没有 TDD 很难 DDD

作者回复: 最好再结合重构、持续集成、迭代开发等实践。


1

方勇(gopher)
2022-12-06 来自北京

小项目采用DDD会增加研发的心智负担

共 1 条评论 >


1

Geek_d1bbb0

2022-12-06 来自广东

1、感觉DDD是一种软件设计方法，微服务是一种软件架构方式

2、DDD是对过去软件设计方法的改良。

作者回复: 说的很好！

关于“软件设计方法”，我再补充两句。“设计”有广义狭义之分。狭义的设计，是与“分析”相对而言的，例如系统设计、架构设计、代码设计等等；广义的设计，包含了分析和狭义的设计，泛指给出一个需求，得到解决方案的过程。“领域驱动设计”中说的设计，是广义的设计。

1

喆

2022-12-10 来自辽宁

1. DDD能解决微服务当中的一些问题，比如：给服务的拆分提供一些理论依据。但它并不是专门给微服务用的，只是它提供的思想和模式，恰好能够解决微服务要解决的一些问题而已。

2.我认为DDD有他创新的地方，比如：他鼓励业务团队和技术团队共同进行领域建模，让业务团队与开发团队之间存在了一个有必要的交集，从而减少了软件需求从业务需求到技术落地过程中的信息损耗。但是他并不是一个翻天覆地的创新，本质上还是对面向对象的总结与提炼，提供一些固定的模式以解决一些特定的问题。如果具备良好面向对象设计的理论知识和丰富的面向对象的开发经验，没有DDD同样能够做出质量属性良好的软件。虽然不属于创新，DDD却将这些面向独享的理论知识和实践经验变成了一种可供交流学习的模式，这更像是面向对象的传道者。

3.我们公司目前正在在一个比较核心的项目上实践DDD，我对DDD抱有一定的期待。

吾真本

2022-12-10 来自广东

痛点中的“协作”问题，除了“开发人员与需求人员”之间的协作，是不是还包括“分析设计人员与开发人员”之间的协作？在有些企业，不写代码的架构师做分析与设计工作；之后交给程序员写代码；然后架构师就去做另一个系统的分析与设计。貌似人尽其才，但实则阻碍了从编码中发现分析和设计不合理的问题并做改进的实践。程序员发现分析和设计的问题，又不好打扰架构师，于是自作主张改动设计。有些改动合理，有些则不合理。最终结果就是分析设计与代码脱节，让前者只能挂在墙上。

絆

2022-12-09 来自广东

老师相关的数据和之前的DDD实战课也都去看过了解过, 在日常的代码也尝试着去做过, 但是很困难. 目前公司的业务主要以电商物联网,以及客户关系为主, 系统框架因为老系统的原因包

含了多种框架. 书中也提到过战略设计比战术设计更重要, 即便就现有的业务模块来反推做战略设计也是困难重重, 画出来的草图最后都变成了复杂的蜘蛛网.

作者回复: 这确实是很具体的实践难题了，您先看完迭代一一直到代码的部分，看有没有启发。关于从现有系统反推领域模型的事情，我们也经常做，算是比较高级的内容，在课程最后的一课会讲。关于战略还是战术重要，这要看项目具体要解决的问题，难以一概而论。

絆

2022-12-09 来自广东

包括之前看的 DDD实战课, 对于提高代码有很大帮助, 让自己的代码更接近面向对象编程, 而不是过程.

作者回复: 希望这门课也能帮到您，祝您成功！

Michael

2022-12-09 来自广东

我比较好奇DDD和四色建模法之间的异同，之前胡皓老师也提到建模相关，但是提四色建模比较多，很少提到DDD，所以在TW这几种建模方式是结合使用的么？

作者回复: 您这个问题很有意思。TW内部确实有人结合起来用。四色建模法来源于Coad 的《Java Modeling In Color With UML》，所以叫“彩色建模法”可能更符合原意。如果有时间，我后面会专门写一篇文章比较两者的关系。大体来说，彩色建模中的四种类型，都是DDD中说的实体，而不是值对象。“人地物”（party,place,thing）指先于具体业务活动而存在的事物，比如说我们例子中的员工、组织、客户等等；“时间性实体”（moment, interval）代表包含起止时间关注点的事物，一般是业务活动相关的实体，比如我们例子中的合同、项目；“描述性实体”（description），对应于我们一般说的字典性的表，其实有点微妙，因为和值对象有关系（本身不是值对象），比如我们课程后面才会讲到的“组织类别”；“角色”（role）说明实体在一个关联中充当什么，既可以是一个单独的实体，也可以“缩”成UML中的“角色”，这些也可以在后面课程讲完建模后再聊。

Leo_Fan

2022-12-08 来自广东

1、DDD不是微服务，微服务是DDD落地的一种方式

2、DDD 不是创新，更像是总结

3、支持钟老师，希望自己不会烂尾

作者回复: 您说的挺好。相信您一定能实现目标 ：）

李威
2022-12-08 来自广东

管他是不是创新，好使就得了。更好使了，那就是创新；不好使的话，叫“发明”都没用。

作者回复: 妥妥的“实用主义”呀 ：）

bighero
2022-12-07 来自湖北

1.DDD的目的：DDD出现应该追溯到2004年，那是个单体架构盛行的年代。很多逻辑处理不在应用层，而是在数据库层。而且面向对象也刚刚行成大气候，但是开发大项目单体业务应用，却还非常繁杂，逻辑复用性差等问题。DDD的出现主要是为了上浮业务逻辑从数据库层到应用层 达到高内聚低耦合目的而建立的一些业务领域的oo原则。

2.DDD它是一种实践方法论。不可否认它是一种创新。它的创新在老oo的基础上进行的一种创新方法论。它首次大胆的提出了限界上下文这一方法论，这是oo一切对象复杂网络关系的大胆切割。大有“庖丁解牛，目无全牛”的艺术与哲学感。如果说oo仅仅是技术，那么分了限界才算接近了“道”。也是“技近乎于道”的一种体现。又怎能不说是种创新创举。

码小呆
2022-12-07 来自广东

ddd 是什么还没明白,是一种思想?

作者回复: 一种思想以及一套方法。准确地说属于软件开发方法学层面。

特修斯之船
2022-12-07 来自广东

1. DDD的目的是对业务知识的封装和隔离
2. DDD是经验论，经验算是指导，还谈不上创新

作者回复: 看了不少大家对创新的界定了，您这又是一种，到时我们总结一下。