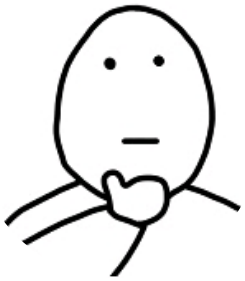


Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

April 8, 2018 · 原创文章

谈谈领域建模

目录

1 前言

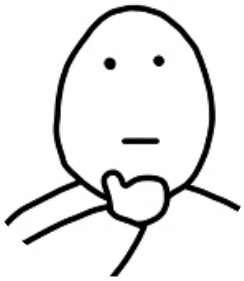
最近工作中，我对于模型的设计，尤其是“领域模型”的设计有一些思考和讨论。这篇文章总结了我对领域建模的看法和方法论，也相当于汇总了各位大师对领域建模的不同思想。

领域建模的目标，是解决复杂业务中软件开发的一系列问题。领域建模也是实现这个目标的一条路径，一种方法论。在实践的过程中可能有一种似曾相识的感觉：虽然没听过这个概念，但是这种做法很有道理而且有可能我们本身就在做。

需要注意的是，领域建模的方法有多种，甚至关于领域模型本身的定义也有一些模糊之处。不同的方法论和流派思路大体相似，在细节上还有一些区别。不过条条大路通罗马，他们没有对错之分。

本文更多表现的是我对领域建模的理解，也希望对各位有所启发。

2 什么是领域模型



Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

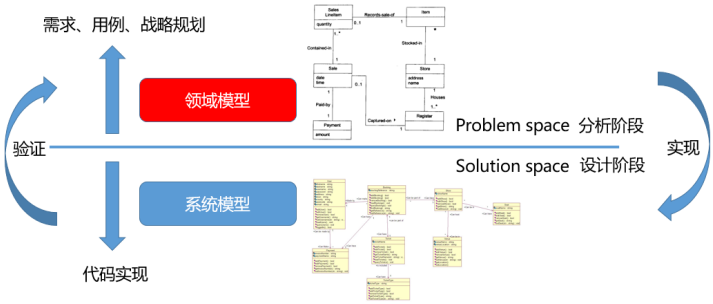
About

2.1 定义

领域模型，在本文中的定义源于《UML和模式应用》[1]，这本书对领域建模的概述是最完整、可操作性最强的。

> 领域模型(domain model)是对领域内的概念类或现实世界中对象的可视化表示。领域模型也成为概念模型、领域对象模型和分析对象模型。

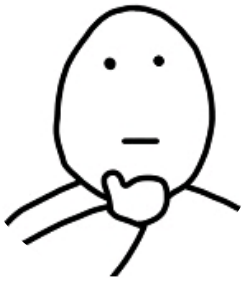
领域模型是一种概念模型，也叫问题域模型。它表述的是某个领域的现实概念。



2.2 与其他作者的定义的异同

本文提到的领域模型，基于C Larman在书中的定义。同时与其他作者定义的模型区别如下：

作者	定义	出处	异同
Martin Fowler	Conceptual model	Analysis Patterns: Reusable Object Models [3]	相同



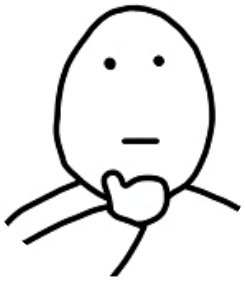
Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

作者	定义	出处	异同
Martin Fowler	Domain Model	Patterns of Enterprise Application Architecture [4]	不同。这里的领域模型指的是solution space的领域层的模型对象，也就是本文指的**系统模型**。其定义需要包括行为和数据的对象模型，即充血模型。



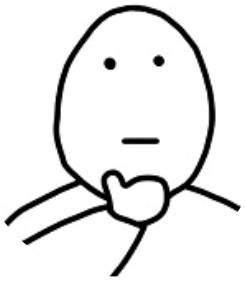
Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

作者	定义	出处	异同
Eric Evans	Model	Domain-Driven Design – Tackling Complexity in the Heart of Software [5]	不完全相同。这本经典的DDD理论，不再将分析模型和程序设计分离开，而是寻求一种能满足这两方面的单一模型。因此DDD的模型**既是领域模型，又是系统模型**。同时，通用语言也可以算作领域模型的一部分。
George Fairbanks	Domain Model	Just Enough Software Architecture [6]	相同



Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

作者	定义	出处	异同
Object Management Group	Computation Independent Model (CIM)	OMG. MDA Guide Version 1.0.1	相同
Grady Booch	Object-oriented analysis	Object-oriented Analysis and Design with Applications, 3rd edition [7]	相同。面向对象分析的产出物就是问题域模型。

其实可以看到，自从面向对象出现以来，出现了很多模型的分析和设计的方法（包括一些需求分析的方法）。其中有一点是共通的：程序设计要从问题域出发，用问题域分析出的模型来指导程序设计。

2.3 领域模型的特点

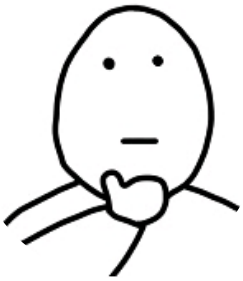
总结一下，领域模型具有如下的特点：

- 领域模型是业务概念的可视化描述，是需求分析的产物
- 领域模型用于指导程序设计，但领域模型与实现方式无关，领域建模时不应该考虑如何实现
- 领域模型需要同项目所有成员（客户、项目经理、开发、测试...）达成共识

3 为什么要做领域建模

首先，建模的重要性在所有工程实践中都已经得到了广泛的认同。建模是一种抽象和分解的方法，它可以将复杂的问题拆解成一个个抽象，代表了特定的一块密集而内聚的信息。[7]

上世纪80年代开始，人们对于面向对象建模产生了许多思考和方法，其中最流行的就是面向对象分析与设计[8]。面向对象分析，强调的是在问题域发



Yilun Fan

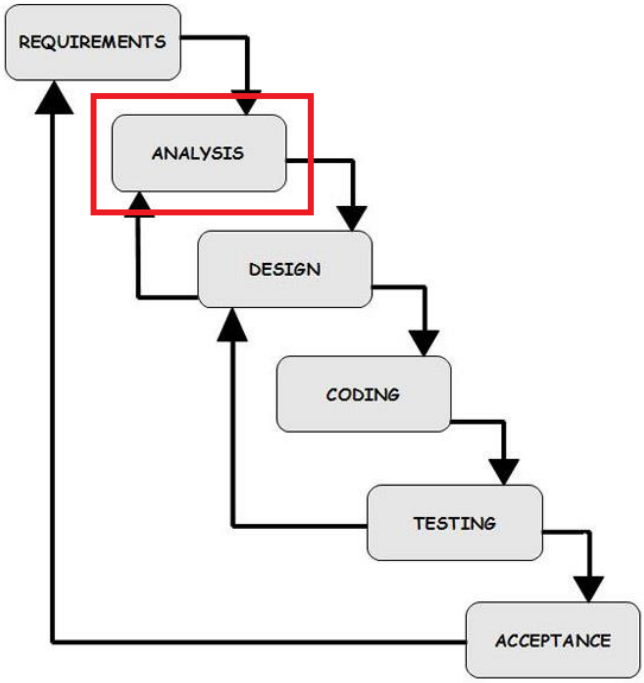
站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

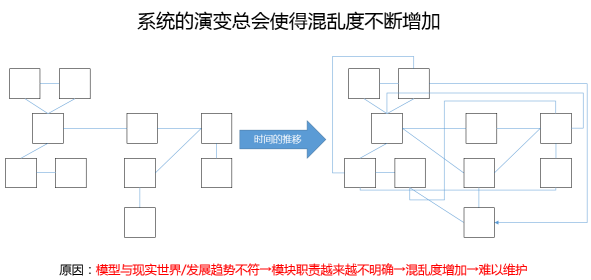
现并描述概念，解决的问题是**做正确的事情**。面向对象设计，强调的是定义软件对象，解决的问题是**正确的做事情**。

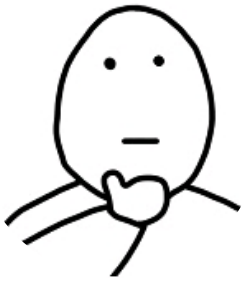
领域模型就是面向对象分析的主要产物，它表达了对现实问题的描述和抽象。



不过我们为什么要按照这种流程来进行开发呢？理论上说，如果不做分析和设计，你也可以直接去做代码实现（甚至使用面向过程的编程语言），一样可以完成软件的功能需求。

- 如果不做设计直接实现，俗称走一步看一步。很大可能在开发过程中发现思维局限，开发进度推倒重来。
- 如果不做分析直接设计，看起来没什么问题。遗憾的是，通过这种方式构造的代码，并没有和现实世界连接起来，当我们的软件和需求稍加修改，这份代码就可能变得异常混乱和难以维护。而通过领域建模的，自上而下的设计，可以保证代码实现的层次结构和模块划分是科学的、稳定的。





Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

- 举一个例子，在某电商系统的初期设计中，一个卖家账号只能开一个店铺，因此卖家和店铺的概念全部由卖家这一个模型来承载。所有店铺相关的模型（店铺红包、店铺评价等）全部与卖家模型做关联。这时，同一个模型拥有了两层业务含义，职责不明确。
- 再举一个例子，在我做的电商供应链管理系统中，对于箱规的定义起初并没有抽象出模型。随着系统更深入的发展，我们发现在多个模型里都维护着箱规，包括商品、货品甚至补货模型里都有着箱规的定义。此时各领域如果没有明确的职责划分，造成同一个模型在多个系统同时维护，势必会带来混乱。
- 想一下，按照上面的模型会有什么危害呢？

总结：

领域建模可以降低软件和现实世界之间的差异，用真实的业务概念划分职责，目的是实现一个可以高效低成本维护的可持续发展的软件系统。

从领域模型推导到系统实现是一套引导思考的方式，也是一套科学的开发流程。其核心目的在于提供了系统设计的“指导方针”。领域模型必须站在用户需求和业务发展的角度上，既可以用来同客户沟通验证需求，又可以避免模型因实现的考量而带偏（实现成本、遗留系统）。

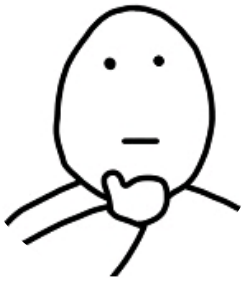
4 如何进行领域建模

同样，领域建模的方法也有很多种。下面列出的是一些常见的方法。需要注意，领域建模是需要依赖大量经验和思考的，各种方法起到的都是引导思路的作用：

4.1 用例分析法

用例分析法是进行领域建模最简单可行的方式。其步骤如下：

- 1.获取用例描述
  - 既然我们的领域模型指的是问题域模型，那么建模也一定要从问题域入手。那么问题域



Yilun Fan

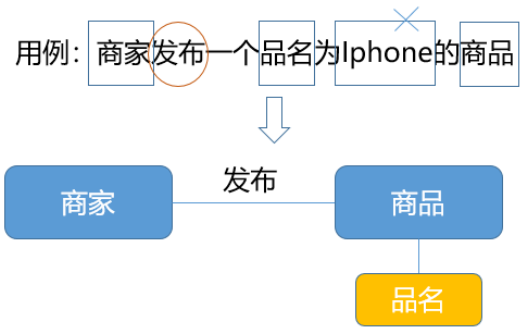
站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

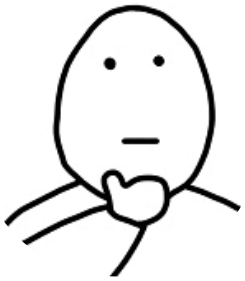
About

的知识如何表现出来呢，一个最常见的方式就是通过用例，也可以通过场景Scenario来分析——总之就是一段**格式化的需求文字描述**。

- 2.寻找概念类
  - 寻找概念类就是对获取的用例描述进行语言分析，**识别名词和名词短语**，将其作为候选的概念类。
  - 当然，需求描述中的名词不可能完全等价于概念类，自然语言中的同义词、多义词都需要在此处进行区分。还有很多名词可能只是概念类的属性，不过没关系，在这一步骤可以都提取出来，在第四步再区分出概念类和属性。
- 3.添加关联
  - 关联意味着两个模型之间存在语义联系，在用例中的表现通常为两个名词被动词连接起来：



  - 并非所有动词关联的概念类都需要作为关联存在，更重要的是我们需要判断，两个概念类的关系是否需要被记住：
    - 试想你是一个业务员，如果某两个概念类的实例之间的关系没有任何人知道，是否会阻碍业务的开展。如果答案是肯定的，那么说明这两个概念类存在关联。如果答案是否定的，那么建议不要加上关联（视情况，也要考虑逻辑上二者的关系是否“被记住”）。
    - 应该尽量避免加入大量关联
    - 关联不代表数据流，也不代表系统调用关系
- 4.添加属性



Yilun Fan

站内搜索

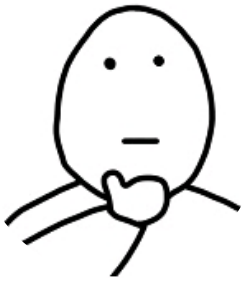
- **个人随笔**3
- **原创文章**16
- **杂谈**1
- **视频**2
- **读书笔记**5

About

  - 对于上文抽取到的名词列表，我们需要区分概念类和属性（当然名词列表也会有无用的词语）。
  - 如何判断一个名词是否是属性?
    - 能完全通过基本数据类型（数字、文本、日期）表达的大多是属性。
    - 如果一个名词只关联一个概念类，并且它自身没有属性，那么他就是另一个概念类的属性。
  - 注意这里表达的依然是业务概念，外键ID不是属性
- 5.*模型精化
  - 模型精化是可选的步骤，有时我们希望在领域模型中表达更多的信息，这时会利用一些新的手段来表达领域模型：包括泛化、组合、子域划分等
  - 领域模型可以使用UML的泛化和组合表达模型间的关系，他们表达的是概念类的”is-a”和”has-a”的关系，并不是实现的软件类的关系。然而，也一些方法论中并不建议使用这种方式来表达领域模型，因为这种精化的领域模型不利于和需求方沟通。
  - 子领域划分是常见的拆解领域的方式，通常来说我们会将更内聚的一组模型划分为一个子领域，形成更高一层的抽象。利于系统的表达和分工。

下面举个例子，内容来自《Object-Oriented Analysis from Textual Specifications》[9]论文，该文章讲述了如何通过自然语言分析来做面向对象分析(OOA)。

- 用例描述:
  - **Vendors** may be **sales employees** or **companies**. **Sales employees** receive a **basic wage** and a **commission**, whereas **companies** only receive a **commission**. Each **order** corresponds to one **vendor** only, and each **vendor** has made at least one **order**, which is identified by an **order number**. One **basic wage** may be



Yilun Fan

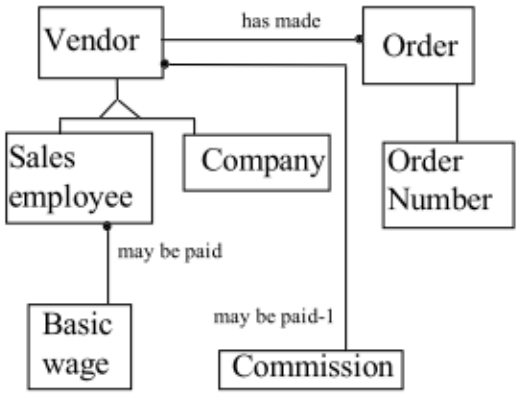
站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

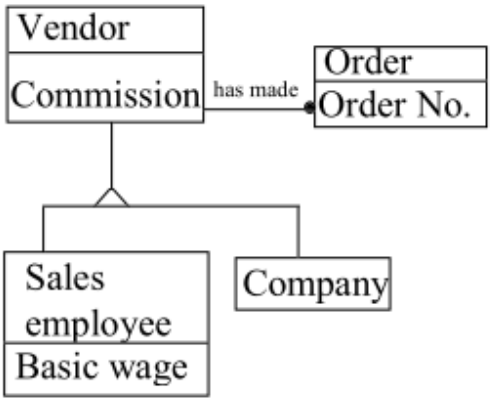
About

paid to several **sales employees**. The same **commission** may be paid to several **sales employees** and **companies**

- 如上，我们把所有名词标记出来：vendors, sales employees, companies, basic wage, commission, order, order number，他们作为概念类的候选类
- 接下来为他们添加关联，连接这些名词的动词会出现在关联的线上：



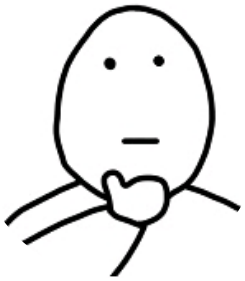
- 最后，为这些候选的概念类选择属性，在本例中，如果一个概念类只处于一个被动的关联关系中（如basic wage, commission, order number），那么它需要作为关联类的属性。



4.2 DDD的方法

Eric Evans的著作Domain-Driven Design领域驱动设计，简称DDD。DDD是一套软件开发方法论，用来解决复杂的现实问题。

DDD本身是一套完整、详尽的方法论，从如何需求沟通（构建领域知识），到高层设计（战略建模）、详细设计（战术建模），细致到代码的实现风



Yilun Fan

站内搜索

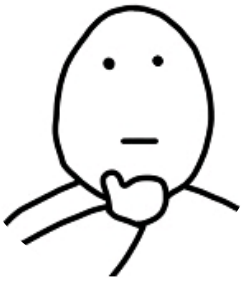
- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

格都给出了示例。本文无意也无力来详述DDD的所有知识，但是关于如何建模，DDD给出了很多思想可以借鉴。

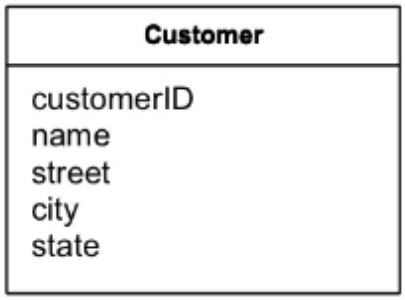
需要再次强调的是，DDD的模型本质上是solution space的模型，然而DDD强调模型与实现绑定，因此这里指的"模型"，可以说是领域模型，也可以是系统模型。下面就是DDD建模的一般步骤：

- 构建领域知识
  - 软件的最终目的是增进一个特定的领域。为了达到这个目的，软件需要跟要它服务的领域"和谐相处"。所谓和谐相处，软件需要精确地反应领域概念和知识，以更好的适应变化。
  - 因此，软件开发者第一步也是最重要的一步就是理解领域知识。DDD鼓励开发者和领域专家工作在一起，通过交谈和提问，让开发者学习到领域知识，挖掘出领域的关键概念。
- 创建通用语言
  - 通用语言是领域专家和开发团队之间定义的标准的术语。目的是把领域知识更完善地传达到软件中。
- 团队在进行所有方式的沟通时（文字，演讲，图形）都需要采用这种一致的语言。
  - 通用语言需要映射到模型中，映射到代码里。做到通用语言的更改就是对模型的更改，也是对代码的更改。
- 创建实体
  - 基于通用语言和领域知识，需要首先分辨出实体。
  - 实体是领域中**需要唯一标识**的领域概念。如果两个实体所有状态都一样，但标识不一样，就是两个不同的实体。
  - 实体同样需要属性来描述

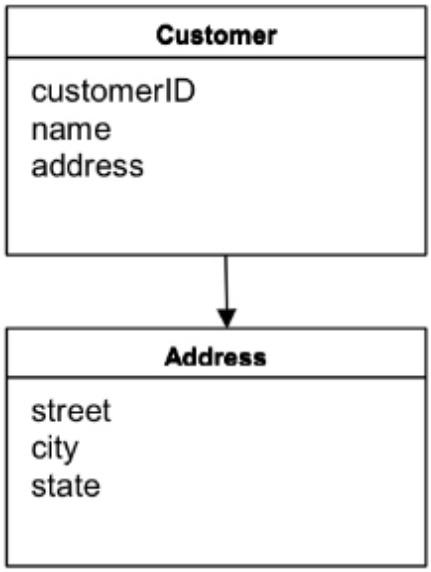


Yilun Fan

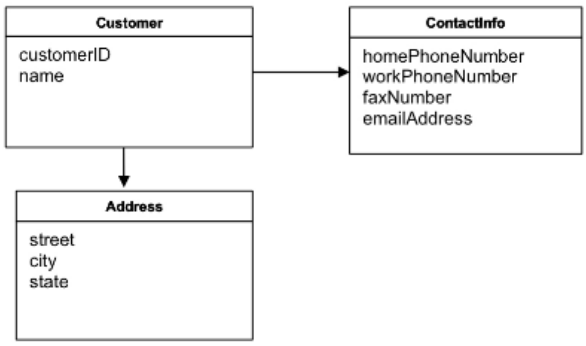
站内搜索



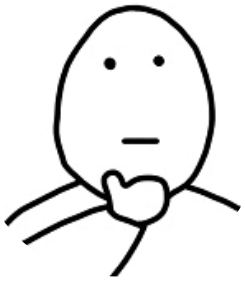
- 创建值对象
  - 值对象是领域中不需要唯一标识的领域概念。
  - 如果两份对象所有状态都一样，我们就认为是同一个值对象。值对象也可以理解为一组聚合的属性。例如地址信息，类目信息。



- 创建聚合根
  - 聚合根是一个实体，将一组模型聚合在一起，与外部模型划分开来。这一组模型全部关联着聚合根，只有聚合根负责与外部访问。
  - 聚合根有助于保持领域模型关联的简化和生命周期的维护。



4.3 四色建模法



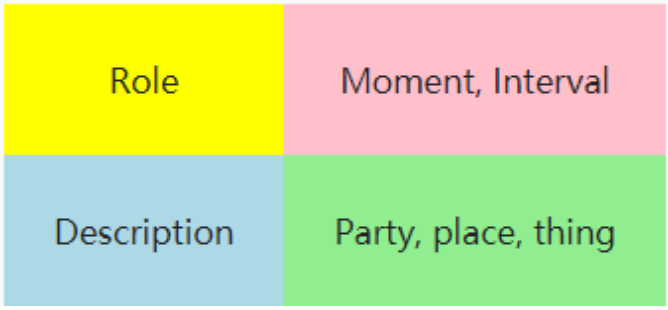
Yilun Fan

站内搜索

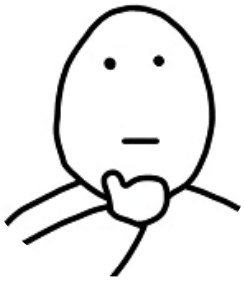
- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

四色建模法源于《Java Modeling In Color With UML》[10]，它是一种模型的分析和设计方法，通过把所有模型分为四种类型，帮助模型做到清晰、可追溯。



- 四色模型
  - moment-interval（时标性原型）：时标性原型是建模的起点，它代表着我们需要记录的，某一时刻发生的事件。例如订单，行程，会议。
  - party, place, or thing（人-事-物原型）：一种有形的，可唯一识别的实体。可以是人、机构、地点、物品等。
  - role （角色原型）：角色是party, place, or thing的一种参与方式。例如，在一份雇佣关系中，某个人扮演者雇员的角色。那么这个人就是”party, place, or thing”，雇员就是”role”。
  - description（描述原型）：表示资料类型的资源，是一种类似目录条目的描述，用来对对象进行分类或标记，可以被其它原型反复使用。例如，一个商品的品牌、描述属性。
- 建模次序[11]
  - 首先以满足管理和运营的需要为前提，寻找需要追溯的事件。
  - 根据这些需要追溯，寻找足迹以及相应的**时标性原型**。
  - 寻找时标对象周围的**人-事-物**。
  - 从中抽象**角色**。
  - 把一些信息用**描述**对象补足。



Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About

5 从领域模型到系统模型

提到模型，大多数人第一反应就是一个类或者对象，我这里指的系统模型就是这种概念。

为了保证程序实现能够遵循领域模型的思想，为了让所有人对领域和职责的认知没有偏差，我**强烈建议每个领域模型都要有一个系统模型与之对应**，最好能完全一一对应（DDD就是这么做的），他们的命名和属性也尽可能保持一致，使用相同的术语。

具体到系统模型的设计，就是面向对象设计的范畴了，这里可以使用各种各样的设计模式、GRASP、SOLID去设计和规划，本文就不在此展开。

需要遵循的宗旨是，领域模型的模型职责、子域边界划分应该作为此处设计的指导原则，在实现中的每个模块不可以突破这些职责约束。

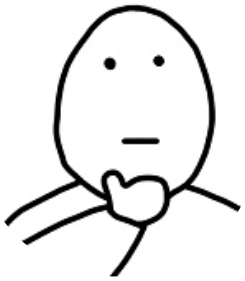
6 领域模型与系统架构

既然谈到了架构，我们先看看什么是架构，下面是ISO/IEC/IEEE 42010对软件架构的定义[12]:

> fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

可以看到有三个要素，elements，relationships，principles：

- element就是系统的组成成分。
  - 但实际上，系统可拆解的角度非常之多，可以是应用部署结构（物理架构），可以是进程，可以是微服务，可以是系统模型……
  - 在本文所指的业务系统架构中，更倾向于element指的是一个个系统模块，它囊括了若干系统模型，组成一个内聚的职责明确的模块



Yilun Fan

站内搜索

- **个人随笔**3
- **原创文章**16
- **杂谈**1
- **视频**2
- **读书笔记**5

About

    - 这些拆解方式有一个共同点，需要是最高层次的分解，因此需要粗粒度的，足够简单，让人容易理解。
- relationship是成分间的关系，可以是层次结构，依赖关系
- principle指的是架构中使用的设计约束，如分层结构、EJB

领域模型和系统架构是什么关系呢？领域模型应该作为架构设计的重要输入，通常来说，领域模型的粒度较细，不足以作为软件架构的element或者component来解释，但是进一步抽象，领域模型的子领域划分是很好的模块划分方式，领域划分可以直接应用于架构的模块职责划分上。

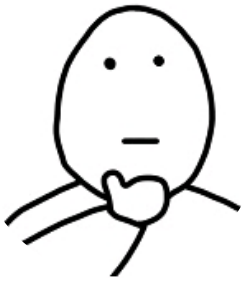
除了领域模型，系统架构还需要考虑其他几个重要因素：

- 前瞻性。前瞻性通常来源于业务发展规划、公司战略方向甚至大到社会发展趋势，领域模型来源于明确的产品设计方案，往往是得不到这些概念性输入的。系统架构设计的前瞻性直接影响到架构的稳定性，如果架构随着业务变化而震荡，那么对技术团队来说就是灾难。
- 性能和稳定性。非功能性需求往往不会在领域模型里体现，系统的实现中可能需要通过缓存，监控，一致性协议等满足非功能需求。
- 公用组件的抽取和沉淀。

7 一些感想

一切的方法论，都需要基于一些理想化的假设，但是现实世界的复杂注定了没有一套方法论是万能的、完整的。现实世界中，往往是这样的：

- 产品经理给出的需求描述（用例）永远都不完整，必须要靠持续的沟通和思考来挖掘真实的需求。
- 不同的人对于领域建模的方法和现实需求的理解各有不同，对未来的前瞻性也不同，这就导致当建模出现分歧时，难以有标准来判断谁是“正确”的。



Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

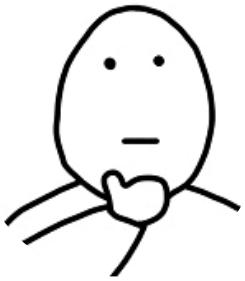
About

因此，领域建模的的确确十分依赖经验，并不同于纯粹的技术，更是个人综合能力的一种体现。

8 参考资料

[1] Larman C. Applying UML and patterns[J]. Pearson Schweiz Ag, 2004.
[2] Martin J, Odell J J. Object-Oriented Methods: A Foundation[J]. Object-oriented methods : a foundation, 1998, 222(2):3–11.
[3] Fowler M. Analysis Patterns: Reusable Object Models[M]. DBLP, 1997.
[4] Fowler M. Patterns of enterprise application architecture[M]. 中国电力出版社, 2004.
[5] Evans. Domain-Driven Design: Tacking Complexity In the Heart of Software[M]. 2004.
[6] Fairbanks G, Garlan D. Just Enough Software Architecture: A Risk-Driven Approach[J]. Crc Press, 2010.
[7] Booch G, Maksimchuk R, Engle M, et al. Object-oriented analysis and design with applications, third edition.[M]// Object-oriented analysis and design with applications /. Benjamin/Cummings Pub. Co. 1994:1275-1279.
[8] https://en.wikipedia.org/wiki/Object-oriented_analysis_and_design
[9] Moreno A M. Object-Oriented Analysis from Textual Specifications[C]// of 9 Th International Conference on Software Engineering and Knowledge Engineering. 1997:157.
[10] Coad P, Deluca J, Lefebvre E. Java Modeling in Color with UML[J]. 1999.
[11] 运用四色建模法进行领域分析
[12] Systems and software engineering — Architecture description
[13] 还有很多思想参考自阿里内网的一些文章，不便贴出：）

← prev • next →



Yilun Fan

站内搜索

- 个人随笔3
- 原创文章16
- 杂谈1
- 视频2
- 读书笔记5

About