尼恩面试宝典专题42：DDD面试题（史上最全、持续更新）

本文版本说明：V175

《尼恩面试宝典》升级规划为：

后续基本上，**每一个月，都会发布一次**，最新版本，可以联系构师尼恩获取， 发送“领取电子书”获取。

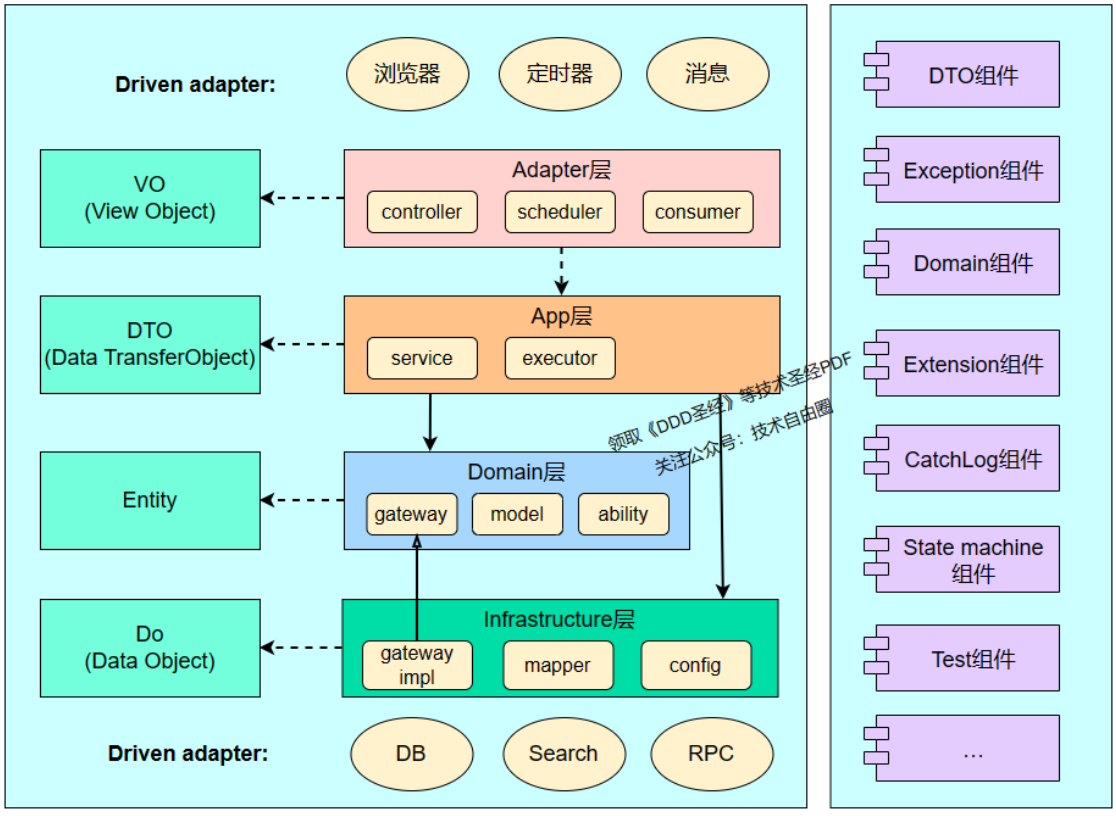
目录

真题1：DDD的外部接口调用，应该放在哪一层?

无论是RPC （如Feign），还是HTTPClient调用Rest API，具体的外部接口调用实现放在infrastructure基础设施层。

在领域驱动设计（DDD）中，基础设施层负责处理与外部资源的交互，包括数据库、文件系统、消息队列、外部服务等。

来看看 cola的 DDD 架构，具体如下图



COLA（Clean Object-oriented Architecture）框架是一种基于领域驱动设计（DDD）和面向对象设计原则的架构风格，旨在帮助开发人员构建清晰、可维护和可扩展的应用程序。

1）适配层（AdapterLayer）：负责对前端展示（web，wireless，wap）的路由和适配，对于传统B/S系统而言，adapter就相当于MVC中的controller；

2）应用层（ApplicationLayer）：主要负责获取输入，组装上下文，参数校验，调用领域层做业务处理，如果需要的话，发送消息通知等。层次是开放的，应用层也可以绕过领域层，直接访问基础实施层；

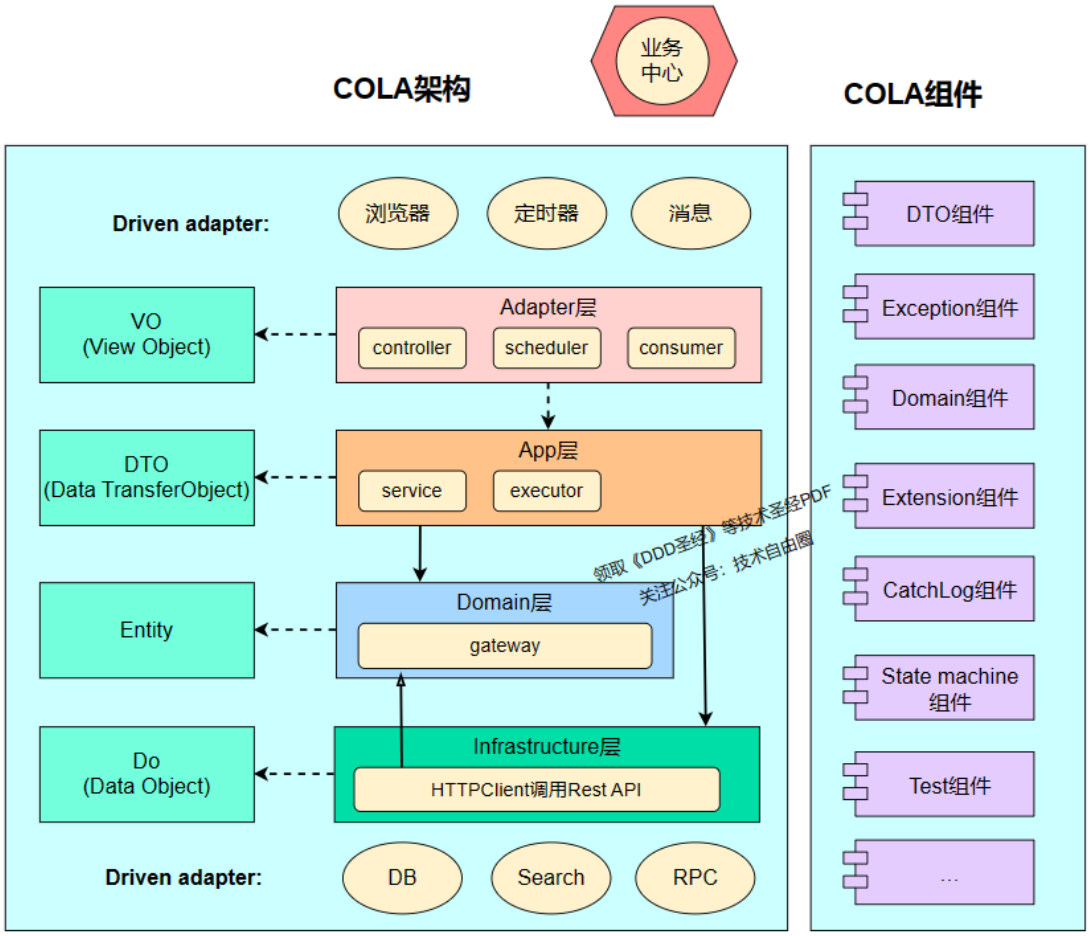
3）领域层（Domain Layer）：主要是封装了核心业务逻辑，并通过领域服务（Domain Service）和领域对象（DomainEntity）的方法对App层提供业务实体和业务逻辑计算。领域是应用的核心，不依赖任何其他层次；

4）基础实施层（InfrastructureLayer）：主要负责技术细节问题的处理，比如数据库的CRUD、搜索引擎、文件系统、分布式服务的RPC等。此外，领域防腐的重任也落在这里，外部依赖需要通过gateway的转义处理或者代理处理，才能被上面的App层和Domain层使用。

gateway 代理的架构，也非常类似与适配器的架构。 gateway 代理架构分为两个部分：

- gateway 接口处于 领域层，
- gateway 的实现处于 基础实施层（InfrastructureLayer）。

具体如下：



在COLA框架中，Gateway通常指的领域层的网关，是一种隔离在应用程序领域层内部和系统外部之间进行通信和交互的接口或者组件。COLA框架中的Gateway通常被用于以下几个方面：

1. **数据交互**：Gateway可以用于在领域层和基础设施层之间进行数据交互。例如，领域层的服务可以通过Gateway将数据传递给基础设施层进行持久化，或者基础设施层可以通过Gateway将数据加载到领域层中进行处理。
2. **外部接口调用**：Gateway也可以用于与外部系统进行通信。例如，Gateway可以封装与外部服务的交互逻辑，使得领域层不直接依赖于外部服务的细节。
3. **跨边界通信**：在复杂的应用程序中，可能存在多个边界或子系统。Gateway可以用于在这些边界之间进行通信，帮助将系统拆分为更小的模块并保持模块之间的松耦合性。

总之，COLA框架中的Gateway扮演着连接不同层和组件之间的桥梁角色，有助于实现应用程序的模块化、可维护和可扩展性。

问题2：什么是聚合?

“聚合”即“高内聚，低耦合”中的“内聚”之意；

聚合是业务和逻辑紧密关联的实体和值对象组合而成，聚合是数据修改和持久化的基本单元，一个聚合对应一个数据的持久化；

聚合在DDD分层架构中属于领域层，领域层包含了多个聚合，共同实现核心业务逻辑，聚合内的实体以充血模型实现个体业务能力，以及业务逻辑的高内聚； 跨多个实体的业务逻辑通过领域服务来实现，跨多个聚合的业务逻辑通过应用服务来实现；

首先我们来看下聚合模式的定义:

> *将实体和值对象划分为聚合并围绕着聚合定义边界。*
>
> *选择一个实体作为每个聚合的根，并仅允许外部对象持有对聚合根的引用。作为一个整体来定义聚合的属性和不变量，并把其执行责任赋予聚合根或指定的框架机制。*

一个聚合包含一个聚合根（aggregation root）和一些相关的其他领域对象。

问题3：什么是聚合根?

在《DDD学习圣经》中，我们讲到了“什么是聚合根”，这里再重复一下。

聚合根中的“聚合”即“高内聚，低耦合”中的“内聚”之意；

而“根”则是“根部”的意思，也即聚合根是一种统领式的存在。

事实上，并不存在一个教科书式的对聚合根的理论定义，你可以将聚合根理解为一个系统中最重要最显著的那些名词，这些名词是其所在的软件系统之所以存在的原因。

为了给你一个直观的理解，以下是几个聚合根的例子：

- 在一个电商系统中，一个订单（Order）对象表示一个聚合根
- 在一个CRM系统中，一个客户（Customer）对象表示一个聚合根
- 在一个银行系统中，一次交易（Transaction）对象表示一个聚合根

你可能会问，软件中的概念已经很多了，为什么还要搞出个聚合根的概念?

我们认为这里至少有2点原因：

1. 聚合根遵循了软件中“高内聚，低耦合”的基本原则
2. 聚合根体现了一种模块化的原则，模块化思想是被各个行业所证明的可以降低系统复杂度的一种思想。所谓的DDD是“软件核心复杂性应对之道”，也即这个意思，它将软件系统在人脑中所呈现地更加有序和简单，让人可以更好地理解和管控软件系统。

在实际项目中识别聚合根时，我们需要对业务有深入的了解，因为只有这样你才知道到底哪些业务逻辑是内聚在一起的。

这也是我们一直建议程序员和架构师们不要一味地埋头于技术而要多关注业务的原因。

事实上，如果让一个从来没有接触过DDD的人来建模，十有八九也能设计出上面的订单、客户和交易对象出来。

没错，DDD绝非什么颠覆式的发明，依然只是在前人基础上的一种进步而已，这种进步更多的体现在一些设计原则上，对此我们将在下文进行详细阐述。

问题4：聚合和MYSQL 表的对应关系是什么？

首先，简单说说聚合根

- 聚合根代表的是一个领域边界
- 聚合根的内容要保证数据一致性（这里的一致性指的不是数据持久化的事务一致性，而是业务数据的一致性，包含业务上的业务校验） 比如订单和订单详情，一个没有订单详情的订单是不完整的
- 聚合根里面有多少个实体，由领域建模决定
- 永远不要删除聚合根
- 聚合根之间有引用，如果删除了聚合根，会导致关联聚合的数据不一致
  这边很容易和实体的生命周期从属于聚合根搞混了。这边的依赖是关联依赖，实体依赖聚合根是has a
- 聚合根引用聚合根值id/或者id值对象

然后，简单说说实体

- 实体一般从属于某个聚合根，要不然就可以定义成聚合根了
- 实体有自己的生命周期，他的生命周期从属于聚合根。也就是聚合根没有，实体也就没了
  比如我可以对订单详情的数据进行编辑，删除。
- 聚合根与实体的关系通常是1:N
  因为如果是1:1,通常不需要定义实体了。直接放在聚合根里面，不需要唯一id了。
- 可以单独更新聚合根中实体数据

最后，说说Domain 聚合和MYSQL 数据表的对应关系是什么

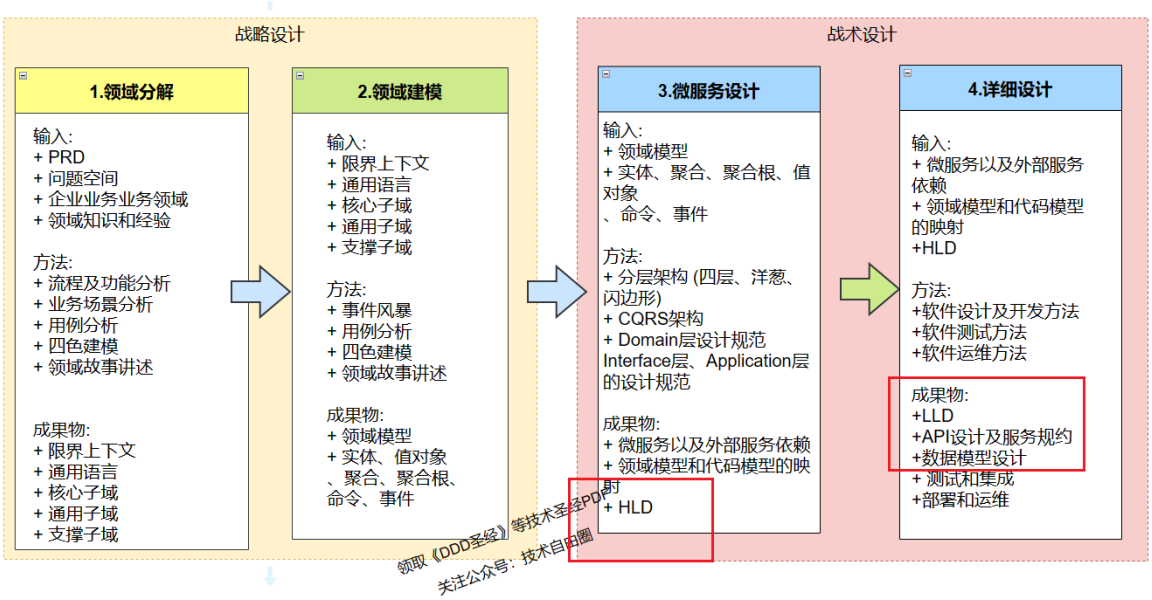
Domain 聚合属于领域建模

MYSQL 数据表设计属于 DB建模

注意，从理论上来说， DB建模(数据表建模)和领域建模没有关系，二者在处于不同的 设计阶段：

- 领域建模处于 概要设计阶段， 理清楚业务的内外边界，业务的内外流程，成果物为 HLD 概要设计文档。
- DB建模 处于 详细设计阶段， 理清楚业务的“关系数据模型”进而推导出“物理数据模型”，成果物为 LLD 详细设计文档。

具体如下图



> DDD建模的四个标准动作，具体请参见 尼恩《DDD学习圣经》以及配套视频。

数据建模的经典方法就是 E-R Model建模， 而当我们做MYSQL E-R建模的时候，我们需要考虑：

- 需要建什么主表 - 可以类比为对象 （当然也有些表只是为了关系映射）
- 表里面需要什么列 - 可以类比为这对象所需的属性
- 需要什么关联 - 表现数据对象与数据对象之间的联系

当然，E-R Model是数据模型的一种表现形式(数据建模不只是E-R Model一种表现形式)，E-R Model以数据为中心，关注的是对象的实体和关系，建模时并不考虑Entity的行为。

在E-R概念模型的基础上可以建立“关系数据模型”进而推导出“物理数据模型”，这是一条以E-R Model为起始的数据建模的路线

ER模型分为实体、属性、关系三个核心部分。实体是长方形体现，而属性则是椭圆形，关系为菱形。
ER模型的实体（entity）即数据模型中的数据对象，例如人、学生、音乐都可以作为一个数据对象，用长方体来表示，每个实体都有自己的实体成员（entity member）或者说实体对象（entity instance），例如学生实体里包括张三、李四等，实体成员（entity member）/实体实例（entity instance） 不需要出现在ER图中。

ER模型的属性（attribute）即数据对象所具有的属性，例如学生具有姓名、学号、年级等属性，用椭圆形表示，属性分为唯一属性（ unique attribute）和非唯一属性，唯一属性指的是唯一可用来标识该实体实例或者成员的属性，用下划线表示，一般来讲实体都至少有一个唯一属性。

ER模型的关系（relationship）用来表现数据对象与数据对象之间的联系，例如学生的实体和成绩表的实体之间有一定的联系，每个学生都有自己的成绩表，这就是一种关系，关系用菱形来表示。
ER模型中关联关系有三种：

1对1（1:1）：1对1关系是指对于实体集A与实体集B，A中的每一个实体至多与B中一个实体有关系；反之，在实体集B中的每个实体至多与实体集A中一个实体有关系。
1对多（1:N）：1对多关系是指实体集A与实体集B中至少有N(N>0)个实体有关系；并且实体集B中每一个实体至多与实体集A中一个实体有关系。
多对多（M:N）：多对多关系是指实体集A中的每一个实体与实体集B中至少有M(M>0)个实体有关系，并且实体集B中的每一个实体与实体集A中的至少N（N>0）个实体有关系。

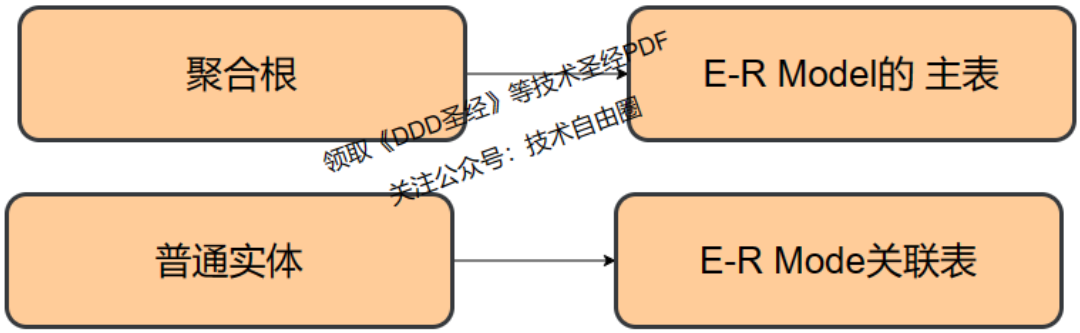
在数据建模中通常将E-R Model就称之为概念数据模型，接下来的层面是关系数据模型和物理数据模型；

而Domain 建模是属于对象建模、业务建模的范畴，Domain 聚合和MYSQL 数据表 二的区别是职责不同， ER建模关心数据如何存储，Domain 对象建模需要为对象建立职责（对象的行为）。

当然Domain 建模 和E-R Model的分析具有很大的相似性，

在一些极为简单的业务系统、业务场景中，二者可以做简单的映射：

- Domain 模型里边 聚合根 可以映射到 E-R Model的 主表
- Domain 模型里边 普通实体、值对象 可以映射到 E-R Model的 关联表



问题5：什么是领域驱动设计(DDD)?

什么是DDD

领域驱动设计最初由Eric Evans提出，

2004年著名建模专家eric evans（埃里克埃文斯）发表的他最具影响力的书籍:

《domain-driven design –tackling complexity in the heart of software》(中文译名：领域驱动设计—软件核心复杂性应对之道)一书。标志着 DDD 这种 设计和架构方法的诞生。

我们在日常开发中，经常针对一些功能点争论“这个功能不应该我改，应该是你那边改”，最终被妥协改了之后都改不明白为什么这个功能要在自己这边改。

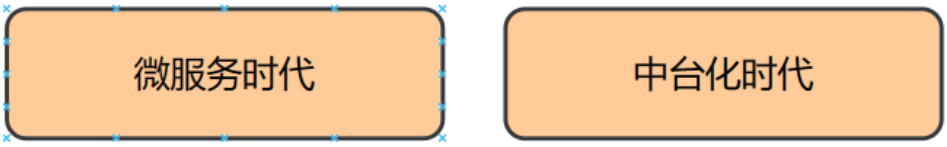
区别于传统的数据驱动架构（Data Driven Design）设计，领域驱动设计（DDD）也许在这个时候能帮助你做到清晰的划分。

但是多年以来一直停留在理念阶段，

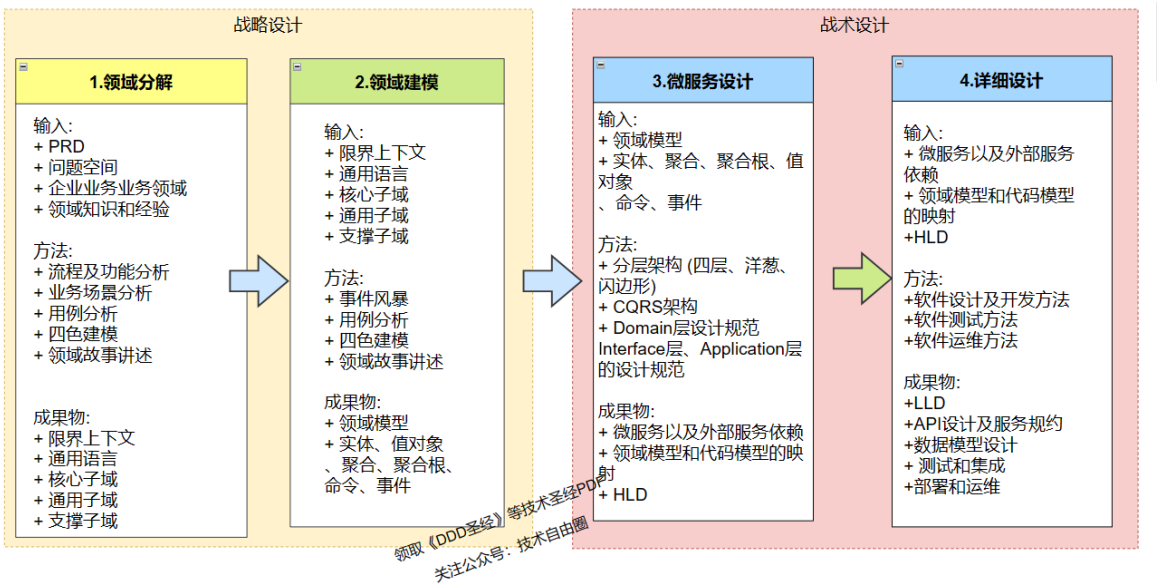
然后，真正能实现并且落地的项目和公司少之又少，

近来，包括阿里在内很多大厂，都在大力推行DDD的设计方法，

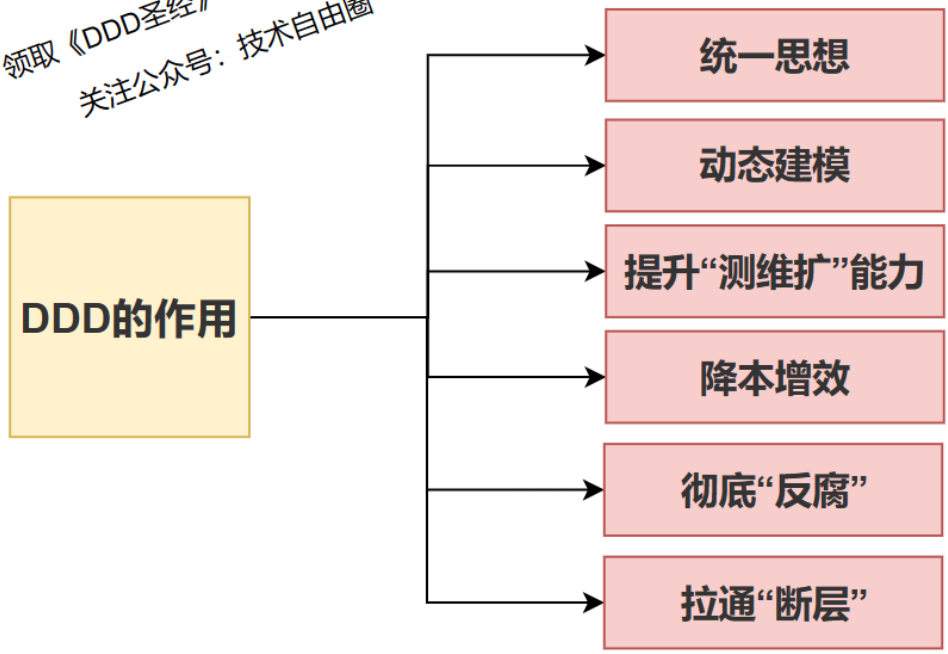
它主要可以帮助我们解决传统单体式、集中式、大泥球架构难以快速响应业务需求落地的问题，并且针对中台和微服务盛行的场景做出指导。



DDD为我们提供的是架构设计的方法论，既面向技术也面向业务，从业务的角度，自顶向下来把握设计方案。



DDD的巨大价值



统一思想：统一项目各方业务、产品、开发对问题的认知，明确组织当中产品、业务、架构、开发的角色和如何配合。通过统一的语言、明确的定义，统一方方面面的理解误差和理解分歧。

DDD工具链能帮助进一步**强化**能力逻辑的**表达**，借助DDD可视化流程构建专业**知识库**，能快速提升产品和技术、技术和技术之间的**沟通效率**，帮助开发同学快速、直观的了解业务，进而提高技术同学对业务有快速/全局了解，快速+全局的业务了解，反向帮助开发通讯能够方便把握细节，降低反复、返工、甚至推导重来的概率

动态建模：需求是不断变化的，传统HLD、LLD建模都是核心思想是静态建模，缺乏有效的动态建模方法论和工具。DDD通过从领域事件、领域命令出发对领域对象进行建模，可以真实的反映这些变化。更好的通过边界划分将复杂业务领域简单化，将隐藏的业务显性化， 隐式流程显性化， 隐式字段显性化，帮助我们设计出清晰的领域边界，准确的业务流程，可以很容易地实现业务和技术统一的架构演进；

拉通“断层”：传统的需求常常是一句话需求，模型设计常常是工程师负责， 工程师的设计路径是 库表驱动+界面驱动，结合常规MVC三层架构进行自底向上的设计，完美的实现了业务人员和编码人员断层，需求和开发隔离的目标。DDD拉通了业务和编码，对常规MVC开发模式做一个反转，以业务为主导，自顶向下的进行领域模型设计，拉通业务和编码之间的巨大“断层”，使得代码更能反馈业务，反哺业务，提升代码逻辑的准确度和生命值。

业务人员和编码人员隔离的原因：轻设计、重编码。 一句话需求，敏捷迭代、极速迭代、轻量级的流程。很容易把代码弄得杂乱无章，混乱不堪。

彻底“反腐”：首先是领域模型与数据模型分离，用领域模型来界定哪些需求在什么地方实现，保持结构清晰，隔离数据模型、存储模型的变化和腐败。除了数据模型，咱们应用 中有许多容易"腐败"的操作，比如直接的外部依赖，例如 MyBatis 的 Mapper 类、HttpClient 注入、RocketMQ 的监听、缓存的直接操作等，DDD通过防腐层的架构设计，实现 业务代码的彻底反腐败。

简单来说：

- “反腐败”设计，从一半业务+一半技术 ， 提升到 业务代码和 基础设施解耦。
- “反腐败”设计，使得领域代码更有业务纯度。

同时，DDD帮助**沉淀**各领域的业务，**标准化**流程链路，各领域间**无耦合**，沉淀的领域能力能够很好的复用，粗粒度的应用能力能基于细粒度领域能力去构建，构建好的能力可以在其他场景直接**复用**，提高开发**效率**，最终提升 领域代码的生命力、复用力

提升"测维扩"能力:

`可维护性 = 当依赖变化时，有多少代码需要随之改变`。传统MVC三层架构，面临各种库升级、依赖服务升级、中间件升级、jar包冲突、微服务框架、存储扩容升级等依赖变化工作，需要从上到下，每一层的代码都要动，可维护性差。

`可扩展性 = 做新需求或改逻辑时，需要新增/修改多少代码`。在库表驱动开发、库表驱动架构的开发流程中，一般做第一个需求都非常的快，但是由于代码复用性低，越做到后面，做第N个需求时需要的时间很有可能是呈指数级上升的，绝大部分时间花费在老功能的重构和兼容上，最终你的创新速度会跌为0，促使老应用被推翻重构。

`可测试性 = 运行每个测试用例所花费的时间 * 每个需求所需要增加的测试用例数量`。在库表驱动开发、库表驱动架构的开发流程中，由于设施搭建困难、用例笨重运行耗时长、业务耦合度高用例爆炸等原因，业务代码很难有比较好的测试覆盖，而绝大部分的业务代码上线前的测试属于人肉的"集成测试"。低测试率导致我们对代码质量很难有把控，容易错过边界条件，异常case只有线上爆发了才被动发现。

最终这个应用变成了一个不敢升级、不敢部署、不敢写新功能、并且随时会爆发的炸弹，终有一天会给你带来惊喜。

DDD根本上解决上面的问题，提升"测维扩"能力。

降本增效： 以爱奇艺DDD落地案例为例，其会员业务部门在打赏业务中实践 DDD 后，取得了以下显著成果：新需求接入开发成本节约**20%**；更换底层中间件开发成本节约**20%**；项目熟悉成本节约**30%**(对DDD有基本了解为前提)；单测开发成本指数级**降低**；上线风险、成本**降低**。

> 以上内容，具体请参见 尼恩《DDD学习圣经》以及配套视频。

DDD宏观概念

1. 领域、子域

领域可以理解为所要面和的客户要开发的软件的业务范围和业务逻辑。

由于领域很大，所以只能关注领域的的某个方面，并进行相关的软件建模，这个软件模型又称之为"领域模型"。

领域又可按照业务归类逻辑拆分为`相互分离`的子领域。

为什么要相互分离？理解一下面向接口的编程就可以了，子域与子域间通过接口关联通信，隐藏子域内部细节。

2. 限界上下文

限界上下文可以理解为：

1. 一般情况，是 一个子域对应一个限界上下文，也就是说，我们拆分领域为分离的子域，也就是在拆限界上下文

2. 在限界上下文内，各种术语、流程、代号有固定的意思。而这些术语、流程、代号在别的限界上下文就可能不是一样的意思。所以才会使用限界上下文。

   举个例子：在软件开发范畴内（限界上下文），质量是代码经过单元测试和集成测试、自动化测试、安全测试等一系统测试的软件，如果在建筑工程范畴，这个质量就是另一种意思了。

   一个很简单的例子，“顾客”在订单子域是下单付费的登录用户，

   而在产品目录子域，“顾客”则是所有浏览商品的用户，包括匿名未登录的和已登录的用户，同一个术语在不同的子域里有不同的意思，所以这两个子域不应规划到一个限界上下文里。
3. 甚至一个子域可能包括多个限界上下文，理论上讲应该将这个子域拆分为更细的子域，以达到一个子域对应一个限界上下文的标准。
4. 可以粗粗地认为限界上下文就是子域关联的一系列的术语、流程和代号。确定统一语言的上下文
5. 当关注到一个子域时，该子域即是核心域，而与其集成的支撑域和通用子域，为了支持该核心域业务方面，该核心域的限界上下文会包含支撑域、通用子域的一部分或全部。

限界上下文包含哪些领域模型呢?

1. 实体
2. 值对象
3. 聚合
4. 领域事件
5. 领域服务
6. ....

限界上下文是一个显式的边界，领域模型便存在于这个边界之内。

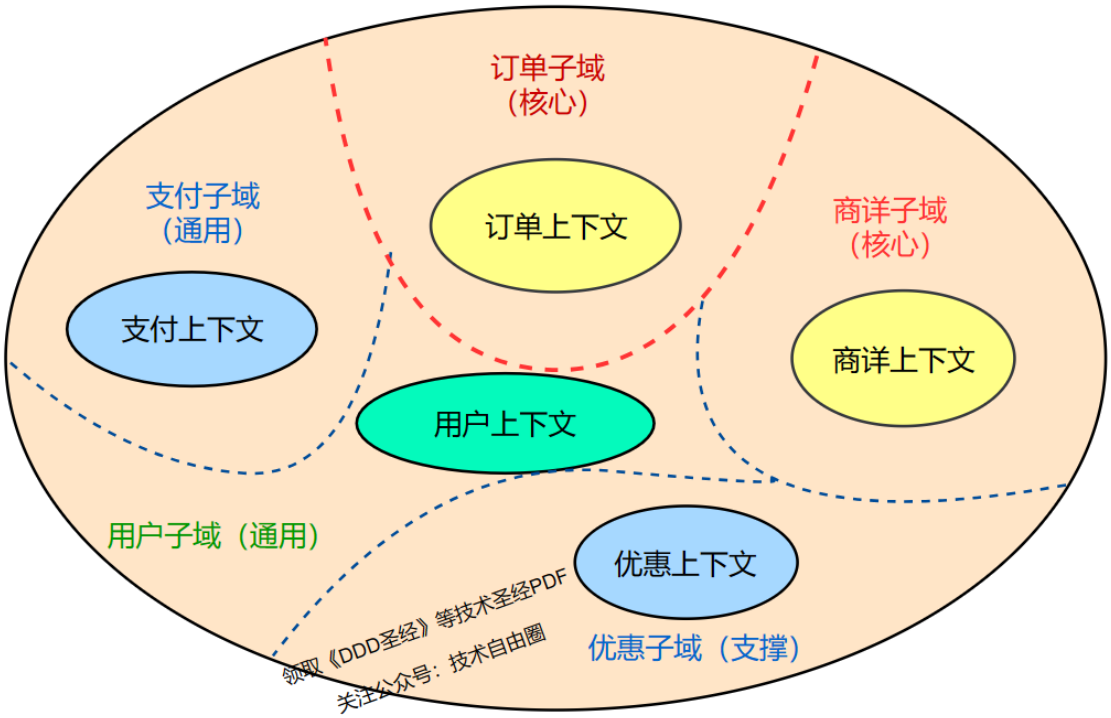
你可以认为他是一个namespace。

限界上下文并不只是局限于容纳模型，它通常标定了一个系统、一个应用程序或者一种业务服务。

3. 核心域、支撑域、通用子域

1. 核心域，可以理解为领域中你关注的问题空间的业务子域。比如说，电商系统领域，订单子域是关于订单业务的，所以是核心域。
2. 支撑域，可以理解为领域中涉及业务但是不是核心业务的子域。比如说，商品品类子域，它是关于业务的，但是相对于核心域来说，它不是核心业务，但是它给核心域提供了商品查询支持，所以它是支撑域。
3. 通用子域，被所有子域需要的子域，一般只提供数据、接口，但不是涉及业务相关。比如说帐号子域，它并不涉及电商的业务，但是却给核心子域提供了帐号相关的数据资料。它属于工具类型，可以理解为对核心域提供接口支持的都可以称之为通用子域。

> 核心域是相对的。
> 当你要解决什么问题，那么该问题对应的子域就是核心域，所以对该子域提供支持的都是支撑域和通用子域。



DDD微观概念

领域驱动设计围绕着**领域模型**进行设计，通过**分层架构**将领域独立出来。

这里有两个关键词：

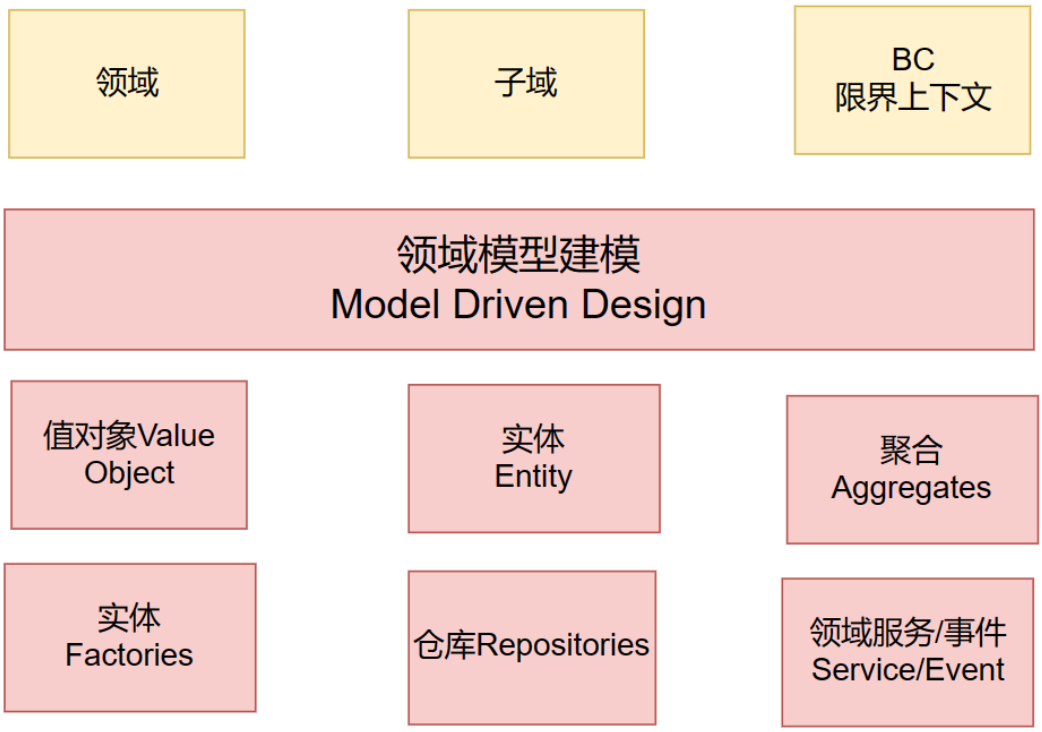
- 领域模型
- 分层架构

领域模型的对象包括：实体、值对象和领域服务，领域逻辑都应该封装在这些对象中。

领域模型 Domain Model

领域反映到代码里就是模型，模型是对领域某个方面的抽象，并且可以用来解决相关域的问题，

Domain Model 的基础单元，分为实体和值对象两种。实体和值对象，二者是领域模型中非常重要的基础领域对象(Domain Object，DO)。



实体对象 Entities

有唯一标志的核心领域对象，且这个标志在整个软件生命周期中都不会发生变化。

这个概念和我们平时软件模型中和数据库打交道的Entity实体比较接近，

不同的是DDD中这些实体会包含与该实体相关的业务逻辑，它是操作行为的载体。

白话解释：实体就是对象的方法和属性实现业务逻辑的类，一般由唯一标识id和值对象组成，属性发生改变，可以影响类的状态和逻辑。

实体 = 唯一身份标识 + 可变性【状态 + 行为】

DDD 中要求实体是唯一的且可持续变化的。

意思是说在实体的生命周期内，无论其如何变化，其仍旧是同一个实体。

唯一性由唯一的身份标识来决定的。

可变性也正反映了实体本身的状态和行为。

实体以 DO（领域对象）的形式存在，每个实体对象都有唯一的 ID。

我们可以对一个实体对象进行多次修改，修改后的数据和原来的数据可能会大不相同。

但是，由于它们拥有相同的 ID，它们依然是同一个实体。

比如商品是商品上下文的一个实体，通过唯一的商品 ID 来标识，不管这个商品的数据如何变化，商品的 ID 一直保持不变，它始终是同一个商品。

值对象 Value Object

依附于实体存在，通过对象属性来识别的对象，它将一些相关的实体属性打包在一起处理，形成一个新的对象。

这些对象是用来表示临时的事物，或者可以认为值对象是实体的属性，这些属性没有特性标识但同时表达了领域中某类含义的概念。

通常值对象不具有唯一id，由对象的属性描述，可以用来传递参数或对实体进行补充描述。

白话解释：不关心唯一性，具有校验逻辑、等值判断逻辑，只关心值的类。

举个栗子：

比如用户实体，包含用户名、密码、年龄、地址，地址又包含省市区等属性，而将省市区这些属性打包成一个属性集合就是值对象。

值对象与实体的区别是什么？

- **值对象没有唯一标识和连续性**，任何属性发生变化, 都可以认为是新的值对象。判断对象是否相同：值对象需要判断所有属性是否相同，而实体只需要判断唯一标识是否相同。
- **值对象一般依附于实体而存在**，是实体属性的一部分，而非独立存在。值对象属性是只读的，可以被安全的共享.

值对象 = 将一个值用对象的方式进行表述，来表达一个具体的固定不变的概念。

还是举个订单的例子，订单是一个实体，里面包含地址，这个地址可以只通过属性嵌入的方式形成的订单实体对象，也可以将地址通过 json 序列化一个 string 类型的数据，存到 DB 的一个字段中，那么这个 Json 串就是一个值对象，是不是很好理解？

当你只关心某个对象的属性时，该对象便可作为一个值对象。

我们需要将值对象看成不变对象，**不要给它任何身份标识**，注意，**不要给它任何身份标识，该对象便可作为一个值对象。**

注意：应该尽量避免像实体对象一样的复杂性。

聚合(aggregate)

实体和值对象表现的是个体的能力，而我们的业务逻辑往往很复杂，依赖个体是无法完成的，这时候就需要多个实体和值对象一起协同工作，而这个协同的组织就是聚合。

聚合是数据修改和持久化的基本单元，同一个聚合内要保证事务的一致性，所以在设计的时候要保证聚合的设计拆分到最小化以保证效率和性能。

白话解释：就是对象之间的关联，只是规定了关联对象规则，操作聚合时，类似于操作Hibernate中的One-Many对象的概念。

在DDD中，实体和值对象是很基础的领域对象。实体一般对应业务对象，它具有相对丰富的业务属性和业务行为。而值对象主要是属性集合，主要完成对实体的状态和特征描述。

聚合在 DDD分层架构里属于领域层，同一个微服务的领域层可以有多个聚合，**每个聚合内有一个聚合根，多个实体、值对象和领域服务等领域对象。**同一个限界上下文内的多个聚合，通过应用层组合在一起共同实现了领域模型的核心领域逻辑。

我们为每一个聚合设计一个仓储完成聚合数据的持久化操作。为了避免聚合数据频繁地提交，建议你尽可能将聚合内变更的数据，封装在一次交易中提交仓储完成持久化。

聚合在领域模型里是一个逻辑边界，它本身没有业务逻辑实现相关的代码。聚合的业务逻辑是由聚合内的聚合根、实体、值对象和领域服务等来实现的。聚合内的实体以充血模型实现自身的业务逻辑。**跨多个实体的领域逻辑通过领域服务来实现。**比如，有的业务场景需要同-个聚合的A和B两个实体来共同完成，我们就可以将这段业务逻辑用领域服务组合A和B两个实体来完成。

跨多个聚合的业务逻辑的组合和编排，是通过应用服务来实现的。比如，有的业务逻辑需要聚合C和聚合D中的两个领域服务来共同完成，为了避免聚合之间的领域服务直接调用，实现微服务内聚合解耦，**此时你可以将这段业务逻辑上升到应用层，**通过应用服务组合两个聚合的领域服务来实现。

聚合是领域对象的显式分组，我们把一些关联性极强、生命周期一致的实体、值对象放到一个聚合里。

聚合定义了一组具有内聚关系的相关对象的集合，每个聚合都有一个根对象（聚合根实体）。

我们把聚合看作是一个修改数据的单元。 一个聚合是一组相关的被视为整体的对象。每个聚合都有一个根对象（聚合根实体），从外部访问只能通过这个对象。

根实体对象有组成聚合所有对象的引用，但是外部对象只能引用根对象实体。只有聚合根才能使用仓储库直接查询，其它的只能通过相关的聚合访问。如果根实体被删除，聚合内部的其它对象也将被删除。

为啥需要进行聚合？旨在支持领域模型的行为和不变性，同时充当一致性和事务性边界。

聚合有两个核心要素：

- 一个聚合根
- 一个上下文边界

这个边界 根据业务单一职责和高内聚原则，定义了聚合内部应该包含哪些实体和值对象，而聚合之间的边界是松耦合的。

按照这种方式设计出来的服务很自然就是“高内聚、低耦合”的。

聚合在 DDD 分层架构里属于领域层，领域层包含了多个聚合，共同实现核心业务逻辑。

聚合根(aggregate root)

如果把聚合比作组织，那聚合根就是这个组织的负责人。

也叫做根实体，一个特殊的实体，它是聚合的管理者，代表聚合的入口，抓住聚合根可以抓住整个聚合。

聚合根也称为根实体，它不仅是实体，还是聚合的管理者。

- 首先它作为实体本身，拥有实体的属性和业务行为，实现自身的业务逻辑。
- 其次它作为聚合的管理者，在聚合内部负责协调实体和值对象按照固定的业务规则协同完成共同的业务逻辑。
- 最后在聚合之间，它还是聚合对外的接口人，以聚合根 ID 关联的方式接受外部任务和请求，在上下文内实现聚合之间的业务协同。也就是说，聚合之间通过聚合根 ID 关联引用，如果需要访问其它聚合的实体，就要先访问聚合根，再导航到聚合内部实体，外部对象不能直接访问聚合内实体。

聚合是一种边界，它可以封装一到多个实体与值对象，并维持该边界范围之内的业务完整性。

在聚合中，至少包含一个实体，且只有实体才能作为聚合根。

简单概括一下：

- 通过事件风暴（我理解就是头脑风暴，不过我们一般都是先通过个人理解，然后再和相关核心同学进行沟通），得到实体和值对象；
- 将这些实体和值对象聚合为“投保聚合”和“客户聚合”，其中“投保单”和“客户”是两者的聚合根；
- 找出与聚合根“投保单”和“客户”关联的所有紧密依赖的实体和值对象；
- 在聚合内根据聚合根、实体和值对象的依赖关系，画出对象的引用和依赖模型。

服务（services）

服务提供的操作是它提供给使用它的客户端，并突出领域对象的关系。

所有的service只负责协调并委派业务逻辑给领域对象进行处理，其本身并未真正实现业务逻辑，绝大部分的业务逻辑都由领域对象承载和实现了。

service可与多种组件进行交互，这些组件包括：其他的service、领域对象和repository 或 dao。

服务又细分为领域服务和应用服务。

领域服务（Domain Service）

接下来，看看领域服务和应用服务两个核心概念。

领域中的一些概念，如果是名词，适合建模为对象的一般归类到实体对象或值对象。

如果是动词，比如一些操作、一些动作，代表的是一种行为，如果是和实体或值对象密切相关的，也可以合并到某个实体或者值对象中。

但是，有些操作**不属于实体或者值对象本身**，或会**涉及到多个领域对象**，并且需要协调这些领域对象共同完成这个操作或动作，这时就需要创建领域服务来提供这些操作。

有些领域的操作是一些动词，并不能简单的把他们归类到某个实体或者值对象中。

领域的动作，从领域中识别出来之后，应该将它声明成一个服务，它的作用仅仅是为领域提供相应的功能。

简单理解： 就是跨多个领域对象的业务方法

当一些逻辑不属于某个实体时，可以把这些逻辑单独拿出来放到领域服务中。可以使用领域服务的情况：

- 执行一个显著的业务操作
- 对领域对象进行转换
- 以多个领域对象作为输入参数进行计算，结果产生一个值对象

领域服务有两个特征：

- 1) 操作代表了一个领域概念，且不是实体或者值对象的一个自然的部分；
- 2）被执行的操作涉及领域中的其他对象；操作是无状态的。领域服务还有一个好处可以避免领域逻辑泄露到t应用层。

因为如果没有领域服务，那么应用层会直接调用领域对象完成本该是属于领域服务该做的操作。

此外，如果实体操作过多或者过大，为了避免臃肿，也可以使用领域服务来解决。

但是，不能把所有的东西都搬到领域服务里，过度使用可能会导致产生的太多的贫血对象。

理想的情况是没有领域服务，如果领域服务使用不恰当，慢慢又演化回了以前逻辑都在 service 层的局面

应用服务（Application Service）

应用层作为展现层与领域层的桥梁，是用来表达用例和用户故事的主要手段。

应用层通过应用服务接口来暴露系统的全部功能。

在应用服务的实现中，它负责编排和转发，它将要实现的功能委托给一个或多个领域对象来实现，它本身只负责处理业务用例的执行顺序以及结果的拼装。

通过这样一种方式，它隐藏了领域层的复杂性及其内部实现机制。

应用层相对来说是较"薄"的一层，除了定义应用服务之外，在该层我们可以进行安全认证，权限校验，持久化事务控制，或者向其他系统发生基于事件的消息通知，另外还可以用于创建邮件以发送给客户等。

领域服务和应用服务的不同：

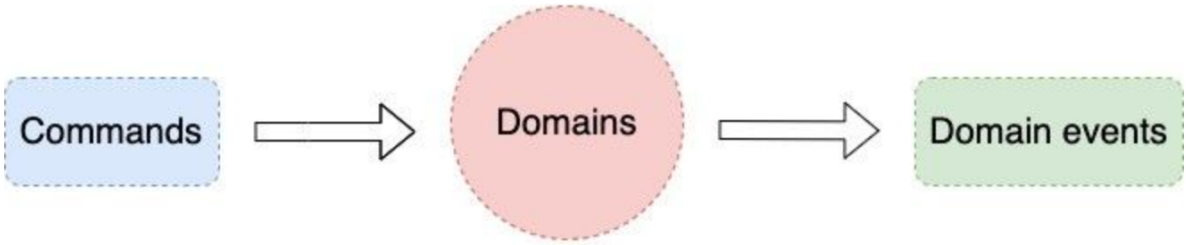
- 领域服务和应用服务是不同的，领域服务是领域模型的一部分，用来处理业务逻辑，而应用服务不是。
- 应用服务是领域服务的直接客户，负责处理**事务、安全**等操作，它将领域模型变成对外界可用的软件系统。
- 跨多个实体的业务逻辑通过领域服务来实现，跨多个聚合的业务逻辑通过应用服务来实现。

跨多个实体的业务逻辑通过领域服务来实现，跨多个聚合的业务逻辑通过应用服务来实现。

- 比如有的业务场景需要同一个聚合的 A 和 B 两个实体来共同完成，我们就可以将这段业务逻辑用领域服务来实现；
- 而有的业务逻辑需要聚合 C 和聚合 D 中的两个服务共同完成，这时你就可以用应用服务来组合这两个服务。

领域事件/领域命令

领域事件是一个领域模型中极其重要的部分，用来表示领域中发生的事件。



领域事件 = 事件发布 + 事件存储 + 事件分发 + 事件处理。

程序事件通常分为：系统事件、应用事件和领域事件。领域事件的触发点在领域模型中。

它的作用是将领域对象从对repository或service的依赖中解脱出来，避免让领域对象对这些设施产生直接依赖。

它的做法就是当领域对象的业务方法需要依赖到这些对象时,就发出一个事件，这个事件会被相应的对象监听到并做出处理。譬如跨限界上下文时，使用关键应用事件触发事件传递。

从尼恩的视角简单来说：领域事件是对 repository或service 的异步解耦。

在尼恩写的深度文章 京东一面：20种异步，你知道几种？ 含协程 中，就有EventBus 这样的事件总线，完成模块之间的异步解耦，也有 RocketMQ这样的分布式消息组件，完成进程级别的异步解耦。

而领域事件 是设计维度的解耦。EventBus 、RocketMQ是实现层面的异步解耦，当然是先有设计，后有实现。

在DDD中，通过领域事件，一个领域模型可以忽略不相关的领域活动，同时明确领域专家要跟踪或希望被通知的事情，或与其他模型对象中的状态更改相关联。

下面简单说明领域事件：

- **事件发布**：构建一个事件，需要唯一标识，然后发布；
- **事件存储**：发布事件前需要存储，因为接收后的事建也会存储，可用于重试或对账等；
- **事件分发**：服务内直接发布给订阅者，服务外需要借助消息中间件，比如Kafka，RabbitMQ等；
- **事件处理**：先将事件存储，然后再处理。

比如下订单后，给用户增长积分与赠送优惠券的需求。如果使用瀑布流的方式写代码。一个个逻辑调用，那么不同用户，赠送的东西不同，逻辑就会变得又臭又长。

这里的比较好的方式是，用户下订单成功后，发布领域事件，积分聚合与优惠券聚合监听订单发布的领域事件进行处理。

在特定的领域由用户动作触发，表示发生在过去的事件，或者领域状态的变化。

比如:

- 充值成功
- 充值失败的事件。

资源仓储/资源库（Repository）

仓储（资源库）是用来管理实体的集合。

仓储介于领域模型和数据模型之间，主要用于聚合的持久化和检索。

仓储里面存放的对象一定是聚合，原因是domain是以聚合的概念来划分边界的；聚合作为一个整体概念，要么一起被取出来，要么一起被删除。

它隔离了领域模型和数据模型，以便我们关注于领域模型而不需要考虑如何进行持久化。

我们将暂时不使用的领域对象从内存中持久化存储到磁盘中。

当日后需要再次使用这个领域对象时，根据 key 值到数据库查找到这条记录，然后将其恢复成领域对象，应用程序就可以继续使用它了，这就是领域对象持久化存储的设计思想。

工厂（Factory）

工厂用来封装创建一个复杂对象尤其是聚合时所需的知识，作用是将创建对象的细节隐藏起来。

客户传递给工厂一些简单的参数，然后工厂可以在内部创建出一个复杂的领域对象然后返回给客户。

工厂（Factory）不是必须的，只有当创建实体和值对象复杂时，建议使用工厂模式。

工厂和资源库都是对领域对象生命周期的管理。

工厂负责领域对象的创建，用于封装复杂或者可能变化的创建逻辑。

资源库负责从存放资源的持久层获取、添加、删除或者修改领域对象。

> 以上内容，具体请参见 尼恩《DDD学习圣经》以及配套视频。

问题6：什么是贫血领域模型?什么是充血模型?它们的优缺点是什么?

DDD四种模式

接下来，看看DDD四种模式

失血模型

领域模型中只有简单的get set方法，是对一个实体最简单的封装，其他所有的业务行为由服务类来完成。

pojo里边光秃秃的，get和set方法都没有

```
@Data
@ToString
public class User {
    private Long id;
    private String username;
    private String password;
    private Integer status;
    private Date createdAt;
    private Date updatedAt;
    private Integer isDeleted;
}
```

而且，其他所有的业务行为由服务类来完成，

```
public class UserService{
    public boolean isActive(User user){
        return user.getStatus().equals(StatusEnum.ACTIVE.getCode());
    }
}
```

贫血模型

贫血模型是指领域对象里只有get和set方法（POJO），所有的业务逻辑（不包含对象的状态变化在内），放在Business Logic层。

Domain Object（领域对象）模型包含对象属性的定义和操作对象属性的getter/setter方法并包含了对象的行为（例如：就像一个完整的人，具有一些属性如姓名、性别、年龄等，还具有一些能力，如走路、吃饭、恋爱等，这样才是一个完整的对象），但不包含依赖Dao层(持久层)的业务逻辑。

这部分依赖于Dao层的业务逻辑将会放到Business Logic层（业务逻辑层）中的服务类来实现，组合逻辑也由服务类负责。

可以看出，贫血模型中的领域对象是不依赖于持久层的。

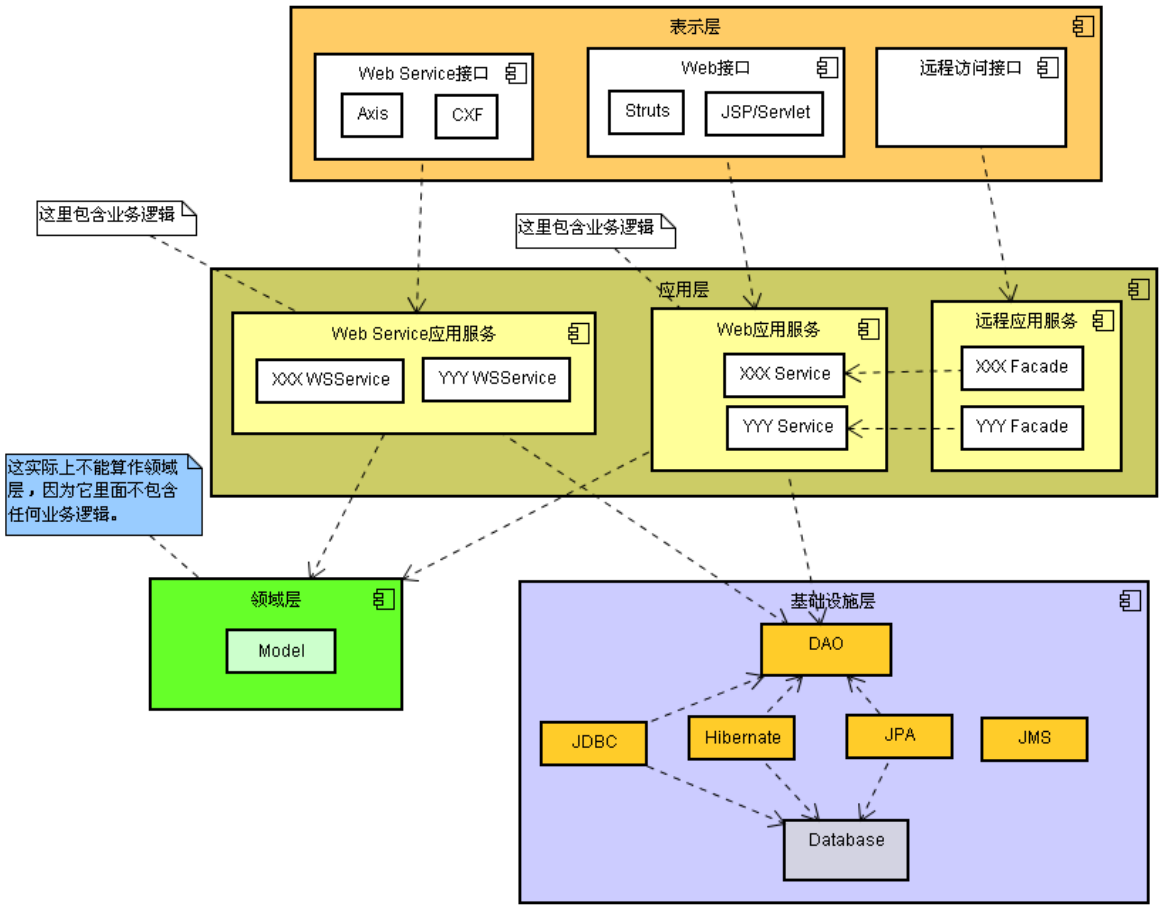
代码架构层次结构是： Client-> Business Facade Service -> Business Logic Service(Business Logic Service是依赖Domain Object的行为) -> Data Access Service

```
@Data
@ToString
public class User {
    private Long id;
    private String username;
    private String password;
    private Integer status;
    private Date createdAt;
    private Date updatedAt;
    private Integer isDeleted;
```

```
    public boolean isActive(User user){
        return user.getStatus().equals(StatusEnum.ACTIVE.getCode());
    }

    public void setUsername(String username){
        return username.trim();
    }
}
```

贫血模型在失血模型基础之上，领域对象的包含一些状态变化，但是停留在内存层面，不关心数据持久化。

贫血模型所有的业务逻辑都不包含在内而是放在Business Logic层。



贫血模型优点是系统的层次结构清楚，各层之间单向依赖，Client->（Business Facade）->Business Logic->Data Access Object。

可见，领域对象几乎只作传输介质之用，不会影响到层次的划分。

这就是 传统的 数据驱动的开发。

在使用Spring的时候，通常暗示着你使用了贫血模型，我们把Domain类用来单纯地存储数据，Spring管不着这些类的注入和管理，Spring关心的逻辑层（比如单例的被池化了的Business Logic层）可以被设计成singleton的bean。

假使我们这里逆天而行，硬要在Domain类中提供业务逻辑方法，那么我们在使用Spring构造这样的数据bean的时候就遇到许多麻烦，比如：bean之间的引用，可能引起大范围的bean之间的嵌套构造器的调用。

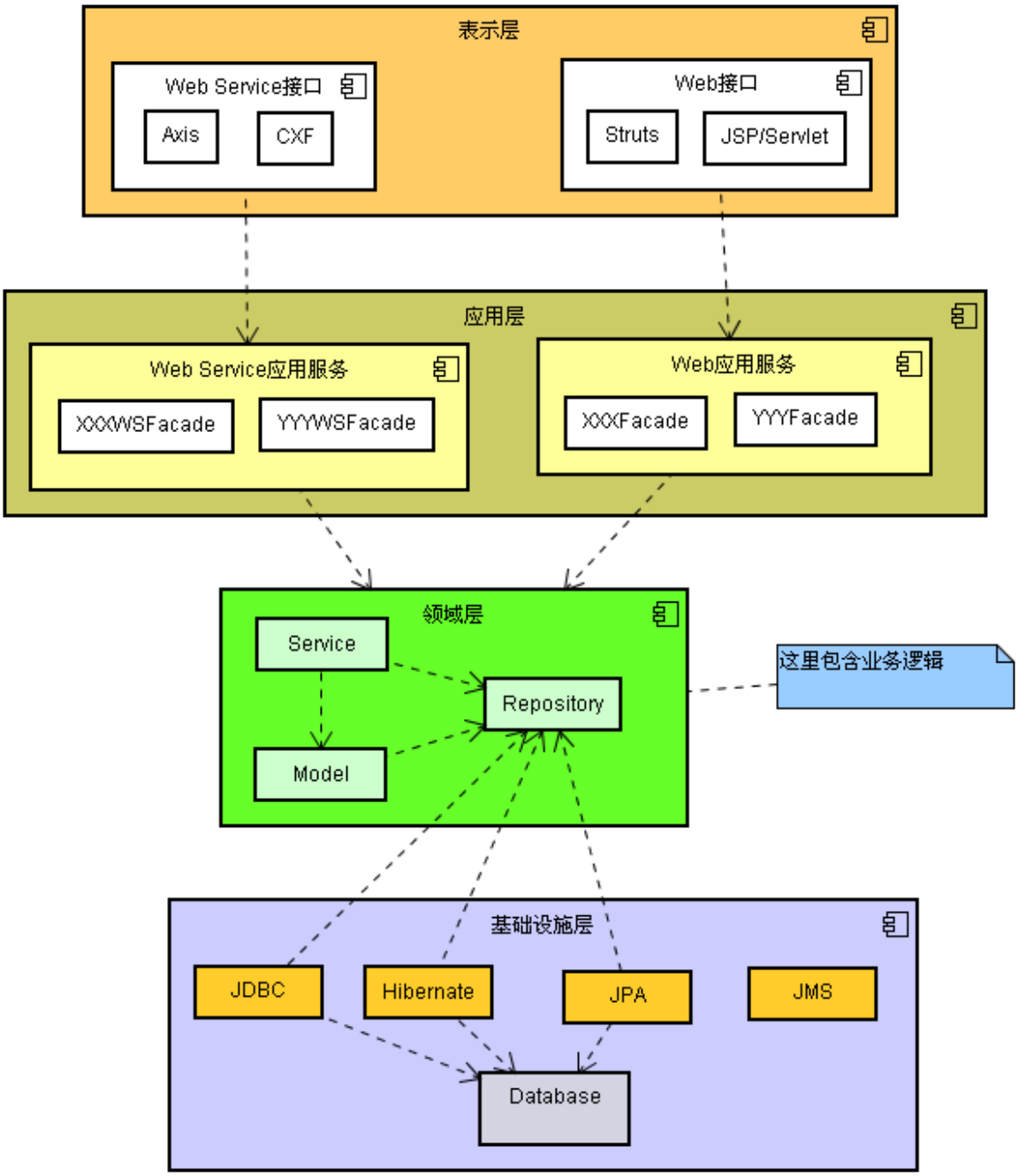
充血模型

在贫血模型基础上，负责数据的持久化。

```
@Data
@ToString
public class User {
    private Long id;
    private String username;
    private String password;
    private Integer status;
    private Date createdAt;
    private Date updatedAt;
    private Integer isDeleted;

    private UserRepository userRepository;

    public boolean isActive(User user){
        return user.getStatus().equals(StatusEnum.ACTIVE.getCode());
    }

    public void setUsername(String username){
        this.username = username.trim();
        userRepository.update(user);
    }
}
```

充血模型层次结构和上面的差不多，不过大多业务逻辑放在Domain Object里面，应用服务层Business Logic只是简单封装部分业务逻辑以及控制事务、权限等，

这样层次结构就变成Client->（Business Facade）->Business Logic->Domain Object->Data Access Object。



它的优点是面向对象，Business Logic符合单一职责，不像在贫血模型里面那样包含所有的业务逻辑太过沉重。

胀血模型

service都不需要，所有的业务逻辑、数据存储都放到一个类中。

对于DDD来说，失血和胀血都是不合适的，

失血太轻量没有聚合，胀血那是初学者才这样写代码。

那么充血模型和贫血模型该怎么选择?

充血模型依赖repository接口，与数据存储紧密相关，有破坏程序稳定性的风险。

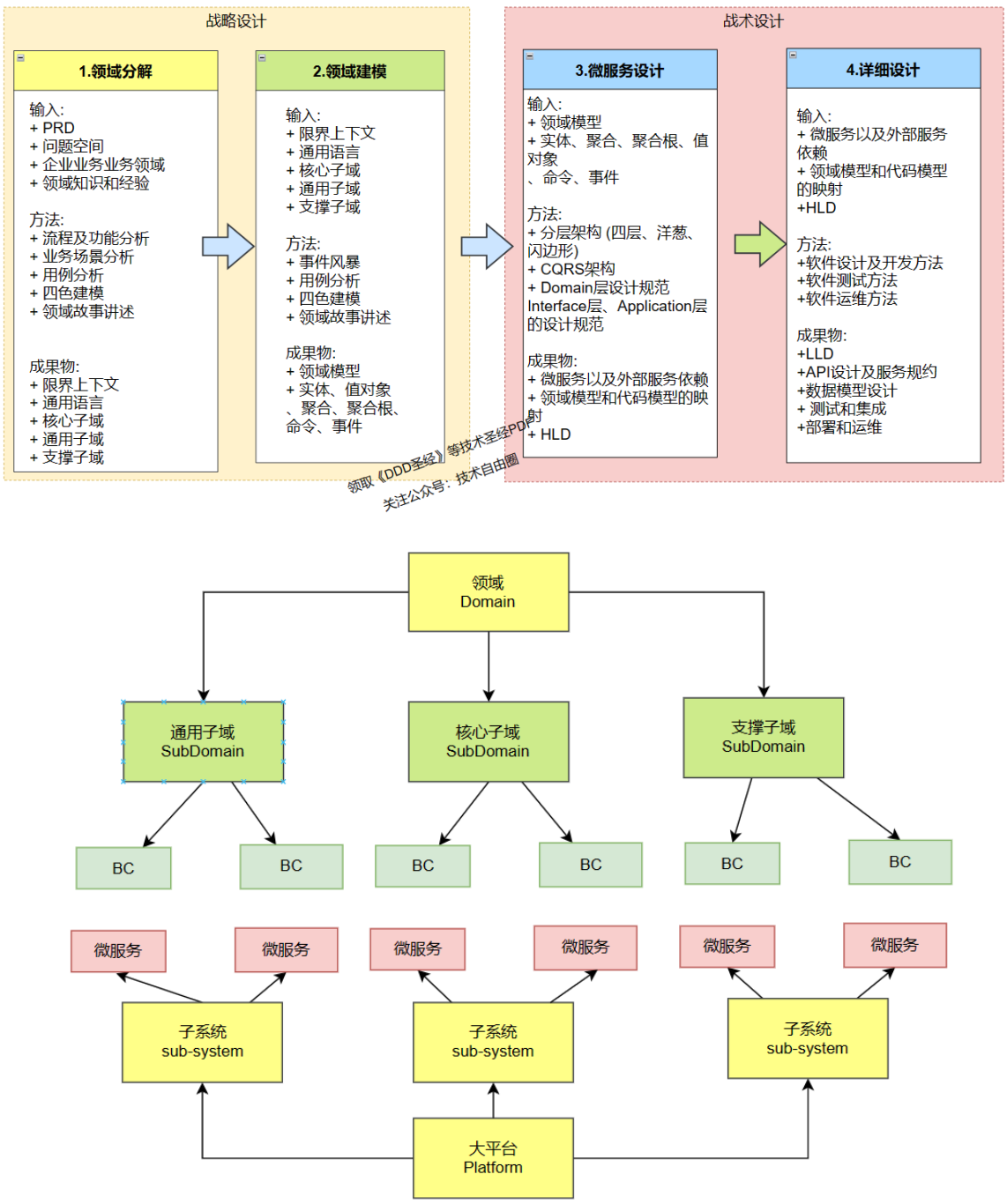
贫血模型与充血模型区别

区别	充血模型（DDD开发方式下	贫血模型（传统OOP开发方式下）
编码方式	以领域对象为主的行为状态转换	事务脚本（CRUD、判断、循环、检查等逻辑只是简单的方法级别拆分）
Service	跨领域逻辑的组合	事务脚本的对外包装
Domain	一个领域的entity集合包	VO/DTO
Entity	包含了一个领域里的状态、以及对状态的直接操作单元，具有自闭环的对象。	VO/DTO
ValueObject	简单的基本值对象，如电话号码、地址等，组成entity的基本单元	语言基本类型
Aggregation	（有点类似数学中集合的定义）将实体和值对象划分为聚合并围绕着聚合定义边界。	多个VO/DTO
Aggregation Root	唯一可以给外界作为某个Aggregation引用的entity。	单个VO/DTO

优缺点

- 贫血模型的优缺点
  - 优点1：很传统的编程思路，被许多程序员所掌握，许多教材采用的是这种模型，对于初学者，这种模型很自然，甚至被很多人认为是java中最正统的模型。
  - 优点2：思路清晰，事务边界清晰一般来说service的每个方法都可以看成一个事务，因为通常Service的每个方法对应着一个用例
  - 缺点：对象状态与行为分离，不能直观地描述领域对象。行为的设计主要考虑参数的输入和输出而非行为本身，不太具有面向对象设计的思考方式。
- 充血模型的优缺点
  - 优点：贫血model偏重个性化，面向过程式。 充血偏共性化，采用OO设计，类拥有其属性及对应的行为，通过将职责分配到相应的模型对象或Service，可以很好的组织业务逻辑，因此非常适合于复杂的企业业务逻辑的实现，以及可复用程度比较高。
  - 缺点1：领域驱动建模要求对领域模型完整而透彻的了解，只给出一个用例的实现步骤是无法得到领域模型的，这需要和领域专家的充分讨论。错误的领域模型对项目的危害非常之大，而实现一个好的领域模型非常困难。
  - 缺点2：对象高度自洽的结果是不利于大规模团队分工协作。一个编程个体至少要完成一个完整业务逻辑的功能。对于单个完整业务逻辑，无法再细分下去了。

问题7：DDD建模与微服务架构设计的关系?



以上内容没有文字，具体请参见 尼恩《DDD学习圣经》以及配套视频。

问题8： 什么是 CQRS(Command Query Responsibility Segregation)?

CQRS（Command Query Responsibility Segregation）是一种简单的设计模式。

CQRS衍生与CQS，即命令和查询分离，CQS是由Bertrand Meyer所设计。

按照这一设计概念，系统中的方法应该分为两种：改变状态的命令和返回值的查询。‘

Greg young将引入了这个设计概念，并将其应用于对象或者组件当中，这就是今天所要将的CQRS。

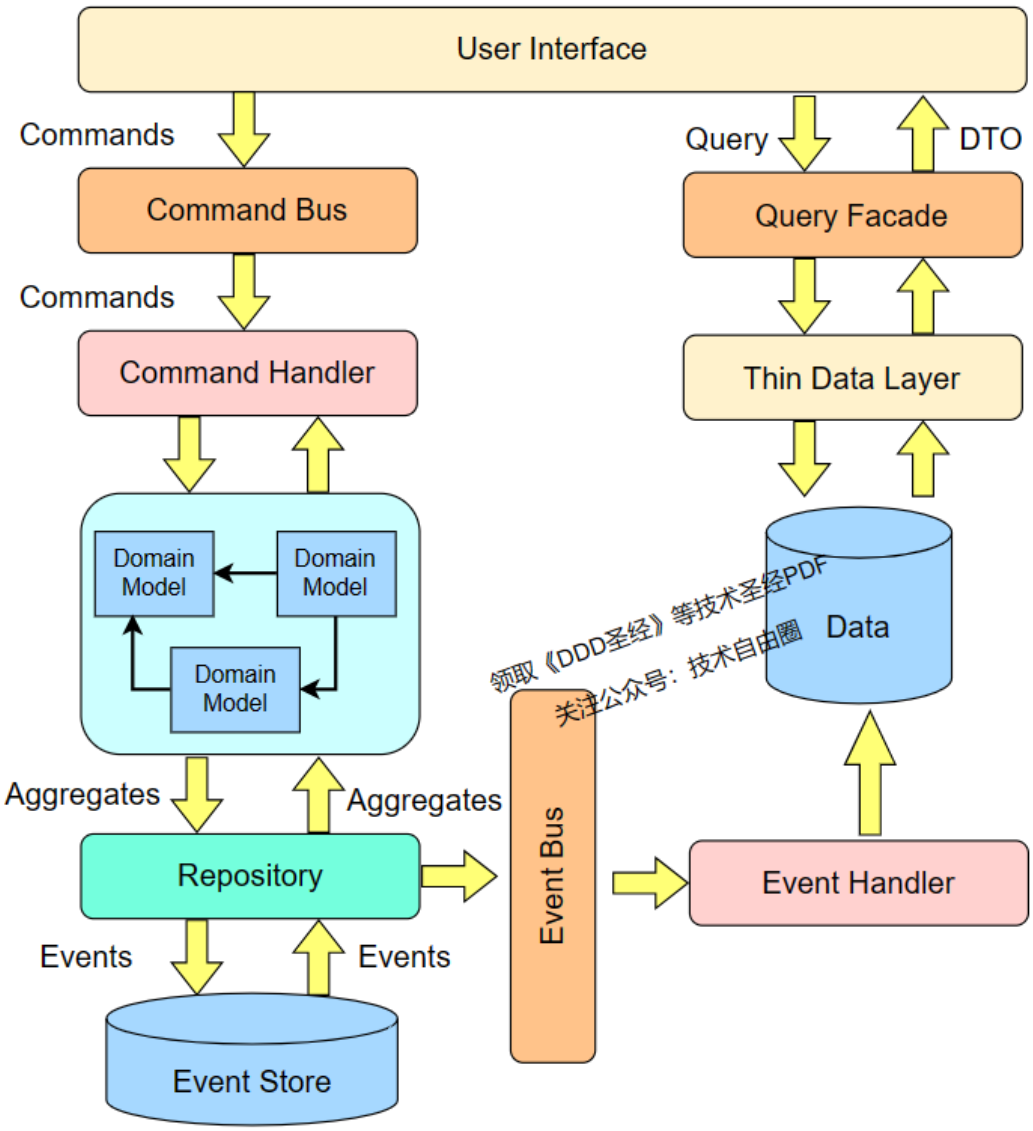
CQRS背后的主要思想是应用程序更改对象或组件状态（Command）应该与获取对象或者组件信息（Query）分开。

具体来说：CQRS(Command Query Responsibility Segregation)，Command 与 Query 分离的一种模式。

- Command：命令则是对会引起数据发生变化操作的总称，即新增，更新，删除这些操作，都是命令
- Query：查询则不会对数据产生变化的操作，只是按照某些条件查找数据

CQRS 的核心思想是将这两类不同的操作进行分离，可以是两个独立的应用，两个不同的数据源，也可以是同一个应用内的不同接口上。

CQRS架构模式，在DDD中是一种很常见的模式，它的用途在于将Command与Query功能进行分离，服务可以独立部署，也可以拆分部署。数据库可以使用一个，也可以读写分离。



从上图可看出，把数据的变更通过数据同步到另一个库用来查询数据，其实就是数据异构。

但这不是我们现在需要做的，我们是要利用CQRS的思想解决领域驱动中查询功能实现复杂的问题

CQRS 说白了，就是"数据查询"和"业务操作"分离。

在COLA 4.0之前，还有Command Bus和Query Bus 。**Command Bus（命令总线）**：是一种接收命令并将命令传递给命令处理程序的队列。**Query Bus（查询总线）**：是一种查询命令并将查询传递给查询处理程序的队列。

在COLA 4.0中，已经移除了Command Bus和Query Bus的处理，进一步简化了COLA架构。

具体的细节，具体请参见 尼恩《DDD学习圣经》以及配套视频。

问题9：微服务一定要DDD，为什么？TDD和DDD 有何关系?

具体答案，请参见此文： https://mp.weixin.qq.com/s/80Gza3-9pO8bYlSJoEDVWg

问题10：DDD架构，如何落地?

具体答案，请参见此文： https://mp.weixin.qq.com/s/_NAHYDIvmT7EbffXkIpqYw

问题11：DDD 领域层，该如何设计?

具体答案，请参见此文：https://mp.weixin.qq.com/s/njBl791vFCd94UplPJEFKw

问题12：微服务如何拆分？原则是什么?

具体答案，请参见此文： https://mp.weixin.qq.com/s/bExjRkVbDLTV2Wf9G6dzrg

问题13：给一个需求，请用DDD设计出来

具体答案，请参见此文： https://mp.weixin.qq.com/s/izzUXIpVmURq6hTCEN0ARQ

技术自由圈 技术圣经系列

技术自由圈 7个 学习圣经 PDF

1 《SpringCloud Alibaba 学习圣经》 377页 PDF
v1 版，发布日期：2023年03月15日

2 《Docker 学习圣经》 145页 PDF
v2 版，发布日期：2023年03月12日

3 《Kubernets 学习圣经》 530页 PDF
v1 版，发布日期：2023年04月11日

4 《响应式圣经》 114页 PDF
v2 版，发布日期：2023年02月14日

5 《缓存之王 Caffeine 红宝书》 175页 PDF
v2 版，发布日期：2022年11月30日

6 《队列之王 Disruptor 红宝书》 75页 PDF
v1 版，发布日期：2022年10月05日

7 《Prometheus+grafna 红宝书》 100页 PDF
v1 版，发布日期：2022年10月08日

技术自由圈 3个 高并发圣经 PDF

8 《Java高并发核心编程 卷1 加强版：NIO、Netty、Redis、ZooKeeper》 609页 PDF
加强版，发布日期：2023年01月01日

9 《Java高并发核心编程 卷2 加强版：多线程、锁、JMM、JUC、高并发设计模式》 448页 PDF
加强版，发布日期：2022年11月14日

10 《Java高并发核心编程 卷3 加强版：亿级用户Web应用架构与实战》 481页 PDF
加强版，发布日期：2022年11月14日

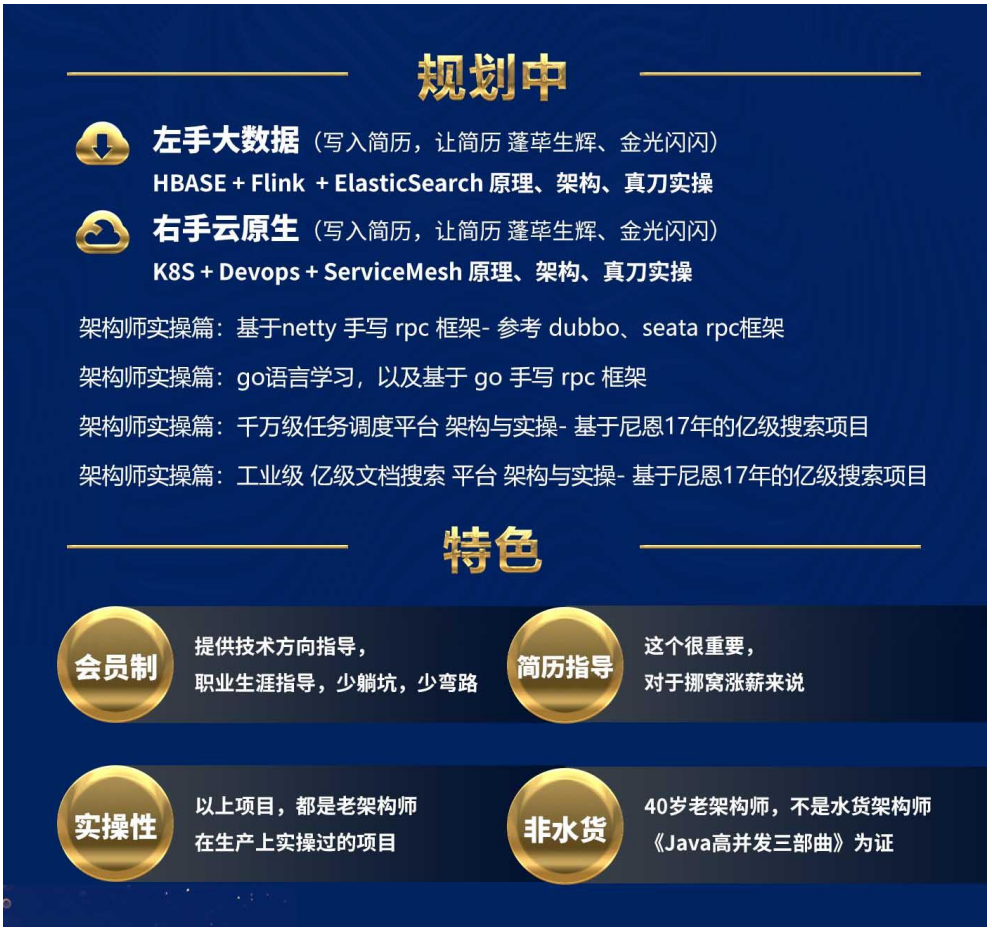
技术自由圈 面试圣经 40个 面试题 PDF

11 《尼恩 Java 面试宝典》40个专题 4000页 PDF
v60 版，发布日期：2023年03月11日

领取方式： 技术自由圈 公众号

硬核推荐：尼恩Java硬核架构班

详情：https://www.cnblogs.com/crazymakercircle/p/9904544.html


规划中
左手大数据（写入简历，让简历 蓬荜生辉、金光闪闪）
HBASE + Flink + ElasticSearch 原理、架构、真刀实操
右手云原生（写入简历，让简历 蓬荜生辉、金光闪闪）
K8S + Devops + ServiceMesh 原理、架构、真刀实操
架构师实操篇：基于netty 手写 rpc 框架- 参考 dubbo、seata rpc框架
架构师实操篇：go语言学习，以及基于 go 手写 rpc 框架
架构师实操篇：千万级任务调度平台 架构与实操- 基于尼恩17年的亿级搜索项目
架构师实操篇：工业级 亿级文档搜索 平台 架构与实操- 基于尼恩17年的亿级搜索项目
特色
会员制
提供技术方向指导，
职业生涯指导，少躺坑，少弯路
简历指导
这个很重要，
对于挪窝涨薪来说
实操性
以上项目，都是老架构师
在生产上实操过的项目
非水货
40岁老架构师，不是水货架构师
《Java高并发三部曲》为证

架构班（社群 VIP）的起源：

最初的视频，主要是给读者加餐。很多的读者，需要一些高质量的实操、理论视频，所以，我就围绕书，和底层，做了几个实操、理论视频，然后效果还不错，后面就做成迭代模式了。

架构班（社群 VIP）的功能：

提供高质量实操项目整刀真枪的架构指导、快速提升大家的：

- 开发水平
- 设计水平
- 架构水平

弥补业务中 CRUD 开发短板，帮助大家尽早脱离具备 3 高能力，掌握：

- 高性能
- 高并发
- 高可用

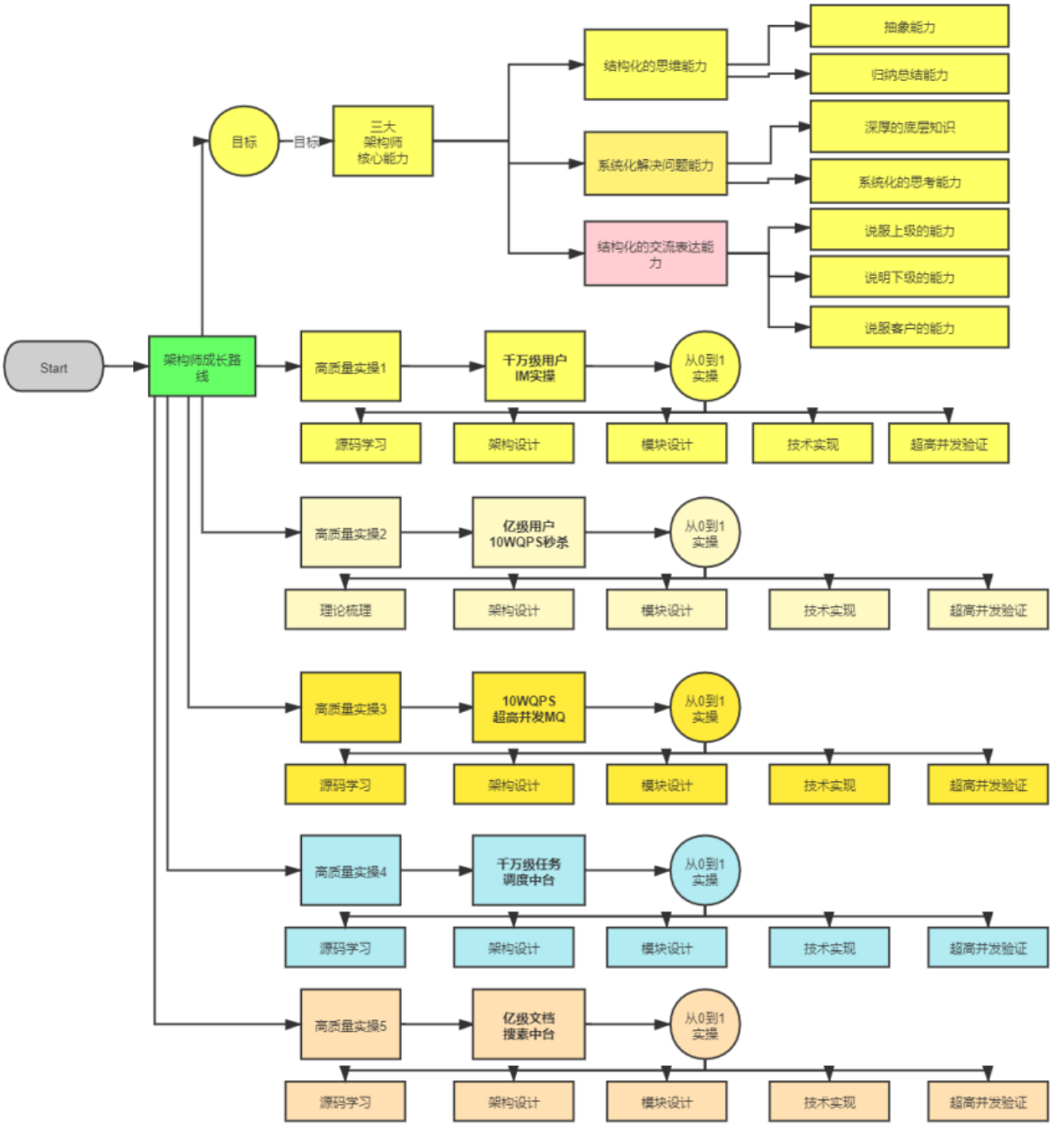
作为一个高质量的架构师成长、人脉社群，把所有的卷王聚焦起来，一起卷：

- 卷高并发实操
- 卷底层原理
- 卷架构理论、架构哲学
- 最终成为顶级架构师，实现人生理想，走向人生巅峰

架构班（社群 VIP）的目的：

- 高质量的实操，大大提升简历的含金量，吸引力，增强面试的召唤率
- 为大家提供九阳真经、葵花宝典，快速提升水平
- 进大厂、拿高薪
- 一路陪伴，提供助学视频和指导，辅导大家成为架构师
- 自学为主，和其他卷王一起，卷高并发实操，卷底层原理、卷大厂面试题，争取狠卷 3 月成高手，狠卷 3 年成为顶级架构师

N 个超高并发实操项目：简历压轴、个顶个精彩



【样章】第 17 章：横扫全网 Rocketmq 视频第 2 部曲：工业级 rocketmq 高可用（HA）底层原理和实操

工业级 rocketmq 高可用底层原理，包含：消息消费、同步消息、异步消息、单向消息等不同消息的底层原理和源码实现；消息队列非常底层的主从复制、高可用、同步刷盘、异步刷盘等底层原理。

工业级 rocketmq 高可用底层原理和搭建实操，包含：高可用集群的搭建。

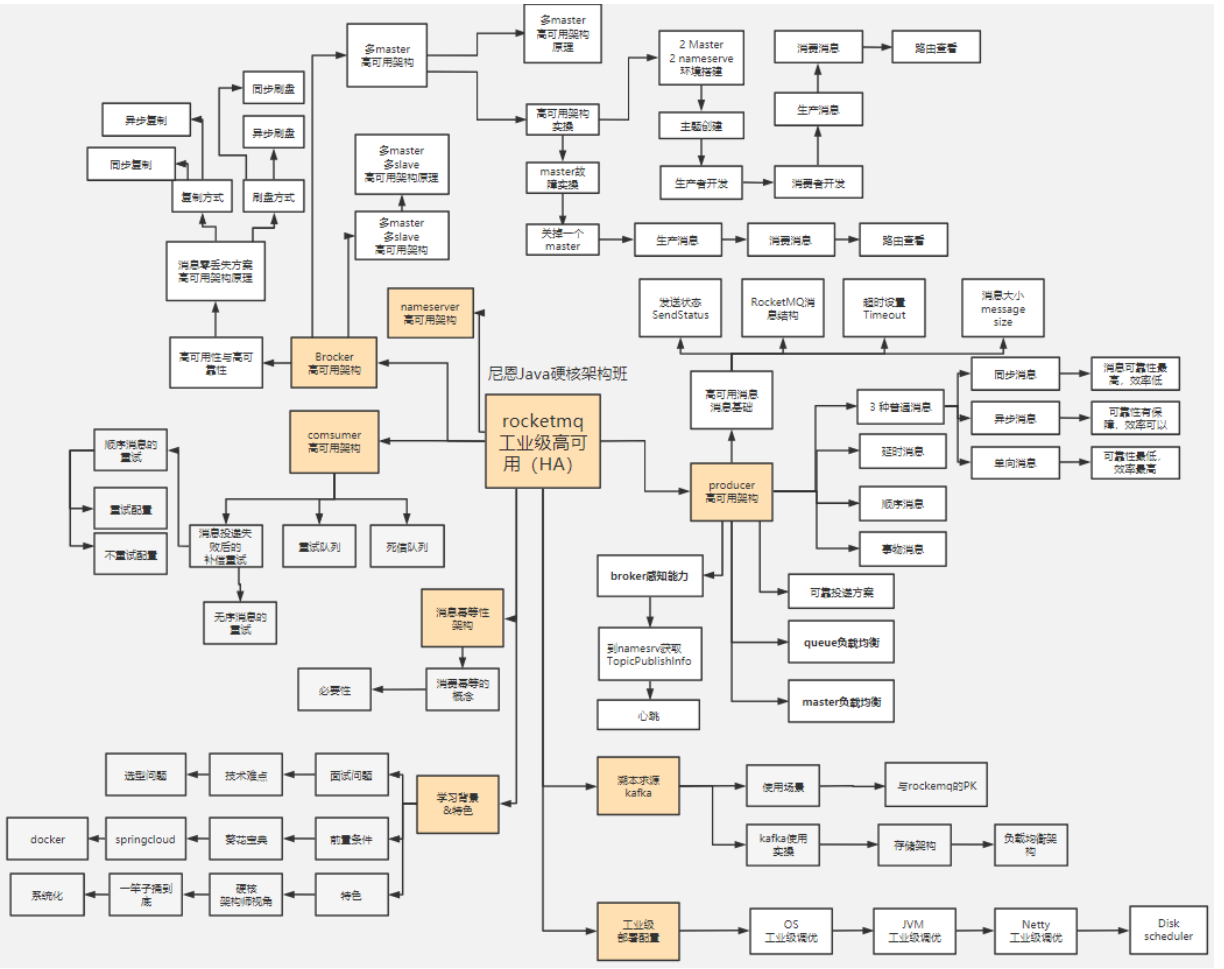
解决以下难题：

1、技术难题：RocketMQ 如何最大限度的保证消息不丢失的呢？RocketMQ 消息如何做到高可靠投递？

2、技术难题：基于消息的分布式事务，核心原理不理解

3、选型难题： kafka or rocketmq ，该娶谁？

下图链接：https://www.processon.com/view/6178e8ae0e3e7416bde9da19



成功案例：2 年翻 3 倍，35 岁卷王成功转型为架构师

详情：http://topcoder.cloud/forum.php?mod=forumdisplay&fid=43&page=1

最新　最后发表　热门　精华

成功案例：[1057号卷王] 3年小伙拿到外企offer，薪酬涨了200%
卷王1号　超级版主　前天 17:41

成功案例：[645号卷王] 4年经验卷王逆袭，被毕业后，反涨24W
卷王1号　超级版主　2022-9-21

成功案例：[878号卷王] 小伙8年经验，年薪60W
卷王1号　超级版主　2022-8-13

年薪70W案例：通过尼恩的指导，小伙伴年薪从40W涨到70W
卷王1号　超级版主　2022-2-11

成功案例：[493号卷王] 5年小伙拿满意offer，就业寒冬季逆涨30%
卷王1号　超级版主　前天 17:43

成功案例：[250号卷王] 就业极寒时代，收offer 涨25%
卷王1号　超级版主　前天 17:38

成功案例：[612号卷王] 就业极寒时代，从外包到自研
卷王1号　超级版主　前天 17:15

成功案例：[913号卷王] 热烈祝贺6年经验卷王，年薪40W
卷王1号　超级版主　2022-9-21

成功案例：[959号卷王] 4年经验卷王，喜获百度、Boss直聘等N个优质offer，最高涨100%
卷王1号　超级版主　2022-9-21

成功案例：[529号卷王] 5年经验卷王喜收2大offer，最高涨5K
卷王1号　超级版主　2022-9-21

成功案例：[811号卷王] 热烈祝贺7年经验卷王，薪酬涨30%
卷王1号　超级版主　2022-9-21

成功案例：[287号卷王] 不惧大寒潮，卷王逆市收4 offer，涨30%，可喜可贺
卷王1号　超级版主　2022-5-30

成功案例：[1002号卷王] 5月份"被毕业"，改简历后，斩获顶级央企Offer，涨薪7000+
卷王1号　超级版主　2022-7-5

成功案例：[7号卷王] 热烈祝贺小伙伴涨薪120%

卷王1号 超级版主 2022-8-13

成功案例：[134号卷王] 大三小伙卷1年，斩获顶级央企Offer，成功逆袭

卷王1号 超级版主 2022-7-6

成功案例：[1008号卷王] 5年经验卷王收42W offer，月涨8000，可喜可贺

卷王1号 超级版主 2022-5-30

成功案例：[453号卷王] 非全日制 6年卷王喜提3 offer，年薪30W，可喜可贺

卷王1号 超级版主 2022-5-21

成功案例：[924号卷王] 6年卷王喜提4 offer，最高涨薪9000，可喜可贺

卷王1号 超级版主 2022-5-21

成功案例：[15号卷王] 4年卷王入职 微软，涨薪50%，可喜可贺

卷王1号 超级版主 2022-5-12

成功案例：[527号卷王] 4年卷王喜提2 offer，涨薪50%，可喜可贺

卷王1号 超级版主 2022-5-13

成功案例：[788号卷王] 3年卷王喜提优质Offer，涨薪60%

卷王1号 超级版主 2022-5-11

成功案例：热烈祝贺：非全日制卷王，喜提2个心仪offer，面3家过2家

卷王1号 超级版主 2022-4-21

成功案例：[693号卷王] 二线城市6年卷王喜提4大优质Offer，含央企offer，最高薪酬35W

卷王1号 超级版主 2022-4-16

成功案例：[85号卷王] 双非2本小伙，春招大捷，喜提9个offer，最高薪酬近30万

卷王1号 超级版主 2022-4-14

成功案例：[741号卷王] 卷王逆袭！6年小伙从很少面试机会到搞定35K*14薪Offer

卷王1号 超级版主 2022-4-12

成功案例：[642号卷王] 热烈祝贺，6年卷王喜提优质国企offer

卷王1号 超级版主 2022-4-7

成功案例：[796号卷王] 热烈祝贺，36岁卷王喜提52万优质offer

卷王1号 超级版主 2022-3-25

成功案例：[15号卷王] 小伙卷1年，涨薪9K+，喜收ebay等多个优质offer

卷王1号 超级版主 2022-3-24

成功案例：[821号卷王] 小伙狠卷3个月，喜提10多个offer

卷王1号 超级版主 2022-3-21

成功案例：[736号卷王] 3年半经验收22k offer，但是小伙志存高远，冲击25k+

卷王1号 超级版主 2022-3-20

成功案例：热烈祝贺1群小卷王offer拿到手软，甚至拒了阿里offer

卷王1号 超级版主 2022-3-16

简历案例：简历一改，腾讯的邀请就来了！热烈祝贺，小伙收到一大堆面试邀请

卷王1号 超级版主 2022-3-10

成功案例：祝贺我圈两大超级卷王，一个过了阿里HR面，一个过了阿里2面

卷王1号 超级版主 2022-3-10

成功案例：小伙伴php转Java，卷1.5年Java，涨薪50%，喜收多个优质offer

卷王1号 超级版主 2022-3-10

成功案例：4年小伙狠卷半年，拿到 移动、京东 两大顶级offer

尼恩 超级版主 2022-3-5

成功案例：[267号卷王] 助力3年经验卷王，拿到蜂巢的17k x 14薪的offer

卷王1号 超级版主 2022-2-27

成功案例：[143号卷王] 二本院校00后卷神，毕业没到一年跳到字节，年薪45W

卷王1号 超级版主 2022-2-27

成功案例：[494号卷王] 尼恩分布式事务助力卷王拿到 中信银行offer

卷王1号 超级版主 2022-2-27

成功案例：[76号卷王] 2线城市卷王，狠卷1.5年，喜收22K offer

卷王1号 超级版主 2022-2-27

成功案例：[429号卷王] 小伙伴在社群卷5个月，涨8k+

卷王1号 超级版主 2022-2-27

成功案例：[154号卷王] 双非学校毕业卷王，连拿 京东到家&滴滴 两个大厂Offer

卷王1号 超级版主 2022-2-27

成功案例：[232号卷王] 涨薪10K，继续卷向食物链顶端

卷王1号 超级版主 2022-2-27

成功案例：狠卷1年技术，喜收 腾讯、阿里、微软三大Offer，最高年薪56W

卷王1号 超级版主 2022-2-27

成功案例：[449号卷王] 应届毕业卷王喜收 滴滴offer, 年薪33W

卷王1号 超级版主 2022-2-27

成功案例：[551号卷王] 小伙伴学完后，成功进入大厂，并且推荐自己的朋友加VIP学习

卷王1号 超级版主 2022-2-10

成功案例：[214号卷王] 助力2年经验卷王，成功拿到17K月薪

卷王1号 超级版主 2022-2-10

成功案例：[92号卷王] 课程实操助力社群小伙伴喜收 喜马拉雅Offer

卷王1号 超级版主 2022-2-10

成功案例：社群卷王小伙伴成功过了滴滴三面 获滴滴Offer

卷王1号 超级版主 2022-2-10

[612号卷王]滴滴小伙伴，蹲点考察半年，觉得靠谱后加入 疯狂创客圈

卷王1号 超级版主 2022-2-10

成功案例：[732号卷王] 尼恩助力3年经验卷王收获 京东offer, 年薪35W

卷王1号 超级版主 2022-2-27

成功案例：[558号卷王] 2年经验卷王，喜收 网易和阿里子公司两个优质offer

卷王1号 超级版主 2022-2-27

成功案例：[569号卷王] 双非应届生卷王, 喜收字节跳动实习offer

卷王1号 超级版主 2022-2-25

成功案例：[420号卷王] 狠卷1年，卷王涨薪80%，涨薪12000元!

卷王1号 超级版主 2022-2-25

成功案例：[76号卷王] 通过尼恩1年半的指导，专科学历小伙伴从0.8K涨到22K

卷王1号 超级版主 2022-2-10

简历优化后的成功涨薪案例（VIP含免费简历优化）


9年 小伙伴拿到
年薪90W offer
9月11日改简历
11月29日晒offer
秘诀:
简历指导+ 狠狠卷
这个是给你微调的
省略号的地方，需要你再补充一点
上面你留着这个就行
Java 开发 9年-修改.docx
34.1 KB
薪资高 稳
主要的工作是啥?
提升了自己的实力，就不用怕
易所 关于数字货 的 易
po 也有 打盹的时候，该裁员，照样一个不少
年包比 po 多 19w
这么多
po 估计有 70 万
那你不有 90 个 W?
po 给我 68
就是吗


小伙8年经验
年薪60w
7月12日改简历
8月10日晒offer
秘诀:
改简历 + 狠狠卷
明天晚上哈
好哒
恩哥，今晚还改简历吗?
今晚还在外边应酬，估计回去比较晚
要不，咱们延迟到明天，如何
明天白天也行
好的，白天吧，答应别人明天给他们简历了。
OK
那就上午 11 点左右哈
好的
涨幅多少呀?
或者，额度多少呀
之前 36*15，现在这个 39*15
今年行情不太好，还有一些 offer 基本都是平薪，或降薪的。
OK
这个马上来
刚在指导简历
哈哈
稍等哈
辛苦啊恩哥
这次找工作，您的 起到效果了。
我这次复习基本看的都是咱们课程的 。


6年小伙伴
年薪40w
9月6日改简历
9月21日晒offer
秘诀:
简历指导+ 狠狠卷
Java - 6年.docx
24.7 KB
恩哥，简历我改好了，您再帮我看一下
Java - 6年(1).docx
24.5 KB
恩哥，看第二个吧
给你前面调整了一下
涨这多
今年这行情，也算可以了
总包多少呀，让我也陶醉一下
大概 40w 吧
谢谢恩哥的指导和鼓励
是在深圳
深圳的行情尤其难
能有面试电话就不错啦
是的
太牛啦
哈哈哈哈，恩哥的鼓励指导也很重要


5年小伙喜提3个offer
年薪 35个W
5月22日改简历
11月29日晒offer
我简历还没写出来 不然可以给老师您看看
算法方面，可以找点 刷刷，当作复习
项目的话，再好好改造一下
好的 谢谢老师
java 开发-本科统招-2022-5 年经验.docx
36.0 KB
OK
我排好号，回头找你哈
秘诀:
简历指导+ 狠狠卷
尼恩老师晚上好，汇报下最近的 offer 情况。最近面试收到了三个 offer，两个是平薪，一个是跨境电商公司的 offer，涨幅暂时不到 20%。通过这次面试也让我知道自己距离高级开发还有一点点距离，还要再多看才能突破。
最后结合自己的情况，先选择去跨境电商的公司再提升下
之前是 20k*13.5，跨境电商这个是说 23k*（14-15）
恭喜恭喜
就业寒冬，能提到 3 个 offer，已经很 ，已经是高手了
很多小伙伴，面试电话一个都接不到，简历海投 7000 份，只收到 3 次面试机会，没有一个机会拿到最终 offer
您这个年薪，算下来也有 35W 了吧
年终部分还要确认下，大部分人听说只有两个月
这个时间点，拿到这个水平，挺不错的啦
持续加油卷哈
嗯嗯！看看明年自己有没有能力冲击高开
把你我视频都吃透
辛苦老师再帮忙指导下哈


1.5年小伙搞定15K offer
就业寒冬涨100%
5月7日改简历
11月21日晒offer
秘诀：
简历指导+狠狠卷
多谢啦
你好，好多公司都觉得我的简历太简单了，我感觉是不是没法优化了
简历.pdf
397.4 KB
好的
晚上指导你改
推送中台需要加不
我还没做啊，准备八股文暂时没时间搞这些，入职了会开始搞
OK
还有其他的项目吗
一个项目，太少啦
那种练习项目也行
他那个不止两倍吧，我才是两倍，原先7k，现在15k
那也很牛
都翻倍了，都是牛人
正常就30%
慢慢来
下一步可以瞄准25k
其实我工作一年
那你就更牛逼拉
一年经验，搞定15k offer，舍你其谁


卷王逆袭成功案例
6年小伙从很少面试机会到
搞定35K*14薪
3月5日改简历
4月11日拿offer
面试邀请少
小伙很苦恼
理想很丰满
目标35K
我要努力争取35+
我指导简历，非vip收费3000，值这个价吧？
一个月拿到了
理想的offer
面试法宝
rocketmq四部曲

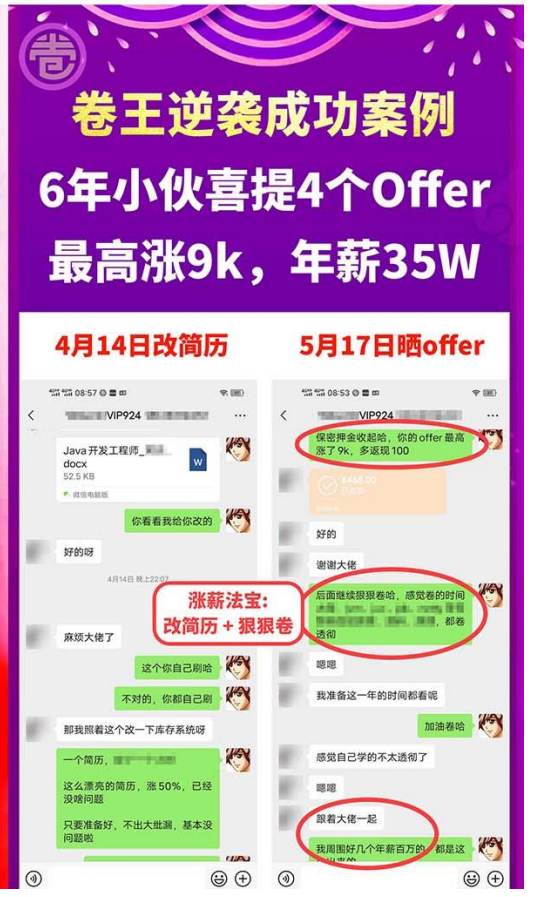

6年 经验小伙伴
喜收25K offer
3月12日改简历
12月1日晒offer
秘诀：
简历指导+狠狠卷
老师
咱们以这个项目为模板改哈
嗯好
项目有多少人，你带领多少人？
你工作几年了？
6年了
项目有多少人，你带领多少人？
语音交流一下
14:48
好
你方便吗
OK
不知道往哪个方向发展
改简历挪窝，20 涨到 25 k，高级开发，P7 带的后端团队
任务重，道路远，并不是没有方向


7年经验卷王
薪酬涨30%
7月11日改简历
9月1日晒offer
秘诀：
改简历 + 狠狠卷
咱们开始修改哈
要语音还是打字也行，还没到家
解决过最复杂的难题是啥？说3个哈
看看，有没有问题？
没问题
看看，有没有问题？
只能挪窝
下次你喊足一点哦
只要本事好，拿 offer 比较容易的
喊了，但是我流水太差了。。。我这家是补贴，银行不认这玩意，只能在上家的基础上加到 30% 封顶（特别优秀除外，我只是不错。。。）
入职了，最终并不能大幅涨薪，还是旧的太保，只涨了 30%
恭喜一下
现在不比往年
能涨 30% 是大牛
拿到 offer 都算大牛
现在很多人投几百发，连个电话都没有
你已经很


4年经验卷王逆袭
被毕业后，反涨24W
7月改简历
8月30日晒offer
秘诀:
改简历 + 狠狠卷
这就是你的简历
差得太多啦
ok
总共写了四个项目，最近的一年的还没补充上
你是在职，还是离职呢?
离职
原因大概是啥?
这个地方要这样写
项目被终止
方便语音沟通不
ok
是的 经你这么指导，非常重要
老哥 我八月十号开始找工作，今天已经入职了
现金基本持平，股票 +24W
总计涨了多少呢
能涨24W
股票这个吧只能到手了才算
也不错啦
很多小伙伴，面试机会都没有
感谢老哥的指导，继续跟你卷技术
继续狠狠卷哈，马上就技术自由拉

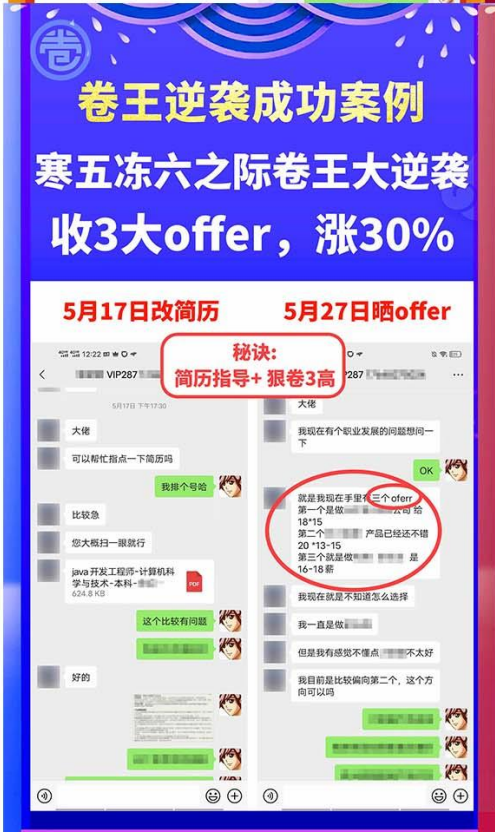

小伙5月份"被毕业"，改简历后
斩获顶级央企Offer
涨薪7000+
5月29日改简历
7月5日晒offer
秘诀:
简历指导+ 狠卷3高
快速看书，就要不求甚解，把目录和 场景大概一下，然后重点的地方，用到的地方，再去回炉
尼恩
我被 " 毕业 " 了
这周末或下周我你改一下简历
毕业了没有关系
ok，发我吧
it行业，跳来跳去，太频繁啦
嗯，其实有点心理准备
简历
不太会写简历
尼恩
我拿到 的offer 了
涨20%，
不还价的
看起来 不差钱呀
超过了8000没
平均算下来
7000多
好的
有啥面试的心得吗
可以分享给其他小伙伴的


卷王逆袭成功案例
武汉6年喜收4个优质offer
最高的年薪35W
2月9日改简历
4月15日晒offer
面试法宝:
改简历 + 实操
尼恩老师，新年好!
能帮忙修改下简历吗?
金三银四准备跳了
可以的
java 开发-6 年-简历-
.pdf
拜托了，恩师。希望能拿 25k 回来给你报喜
好，我加一下
还有吗?
恩哥，决赛圈 offer 了
截图是以我目前认知能写出来的评分了，麻烦帮我参考下
选择大于努力，恩师助我上岸
这么多 offer，我看着哈
都是恩哥指点有方，本来还有个新能源汽车的，35w 给拒了，主要太远了
跟着恩哥卷的时间太短了，目前实力也只能到这儿了
这里边有个大数据的，感觉也不错


卷王逆袭成功案例
6年小伙喜提4个Offer
最高涨9k，年薪35W
4月14日改简历
5月17日晒offer
涨薪法宝:
改简历 + 狠狠卷
Java 开发工程师_
.docx
你看看我给你改的
好的呀
麻烦大佬了
这个你自己刷哈
不对的，你都自己刷
那我照着这个改一下库存系统呀
一个简历，
这么漂亮的简历，涨 50%，已经没啥问题
只要准备好，不出大纰漏，基本没问题啦
保密拼金收起哈，你的 offer 最高涨了 9k，多返现 100
好的
谢谢大佬
后面继续狠狠卷哈，感觉卷的时间 都卷透彻
嗯嗯
我准备这一年的时间都看呢
加油卷哈
感觉自己学的不太透彻了
嗯嗯
跟着大佬一起
我周围好几个年薪百万的，都是这


卷王逆袭成功案例
5年经验小伙收2个offer
最高涨薪8k，年薪42W
5月9日改简历
5月30日晒offer
秘诀:
简历指导+ 狠卷3高
以此为样
大家狠狠卷
打造最卷IT社群


卷王逆袭成功案例
非全日制 6年经验卷王
喜提3个Offer，年包30W
5月9日改简历
5月18日晒offer
面试法宝:
改简历 + 狠狠卷


卷王逆袭成功案例
寒五冻六之际卷王大逆袭
收3大offer，涨30%
5月17日改简历
5月27日晒offer
秘诀:
简历指导+ 狠卷3高


卷王逆袭成功案例
4年卷王入职微软，涨50%
3月7日改简历
5月12日晒offer
涨薪法宝:
改简历 + 狠狠卷


4年小伙喜收百度、Boss直聘
等N个顶级Offer
最高涨幅100%
6月27日改简历
9月19日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
4年卷王入收2个offer，涨50%
3月23日改简历
5月12日晒offer
涨薪法宝:
改简历＋狠狠卷


小伙大三暑期很焦虑
跟着尼恩卷一年
校招斩获顶级央企Offer
去年5月19日加入VIP群
今年7月5日晒offer
秘诀:
狠狠卷 书+视频


小伙高中学历
薪酬涨120%
5月6日改简历
7月22日晒offer
秘诀:
改简历＋狠狠卷


卷王逆袭成功案例
非全日制卷王 面试3家
收2个offer 涨薪30%
4月13日改简历
4月21日晒offer
面试法宝：
改简历 + 面试题


5年卷王喜收2大Offer
最高涨5K
5月19日改简历
9月13日晒offer
秘诀：
改简历 + 狠狠卷


卷王逆袭成功案例
3年经验卷王，涨60%
4月16日改简历
5月11日晒offer
涨薪法宝：
改简历 + 狠狠卷


卷王逆袭成功案例
双非二本小伙春招大翻身
喜提9大offer
2月22日改简历
4月13日晒offer
面试法宝：
改简历 + IM实操
9大offer 最高年薪30万

修改简历找尼恩（资深简历优化专家）

- 如果面试表达不好，尼恩会提供 简历优化指导
- 如果项目没有亮点，尼恩会提供 项目亮点指导
- 如果面试表达不好，尼恩会提供 面试表达指导

作为 40 岁老架构师，尼恩长期承担技术面试官的角色：

- **从业以来，“阅历”无数，对简历有着点石成金、改头换面、脱胎换骨**的指导能力。
- **尼恩指导过刚刚就业的小白，也指导过 P8 级的老专家，都指导他们上岸。**

如何联系尼恩。尼恩微信，请参考下面的地址：

语雀：https://www.yuque.com/crazymakercircle/gkkw8s/khigna
码云：https://gitee.com/crazymaker/SimpleCrayIM/blob/master/疯狂创客圈总目录.md