基于DDD思想的应用架构COLA以及华为GTS的落地实践

华为GTS 主任工程师 / 张超

InfoQ 传媒和整合营销服务

对技术人群极具影响力的新闻网站 / 技术社区

InfoQ 是一家全球性的在线新闻 / 社区网站，创立于 2006 年，创始人是 Floyd Marinescu。目前全球拥有英、法、中、日共五种语言的站点。InfoQ 中国于 2007 年由极客邦科技创始人兼 CEO 霍太稳引入中国。

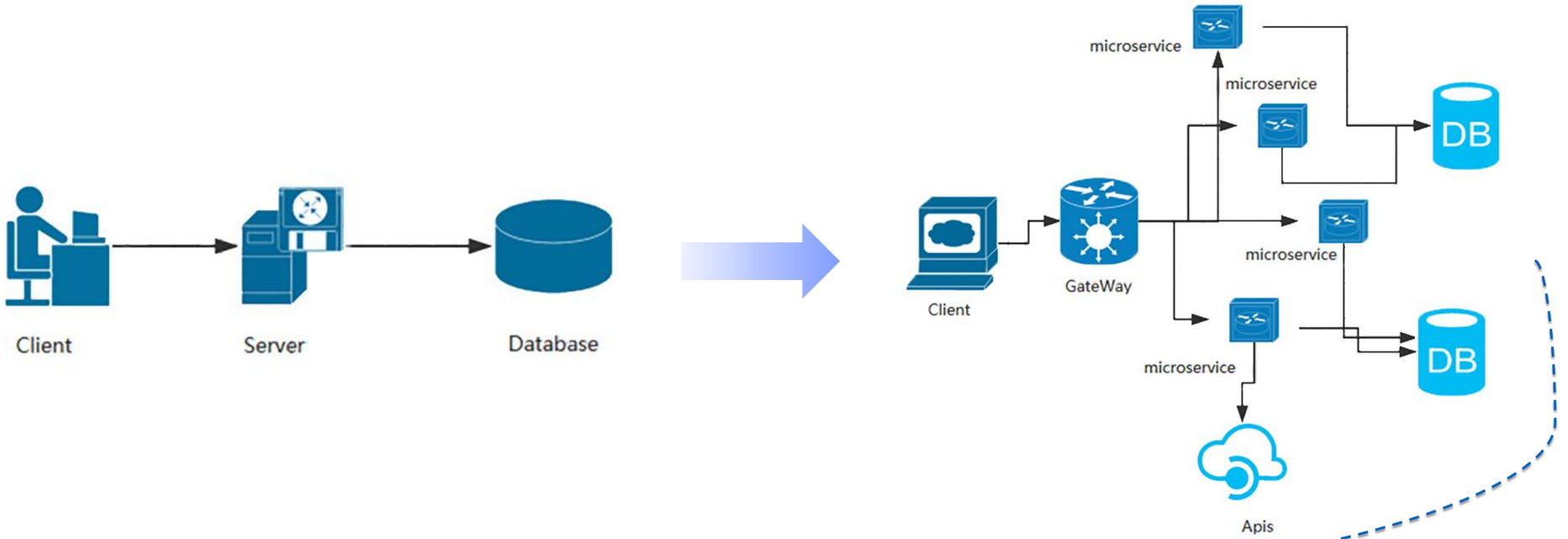
十五年来，InfoQ 致力于促进软件开发及相关领域知识与创新的传播，凭借在技术服务领域的深耕。

300W+	150W+	100W+	300W+	1600+
InfoQ 网站 月访问量	积累公众号 粉丝	微博 粉丝	覆盖中高端 技术开发者	CTO、 技术高管

大纲

- 从DDD思想到应用架构
- COLA应用框架
- 华为GTS的落地实践
- 思考与总结

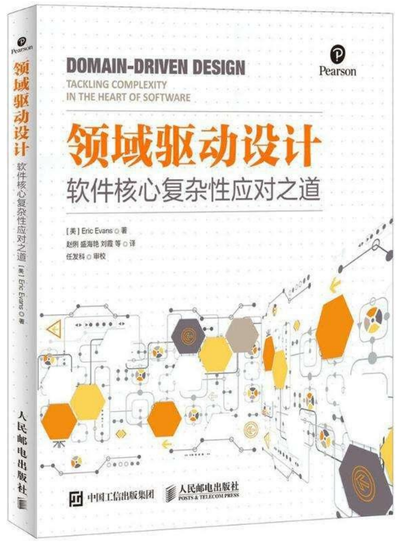
随着微服务架构的兴起，DDD正在焕发青春



- 微服务的边界如何划分？
- 微服务之间的依赖如何解耦？
- 微服务本身架构如何实现？
- 需求理解不正确、需求变化频繁、新人接手慢
- …

- ✓ 边界上下文（Bounded Context）
- ✓ 上下文映射（Context Map）
- ✓ 分层架构 （Layered Architecture）
- ✓ 统一语言、良好的架构和设计
- ✓ …

我们眼中的DDD



事件风暴 → 命令风暴 → 寻找实体 → 领域划分

- 领域事件
- 领域
- 命令
- 子域
- 实体
- 支撑域
- 值对象
- 通用域
- 聚合
- 限界上下文
- 聚合根
- 统一语言
- ...

概念多、操作复杂、落地难!

DDD落地困难综合征

DDD作为一种优秀的设计思想，的确为复杂业务治理带来了曙光。然而因为DDD本身难以掌握，很容易造成DDD从理论到工程落地之间出现巨大的鸿沟。**导致很多DDD项目不仅没有治理复杂，反而造成了更多的复杂**。

信心满满

开始腾飞

跪了…

DDD理论

工程落地

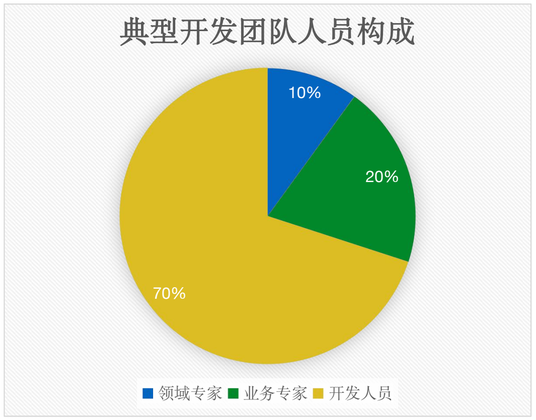
DDD工程落地的TOP困难?

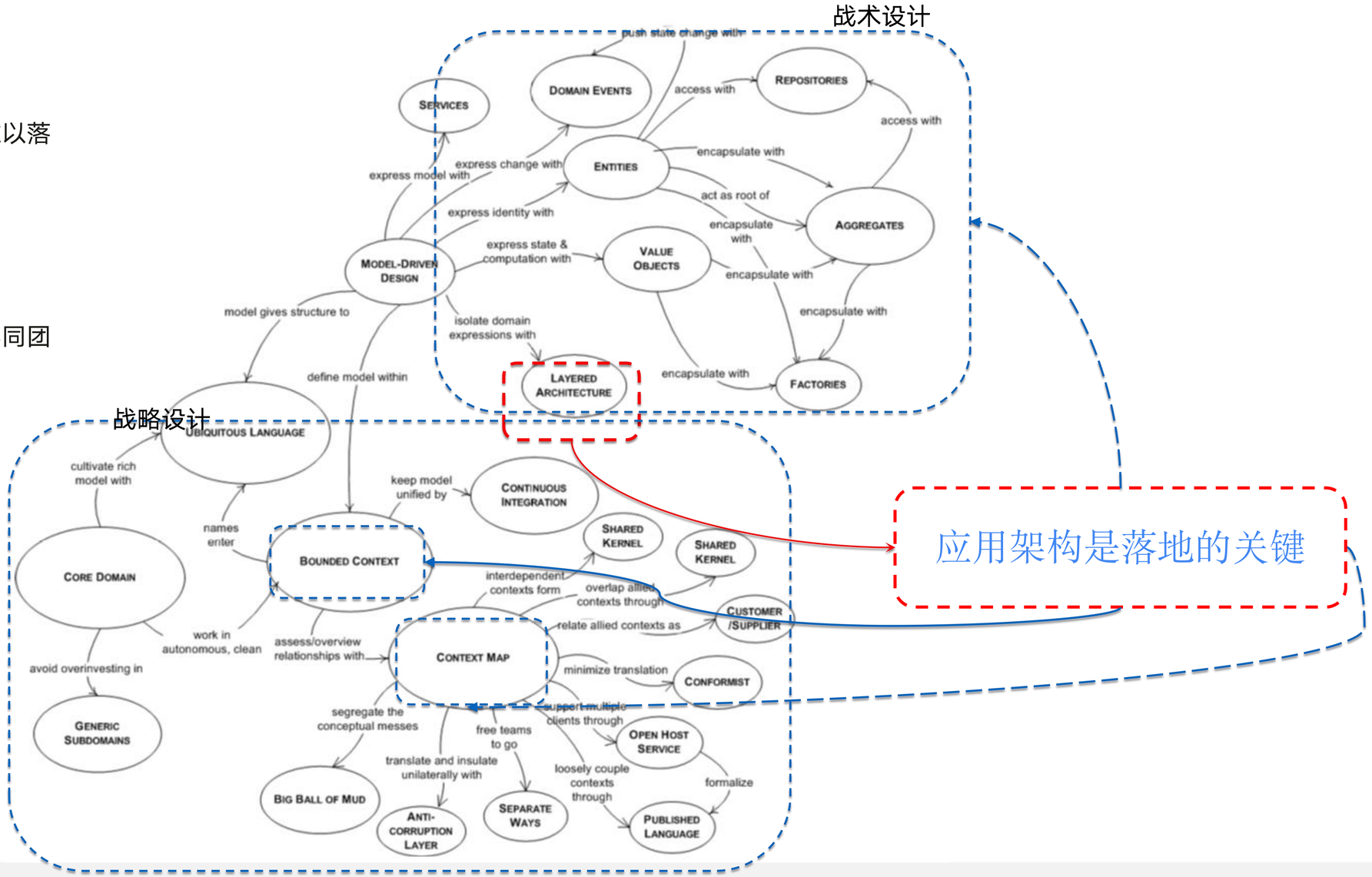
DDD认知层面：

- DDD概念复杂难以掌握；
- 团队上下认知不一，投入无法保证，难以落入开发流程；

工程实践层面：

- 微服务如何划分，依赖如何解耦？
- 实现过程包、类无较为通用的规范，不同团队差异较大；
- 包结构、类关系复杂、新人接手困难，
- …





什么是架构?

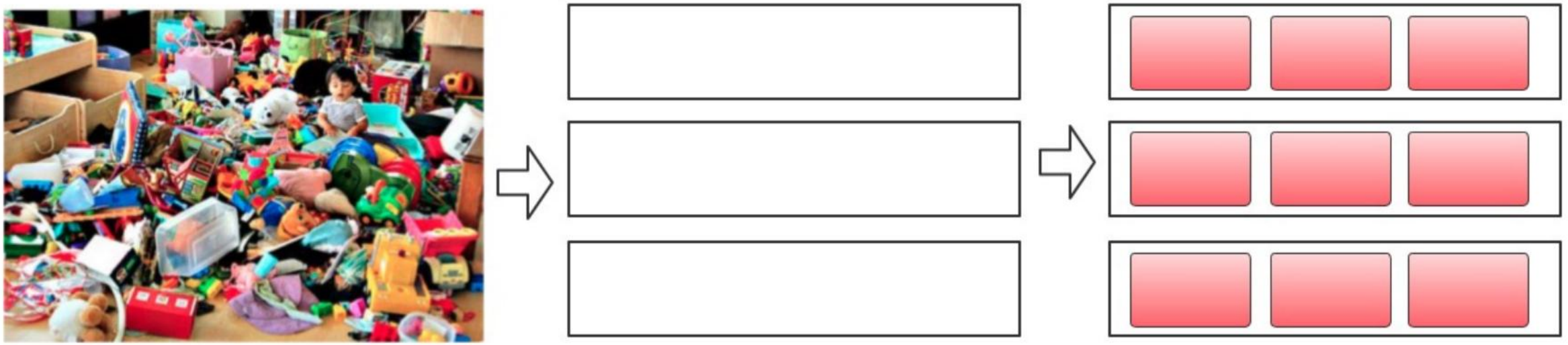
IEEE关于架构的定义：http://www.iso-architecture.org/ieee-1471/defining-architecture.html

Architecture: The fundamental organization of a system, embodied in its components,
their relationships to each other and the environment, and the principles governing its design and evolution.

架构：系统的基本组织，体现在其组件中， 它们与彼此和环境的关系， 以及支配其设计和演变的原则；

应用架构目的—秩序

软件的要素是代码，应用架构就是要解决代码如何被组织划分、以及后续演进规范指导的问题。



无序　　分层　　分层 + 分包

大纲

- 从DDD思想到应用架构
- COLA应用框架
- 华为GTS的落地实践
- 思考与总结

COLA简介

Clean Object—oriented & Layered Architecture

Github 8k Star & 2k Forks

https://github.com/alibaba/COLA

COLA: Clean Object-oriented & Layered Architecture

architecture clean cola

Readme

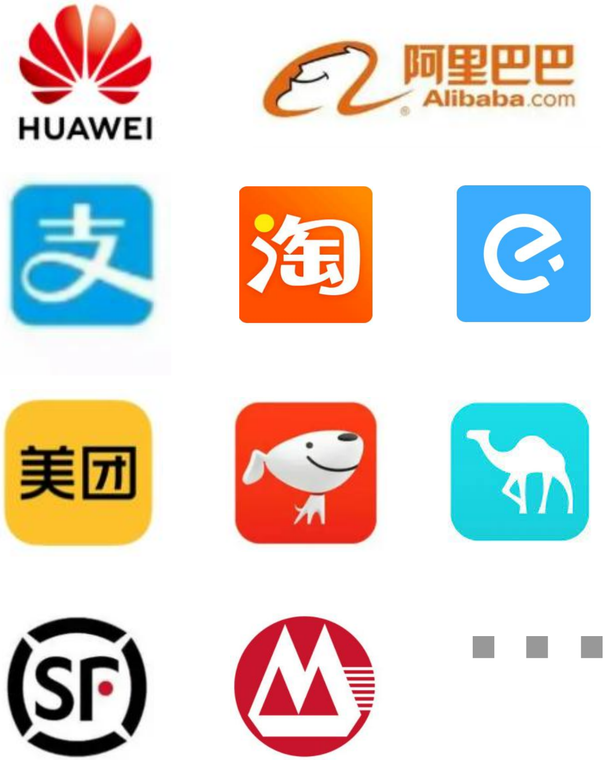
LGPL-2.1 license

8k stars

319 watching

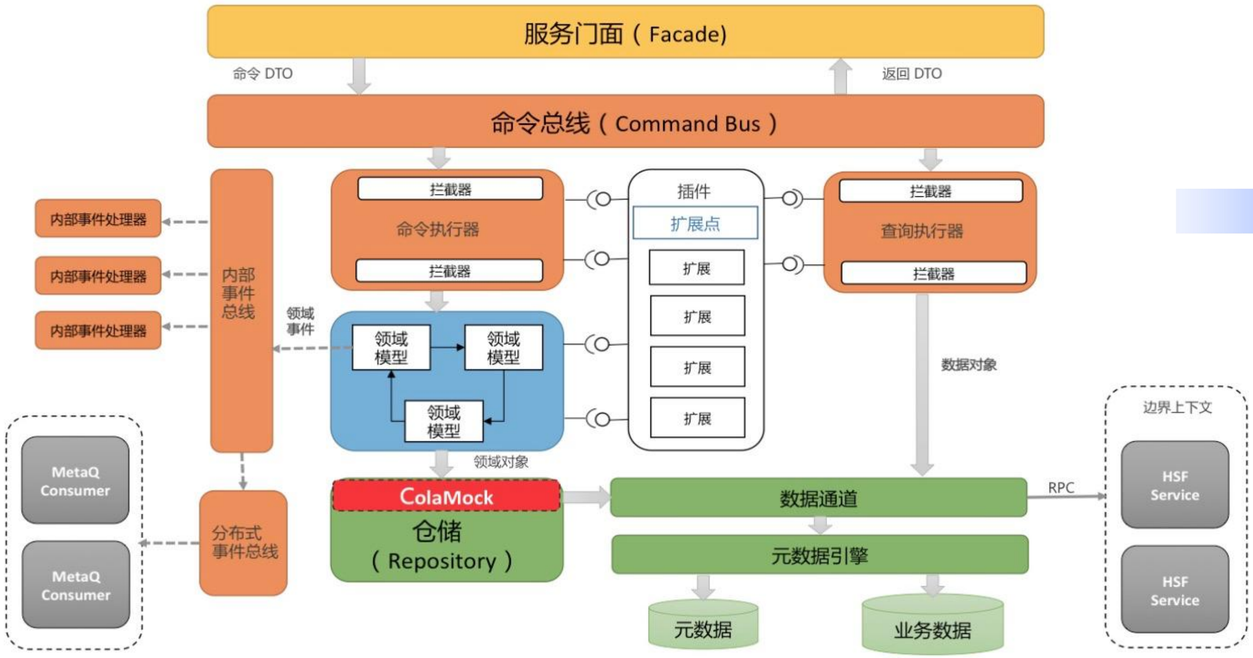
2.2k forks

- cola-archetypes
- cola-components
- cola-samples
- scripts

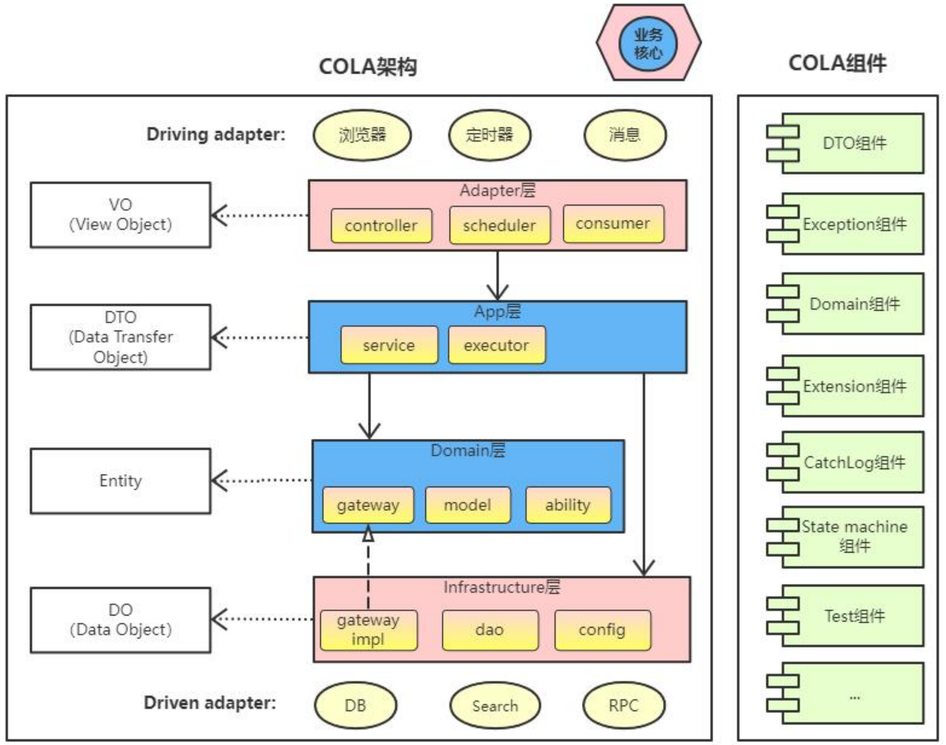


COLA 4.0 整体架构

COLA v1.0

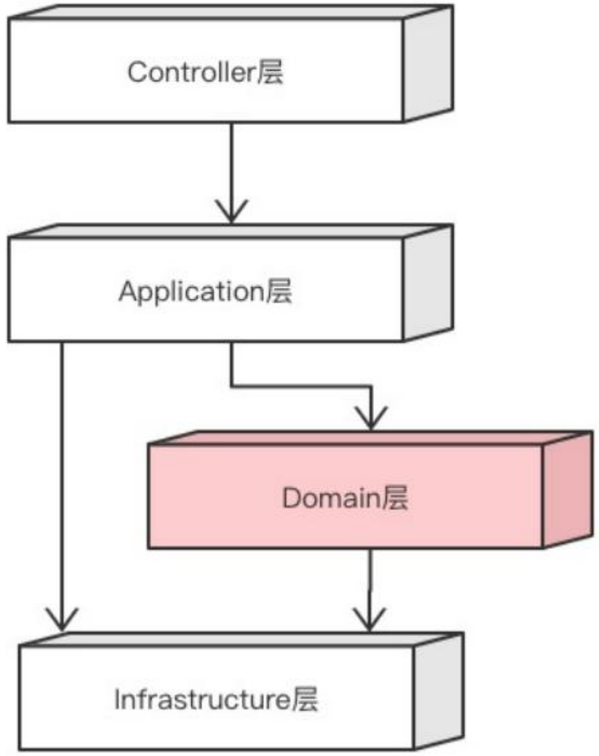


COLA v4.0



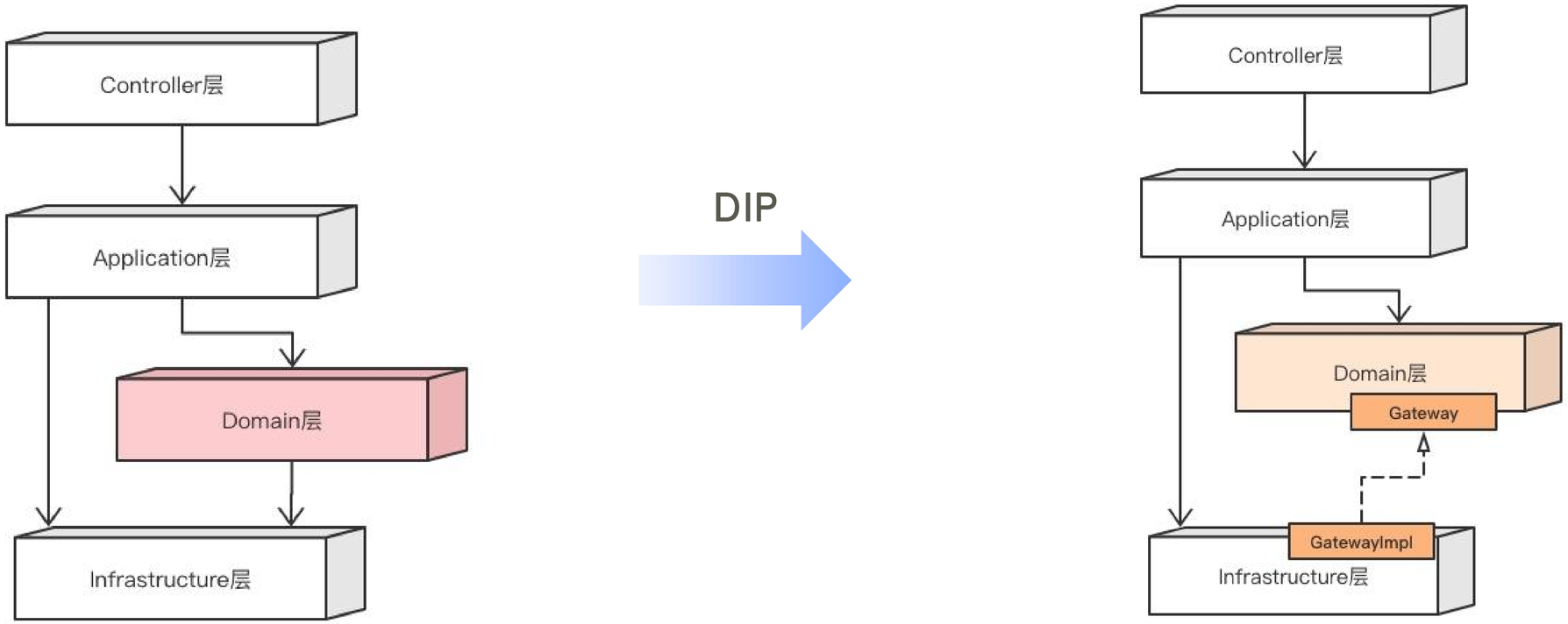
COLA分层设计

分离关注点，让每一层只解决该层关注的问题，从而将复杂的问题简化，起到分而治之的作用。



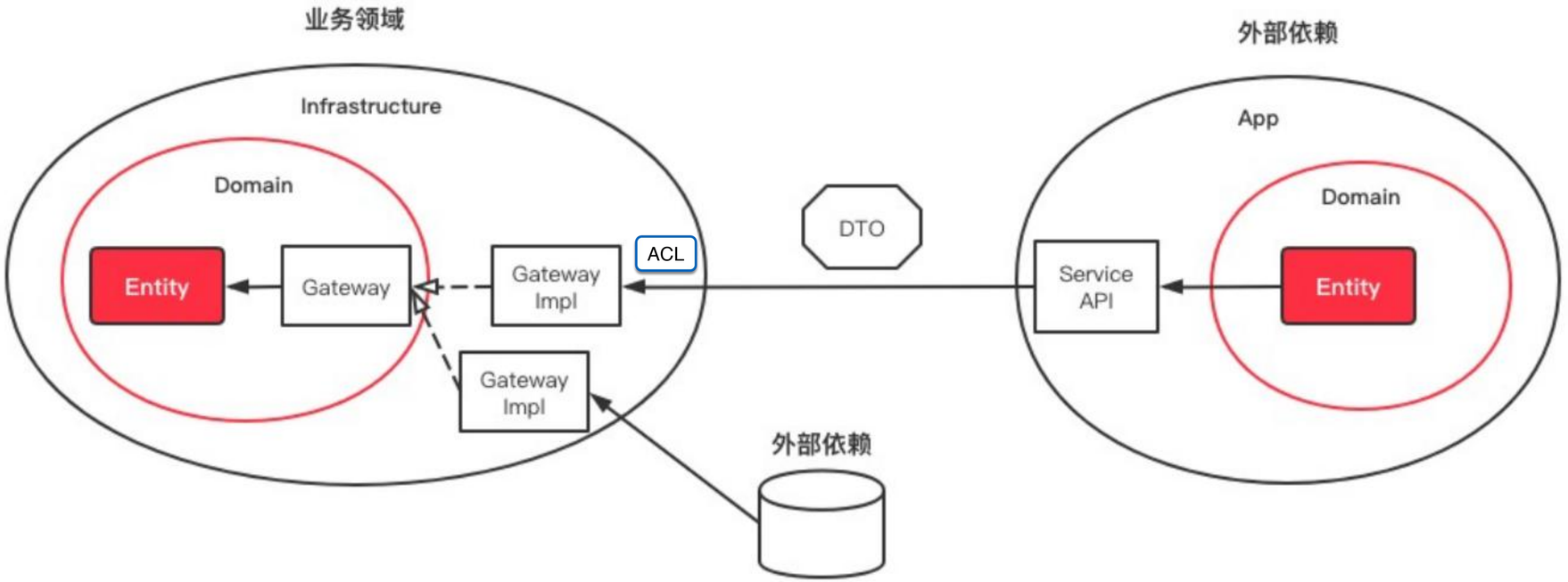
COLA解耦设计

领域层和基础设施层的依赖通过Gateway实现依赖反转，Domain摆脱对技术细节的依赖，专注处理业务逻辑。



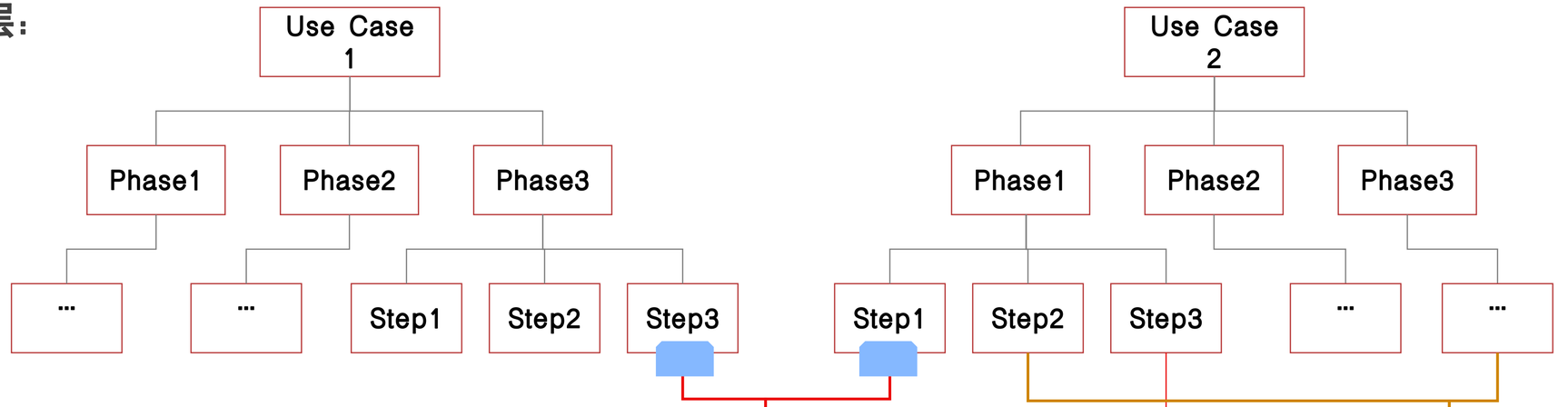
COLA解耦设计

通过Gateway进行解耦、转义，从而确保本领域的概念完整和独立，起到防腐和隔离的作用。**不能让外领域随便“入侵”到本域**。

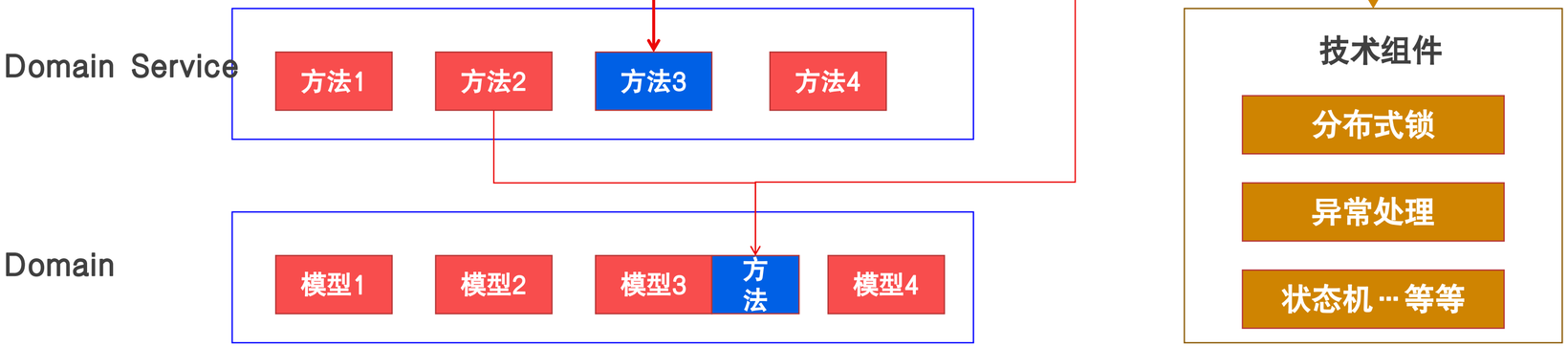


COLA实践：方法论总结——上下结合，循序渐进

App层：

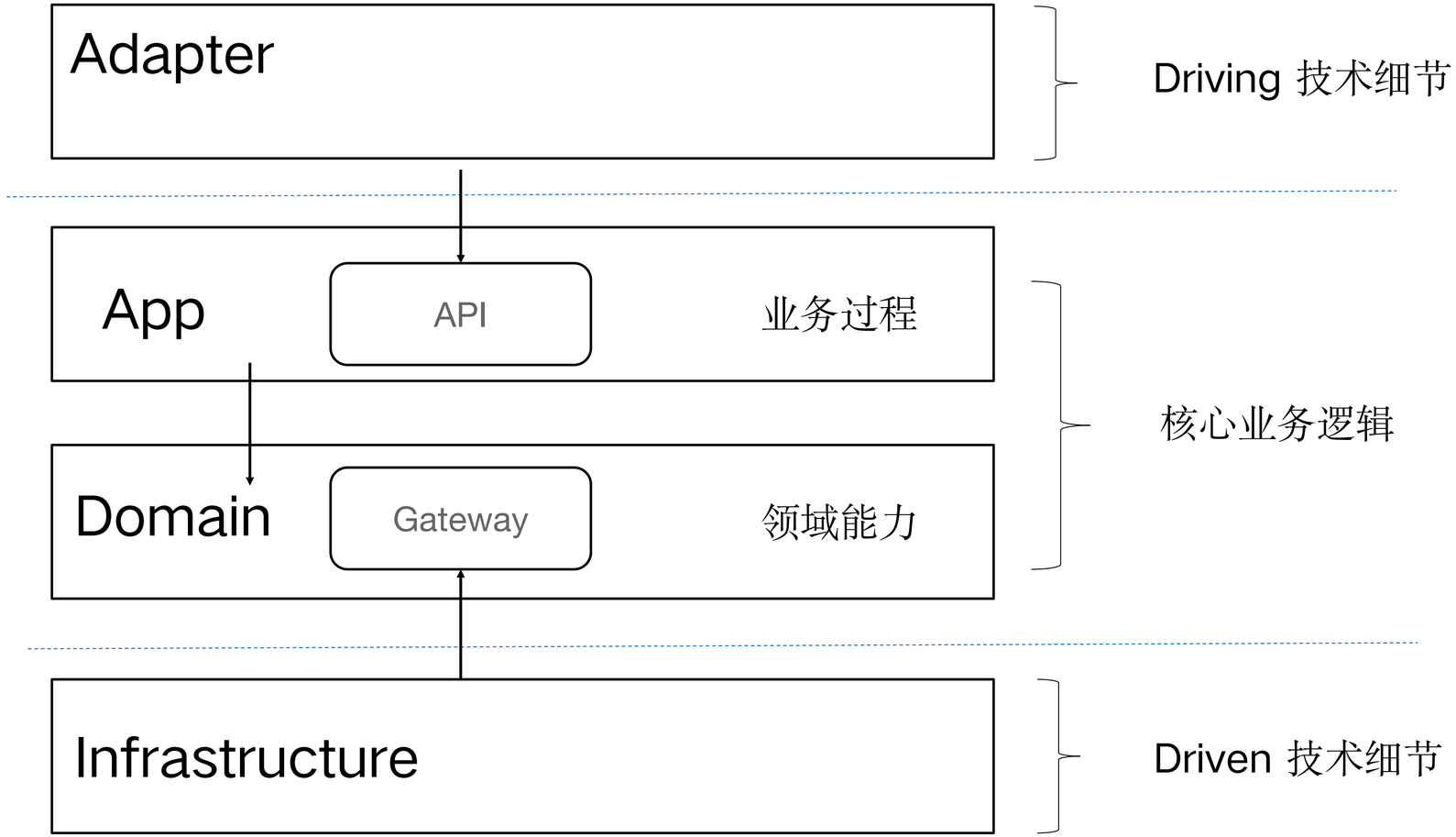


Domain层：



循序渐进沉淀领域能力：1、内聚性；2、复用性；3、可理解性（业务语义表达）

COLA实践：方法论总结——让上帝的归上帝，凯撒的归凯撒

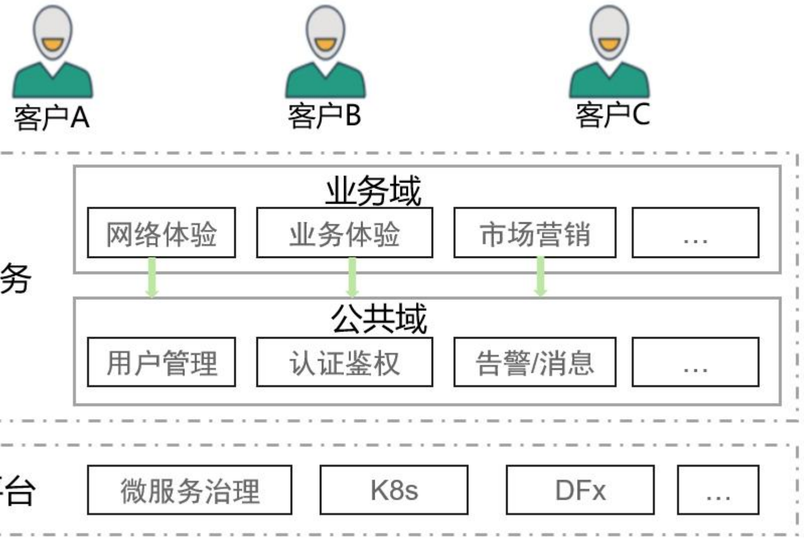


在架构的指引下，实现“核心业务逻辑”和“技术细节”分离

大纲

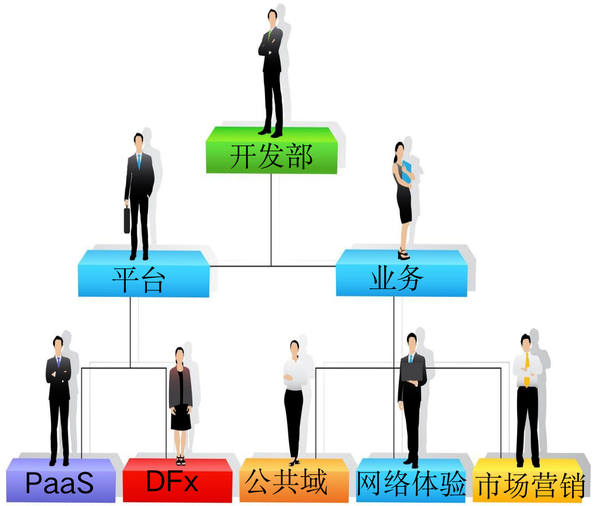
- 从DDD思想到应用架构
- COLA应用框架
- **华为GTS的落地实践**
- 思考与总结

华为GTS某产品



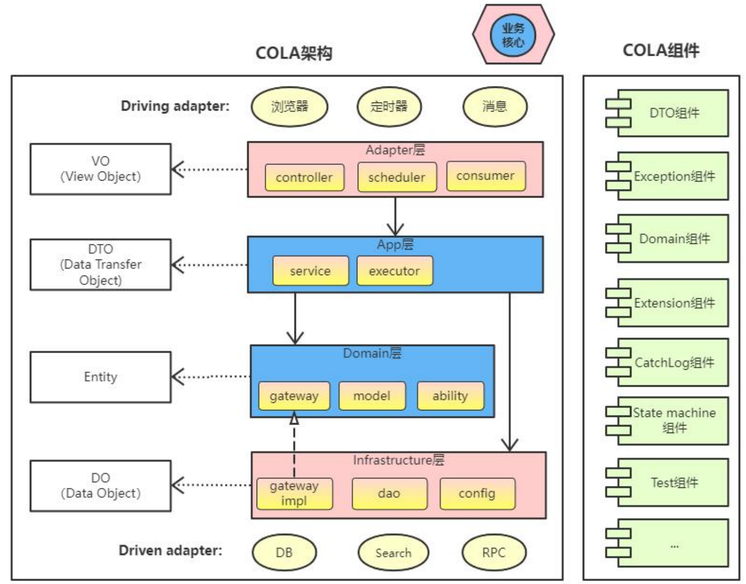
要素是用户、场景、功能

产品架构



要素是员工

组织架构



要素是代码功能单元

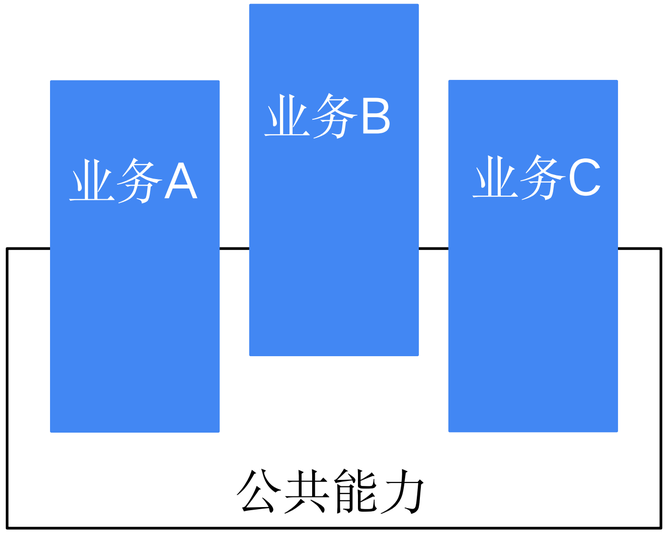
应用架构

组织匹配

微服务

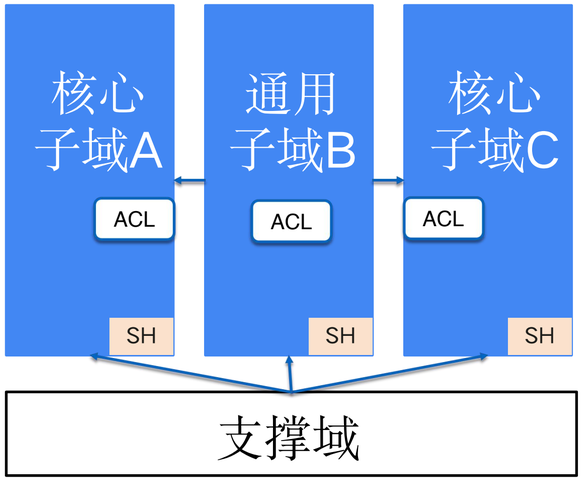
产品架构的问题洞察与应对措施

产品架构导致协作、效率、治理问题频发



1. **业务和公共能力耦合：**公共能力深入介入业务，导致协作成本倍增。
2. **一致性成本：**部分公共能力在多个业务中均有实现，功能重复，这就给稳定性带来了极大隐患。
3. **平台认知成本：**平台能力复杂，对大部分人来说，学习成本、问题排查成本高。

解决措施 = 解耦 + 抽取公共领域



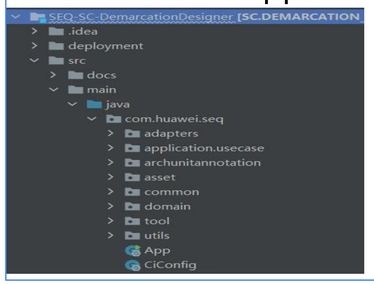
1. **识别核心子域：**划分界限上下文，识别网络域、业务域、营销域、位置域等核心子域，同时构筑自己的防腐层（ACL）。
2. **抽取通用子域：**将告警、定界等公共能力以ServiceAPI方式对外提供调用。
3. **抽取支撑子域：**将操作日志、权限、License等通过共享内核（Shared Kernel）以SDK的方式提供使用。

应用架构的问题洞察与应对措施

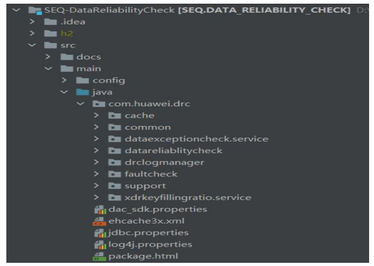
问题洞察

1. 部分应用架构不统一，维护困难

Demarcation App

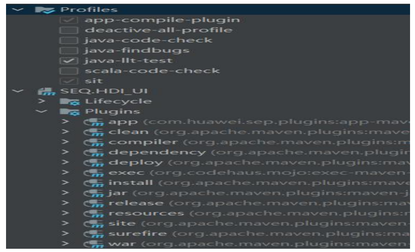


DRC App

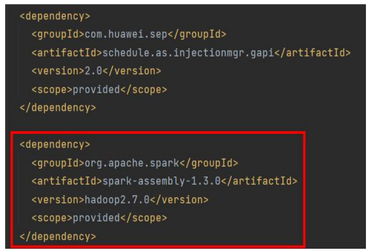


2. 部分前后端部署耦合，前端编译部署成本高

前端、后端、平台部署耦合

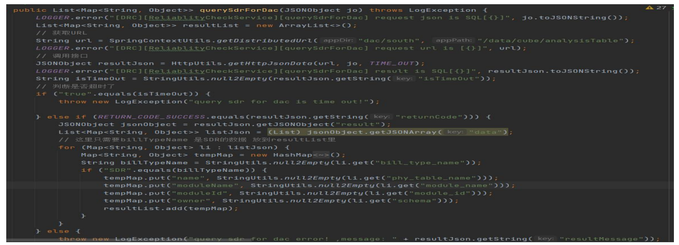


依赖混乱



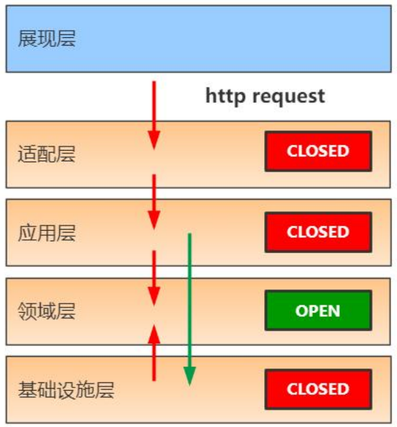
3. 内部API不统一，可测性差

内部接口 手工API解析，缺少API规范指导

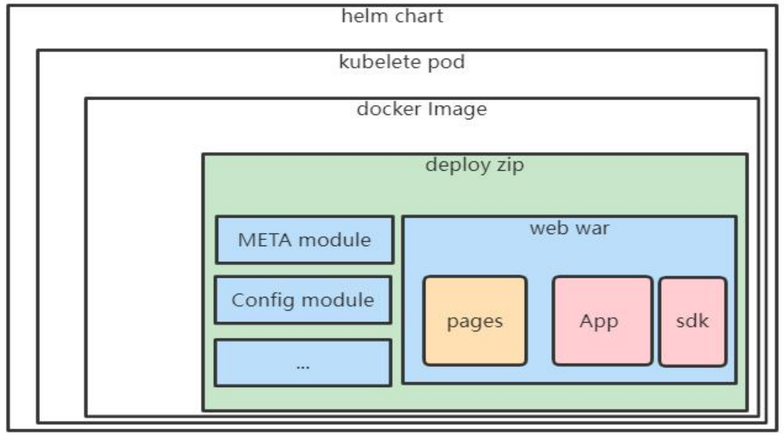


应对措施

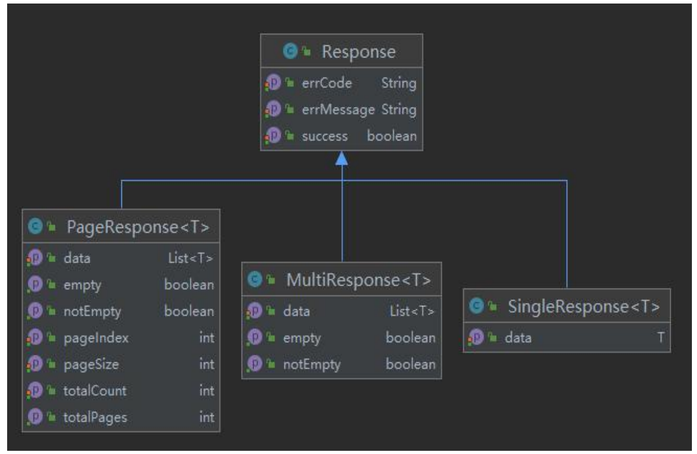
1. 统一应用架构



2. 统一部署架构



3.统一API规范



基于契约和规范、易维护、易理解、易测试、稳定可靠的应用整洁架构

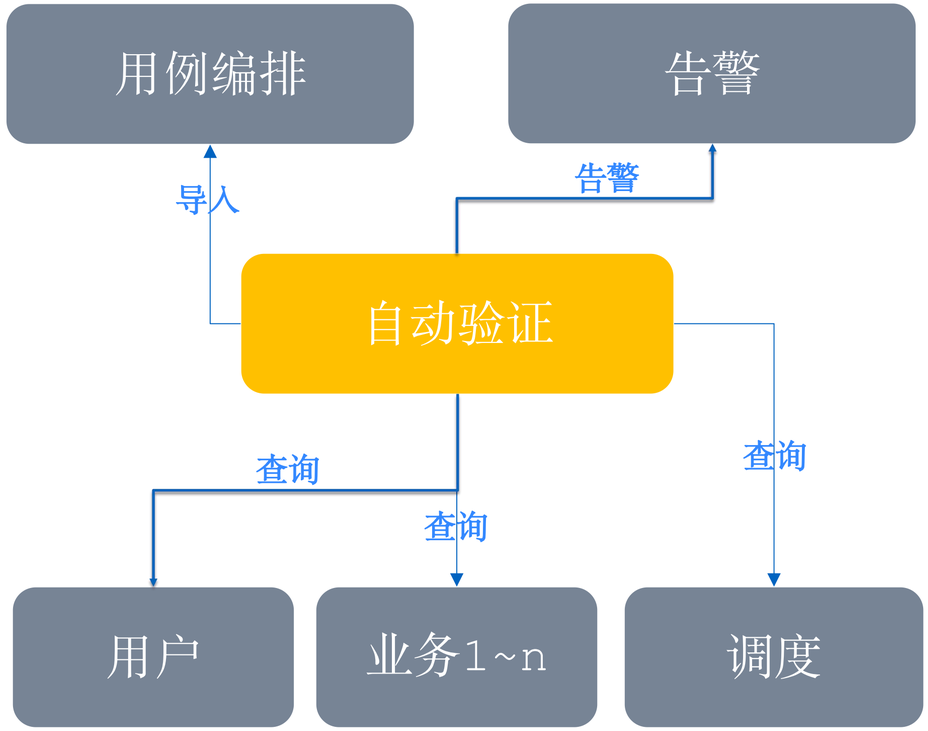
案例介绍—自动验证服务

需求背景：

由于我们的产品已经进入客户生产流，所以客户对系统和数据稳定性的要求很高，每次在升级后均需要人工进行大量用例验证；在每年上百个客户，多次版本升级的场景下，亟需产品提供一个服务，将客户关注的系统状态、数据状态按照既定的规则进行自动验证，确保版本升级后，数据流量稳定、流&批处理任务数据指标无异常、业务功能无异常；

需求简要分析：

- 用户可以根据接口定义生成**请求参数**，里面的字段支持按**规则**在运行期回填；
- 用户可以选择一个**接口**，定制响应数据的校验规则，如一致性检查、波动性检查等规则；
- 用户可以选择**多个接口**进行关联，后面接口的参数可以按规则读取前面接口返回的结果，同时支持结果数据的连续性检查、关联度检查等规则；
- 用户可以导入编排好的用例资产，并可以对导入的**资产**和**用例**进行管理；
- 用户可以从界面上创建**任务**，选择用例集进行执行，并对创建的任务进行管理；



自动验证服务—达成共识，输出统一语言

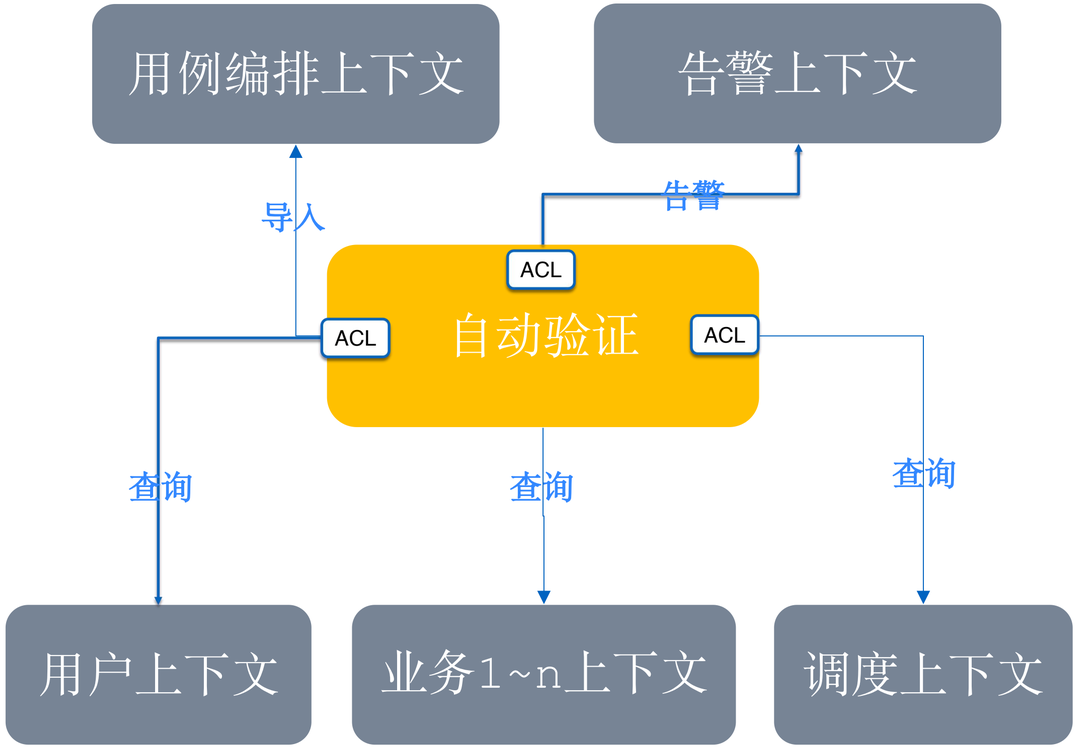
英文	中文	解释
Asset	资产	编排的测试用例资产包
Case	用例	场景化的测试用例，可能包含多个关联的接口；
Task	任务	测试用例的运行态管理实体
Interface	数据接口	API接口的信息，包含请求参数、接口类型
Request	请求参数	API接口请求参数，
ParamRule	参数规则	描述请求模板中需要替换的字段的规则；
Client	请求客户端	执行接口请求的对象，需要支持HTTP、WS等多种方式
Response	相应对象	API接口返回的结果
Check	结果检查	测试用例返回结果检查
CheckRule	结果检查规则	结果检查的规则；含准确性、波动性、连续性等；

使用

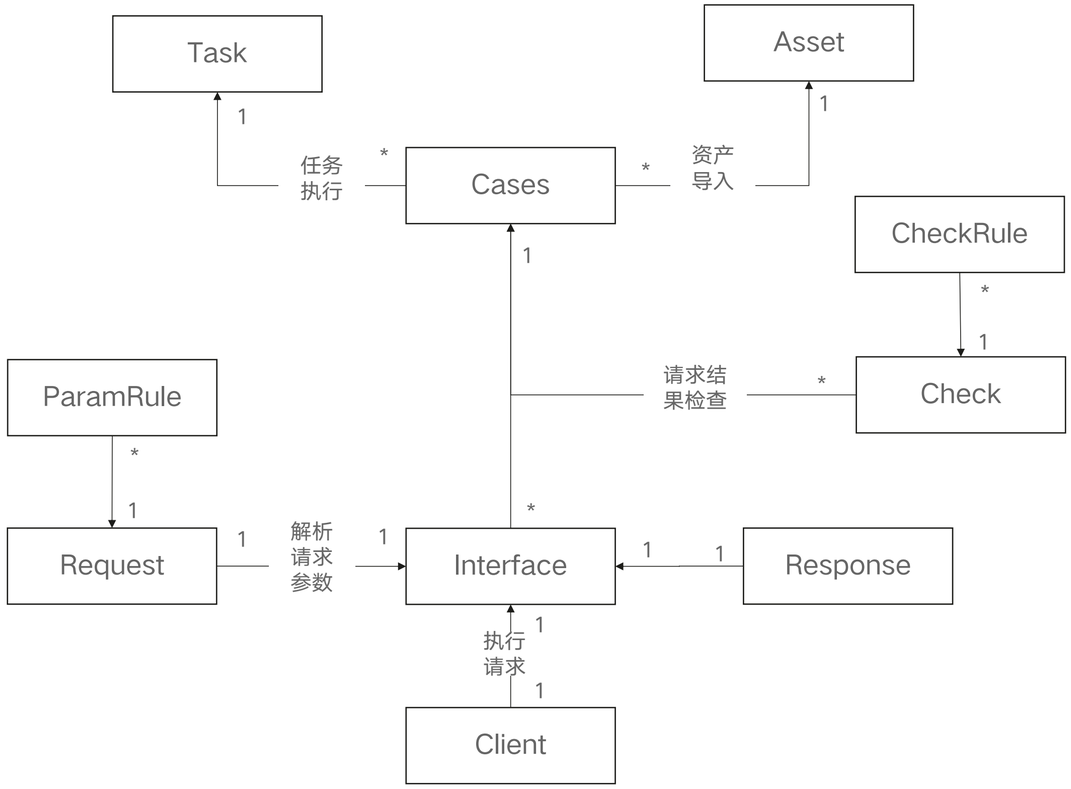
业务概念　需求文档　日常沟通　设计文档　代码

自动验证服务—领域模型设计

上下文映射，通过防腐层

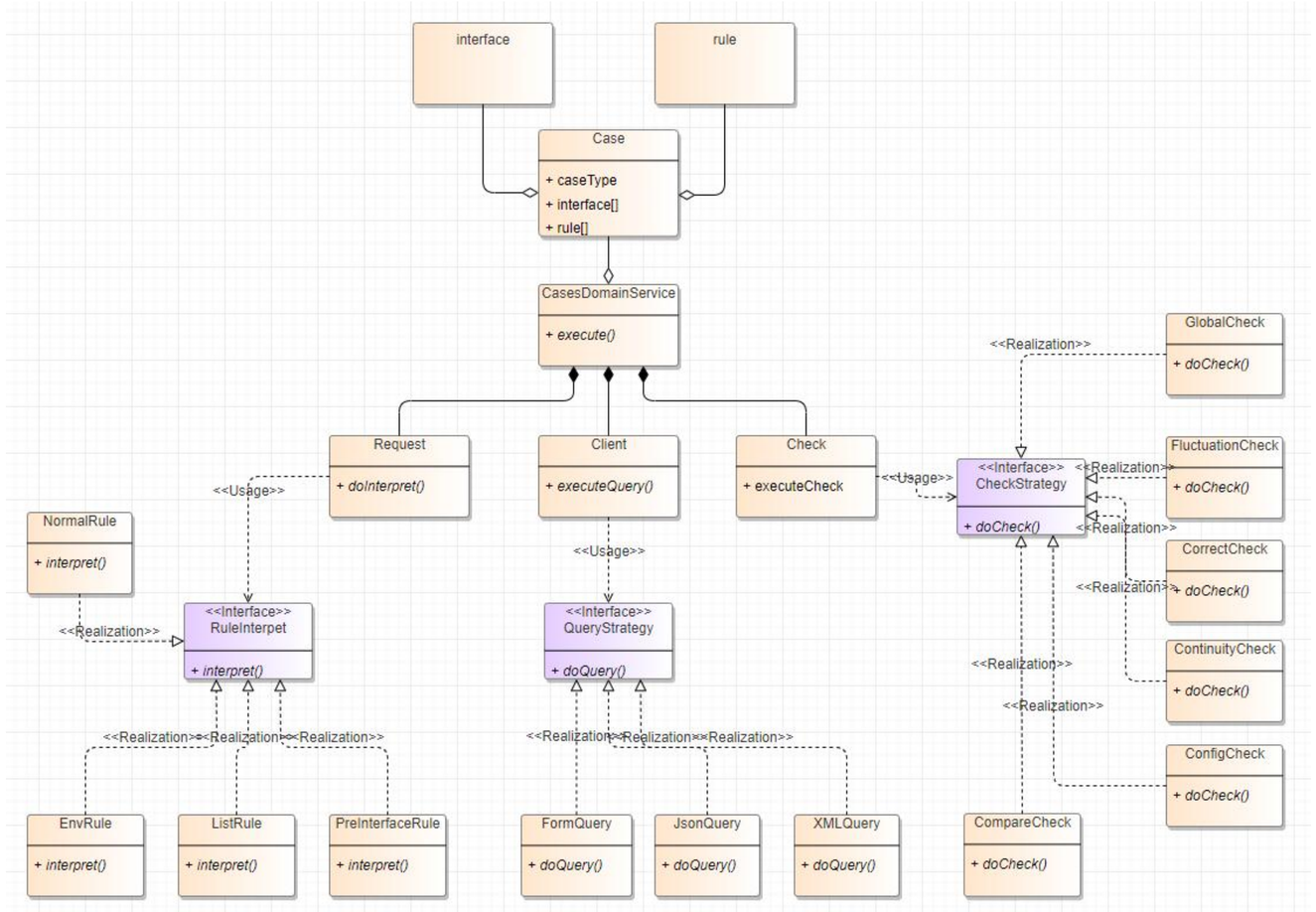


各个实体的依赖关系定义



自动验证服务--领域模型详细设计

Case聚合中请求参数、请求客户端、检查规则的设计是抽象的，方便扩展更多场景。

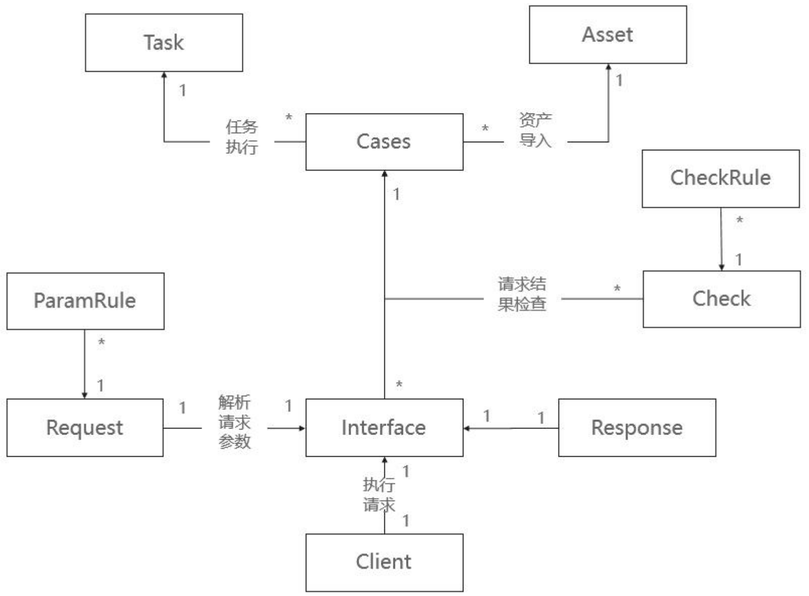


自动验证服务--文档、设计、代码实现一致

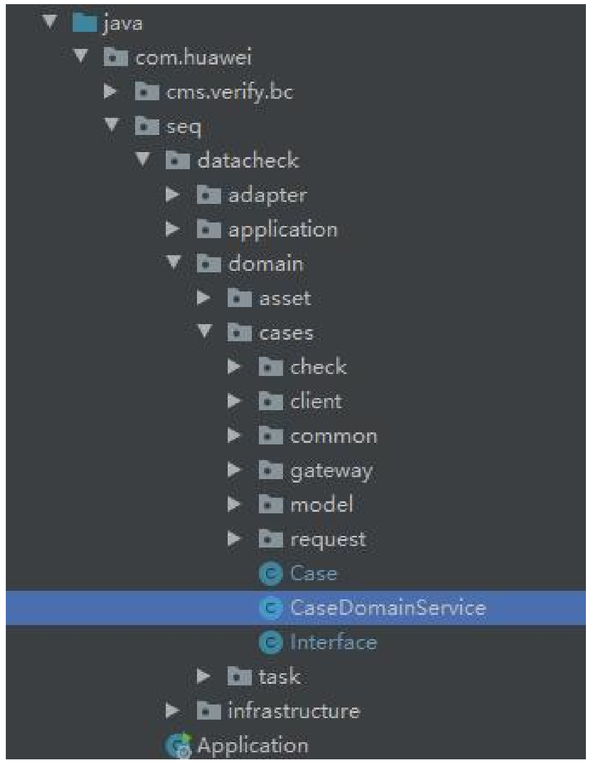
文档

英文	中文
Asset	资产
Case	用例
Task	任务
Interface	数据接口
Request	请求参数
ParamRule	参数规则
Client	请求客户端
Response	相应对象
Check	结果检查
CheckRule	结果检查规则

设计



代码



统一语言

自动验证服务—架构守护

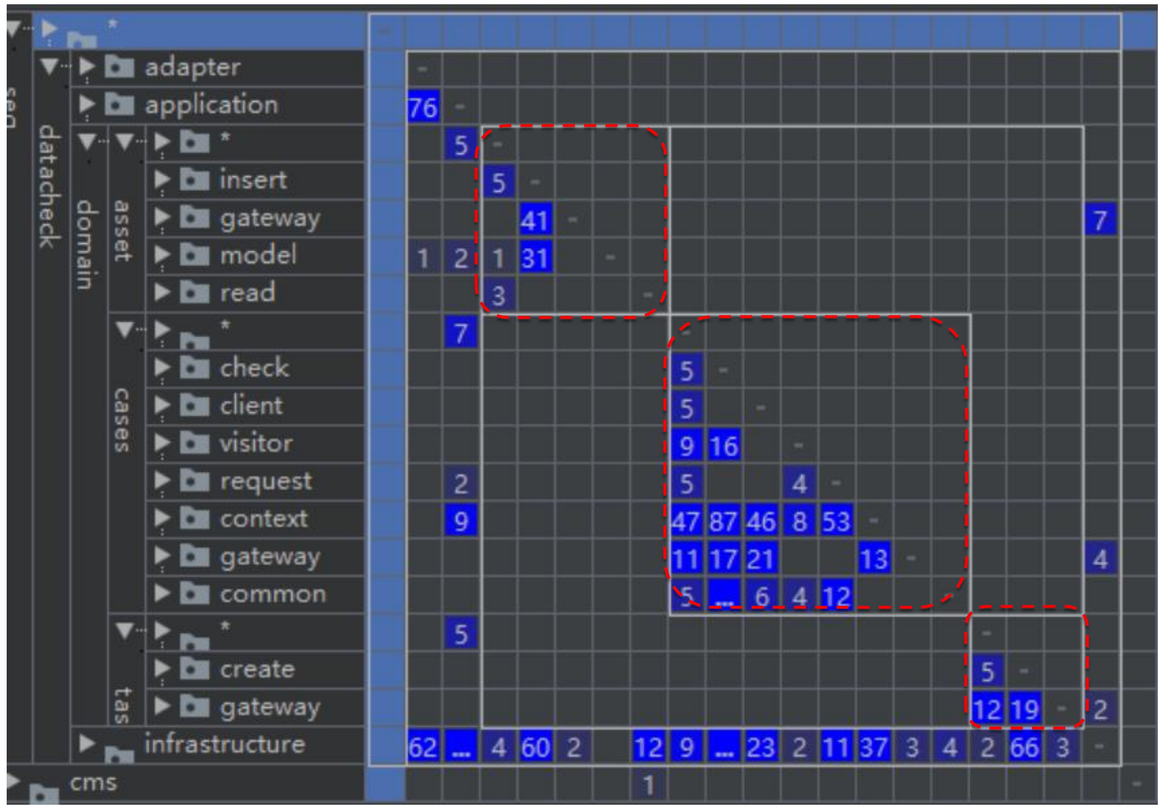
ArchUnit架构分层、依赖看护

```
public class CleanArchTest {

    @Test
    public void protect_clean_arch() {
        JavaClasses classes = new ClassFileImporter()
                .withImportOption(ImportOption.Predefined.DO_NOT_INCLUDE_TESTS)
                .importPackages("com.huawei.xxx");

        layeredArchitecture()
                .layer("adapter").definedBy("com.huawei. xxx.adapter")
                .layer("application").definedBy("com.huawei. xxx.application")
                .layer("domain").definedBy("com.huawei. xxx.domain")
                .layer("infrastructure").definedBy("com.huawei. xxx.infrastructure")
                .whereLayer("adapter").mayNotBeAccessedByAnyLayer()
                .whereLayer(" application ").mayOnlyBeAccessedByLayers(" adapter ")
                .whereLayer("domain").mayOnlyBeAccessedByLayers("application",
"infrastructure")
                .as("The layer dependencies must be respected")
                .because("we must follow the Clean Architecture principle")
                .check(classes);
    }
}
```

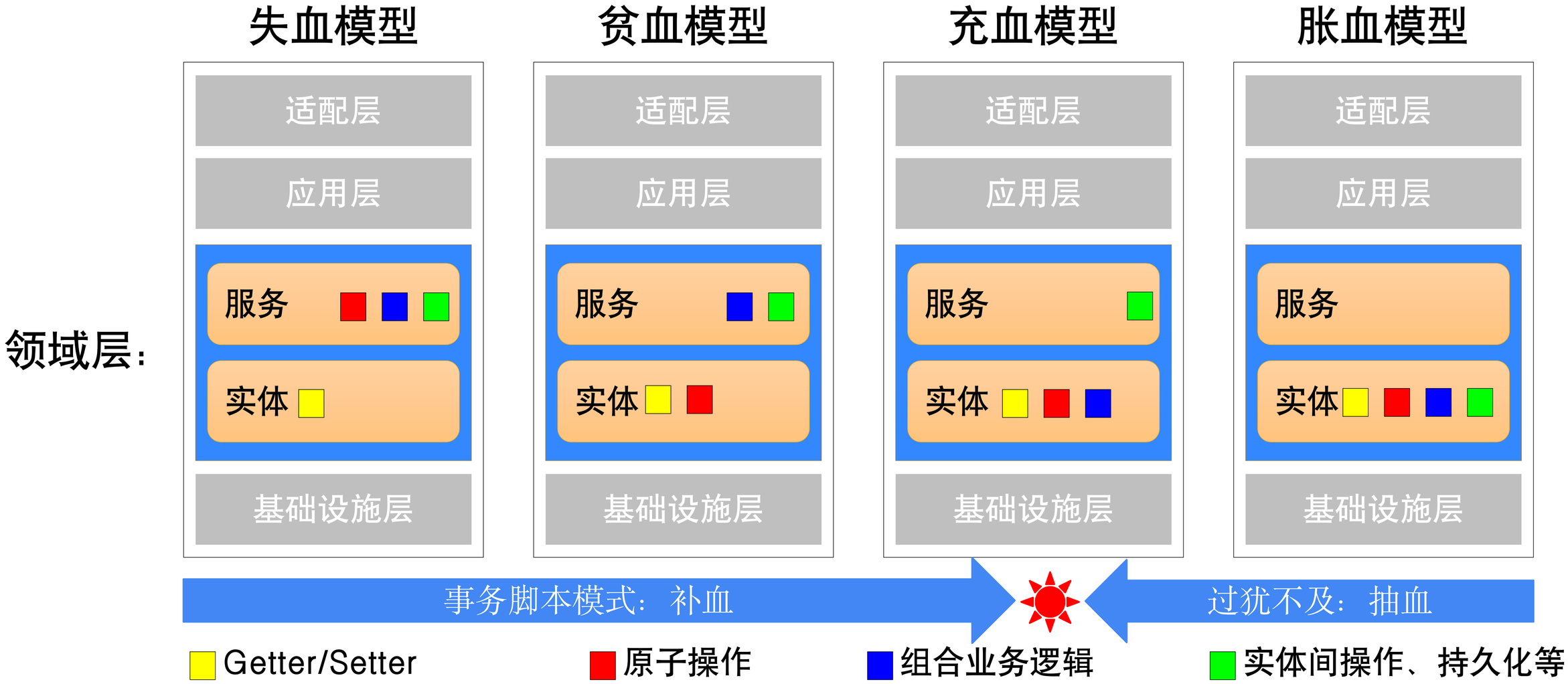
干净的层次依赖、领域模型之间无循环依赖



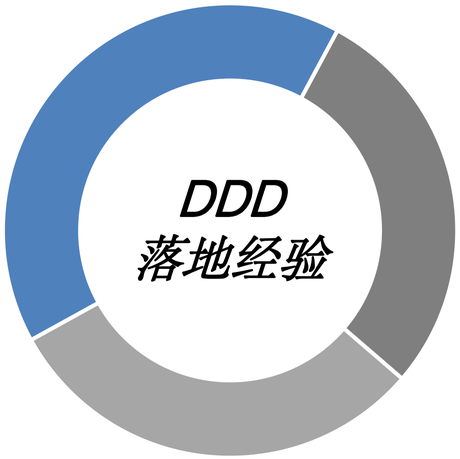
大纲

- 从DDD思想到应用架构
- COLA应用框架
- 华为GTS的落地实践
- **思考与总结**

领域模型不是一蹴而就，需要在迭代中持续完善



组织重视+流程文化氛围牵引是DDD成功落地的保障



组织认可

- ✓ 跟团队领导及各领域角色一起识别落地的成本和长远的收益。
- ✓ 对齐DDD落地愿景，用愿景作为落地的最大内驱力。

技术债务例行管理

- ✓ 团队技术决策组织，要认识到没有最优设计，只有最合适设计，保持简单设计。
- ✓ 例行识别技术债务，纳入管理，及时落地，防止利息膨胀；

流程文化牵引

- ✓ 软件不仅是技术，还是工程，从管理和流程角度也需要适配；
- ✓ 领域模型设计，作为开发的前置条件，防止方向错误，越走越远。

One More Thing：实践总结，达成持续DDD落地的4条军规

- ✓ **架构规范**：团队核心成员一起输出产品架构、应用架构**规范**，软件设计必须**遵守**规范；
- ✓ **设计规范**：模块第首次开发，必须**输出**软件设计（统一语言、领域模型、模型详细设计）；如果模块有变化，须同步**刷新**软件设计；同时设计中需包含**测试用例**设计，从业务视角保证领域对象的正确性；
- ✓ **文档即代码**：软件设计以MD格式承载并提交到**代码仓**中，包名、类名跟领域模型代码实现**保持一致**；
- ✓ **持续演进**：没有适用于所有场景的架构，**持续重构**，以代码持续CleanCode、架构持续CleanArchitecture为目标；

《数字人才发展体系：粮仓模型白皮书》

—— 推动数字人才全面发展

扫描左侧二维码
免费下载白皮书

极客时间 | 企业版

想一想，我该如何把这些技术应用在工作实践中？

THANKS