

基于角色的软件系统架构设计

周学文 涂宏魁

国防科技大学计算机学院 四川 成都 610036

摘 要：本文深入探讨了基于角色的软件系统架构设计方法，强调了在软件系统全生命周期中，不同利益相关方（如业务方、设计团队、开发团队、测试团队和运维团队）对架构视图的需求和关注点。文章通过定义不同的架构视图（业务视图、设计视图、开发视图、测试视图和运维视图），阐述了如何系统化地识别关键角色，并通过架构视图体系来指导软件架构设计，确保架构设计的有效性和完整性。文章还讨论了架构视图之间的映射关系，以及如何通过这些视图来指导软件系统的构建、测试、部署和运维。

关键词：软件系统架构；架构视图；系统角色

“视图是一组相关关注点的表达。架构视图可以用模型表达，以展现利益攸关者在架构中所关注的领域。”通过系统化识别参与系统的关键角色，进而定义符合系统特征的架构视图体系，方能把握架构设计的本质要义^[1]。本文将识别系统的利益攸关者，将其定义为参与架构活动的不同的角色，并以这些角色为服务对象定义架构视图，对应业务人员、设计人员、开发人员、测试人员和运维人员，将软件系统的架构分解为业务视图、设计视图、开发视图、测试视图、运维视图，解析其内存逻辑和设计思路。

1 业务视图

在软件系统构建过程中，业务人员不应在提出需求后即脱离项目进程。软件工程区别于传统工程建设的显著特征在于其需求的动态演化特性。对需求理解的过程，就是一个知识消化的过程，Eric Evans认为：“知识消化并非一项鼓励的活动，它一般是在开发人员的领导下，由开发人员与领域专家组成的团队来共同协作。”因此，领域驱动设计方法论尤其强调领域专家与开发团队协同工作，这一原则源自业界实践经验总结，为准确呈现业务人员的架构关注点，将该视图定义为“业务视图”^[2]。该视图不仅是业务人员期望的成果展现形式，更应作为业务人员与开发团队协作参与、持续演进的架构模型。视图内容涵盖业务功能的逻辑分区及核心领域概念模型，如图1所示：

业务视图的基本组成单元使用了UML的模块构造型，它“可用于构建更复杂结构的一组标准化零件或独立单元中的一个”。模块内部，使用圆角矩形框表示核心的领域概念。业务视图并不能决定在设计时一定采用面向对象设计，即使采用了面向对象设计，领域概念的内容也未必恰好和一个类的定义刚好吻合。

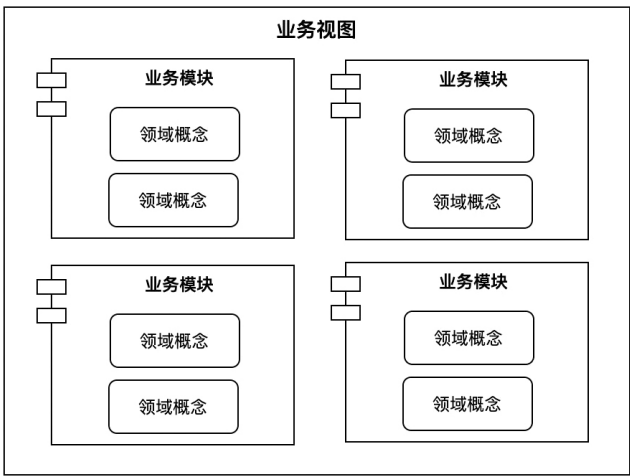


图1 业务视图

2 设计视图

如同接口设计需基于调用者视角进行构思，架构视图的定义同样需要立足于目标角色的核心诉求，致力于构建对设计人员具备高度亲和力的架构表达体系^[3]。鉴于该架构视图专为设计人员量身打造，可将其明确定义为“设计视图”。当设计人员审视架构视图时，他们期望从中获取哪些关键价值？参考Simon Brown提出C4模型中的系统上下文图，通过系统化梳理，其内容框架应完整包含以下维度：

- ◎目标系统与外部环境的关系
- ◎目标系统被分解为哪些组成元素
- ◎组成元素之间的调用关系

除此之外，系统上下文图可以直观呈现目标系统与外部环境的关系，同时，它也有助于明确目标系统的边界。如图2所示：

图很简单，要表达的含义却很多，包括：

- ◎明确目标系统的边界，即确定解空间的边界

◎明确哪些角色与目标系统有关

◎明确目标系统与外部系统之间的调用关系，包括调用协议，依赖方向

◎明确目标系统与外部系统所属团队之间的协作关

定义企业级系统上下文模型是构建IT生态系统框架的核心，涉及系统集成与交互分析。系统逻辑架构由三个关键要素构成：

◎层

◎功能组件

◎资源库

视图中的层是对目标系统的关注点分离，分离维度可以：

◎从调用者到被调用者：调用者位于顶部，代表如GUI这样的客户端；被调用者就是系统的访问资源，如数据库或其他中间件，通常位于底部。这一划分维度偏向于技术维度。

◎从通用到专有：位于下方的层次越通用，位于上方的层次越专有。这一划分维度偏向于业务维度。

◎从核心到非核心：越是核心的内容，越不容易变化，因而位于逻辑架构的内部；围绕在核心之外的，是一些发散的非核心功能，对应不同的业务关注点。这是一个内外分层架构。

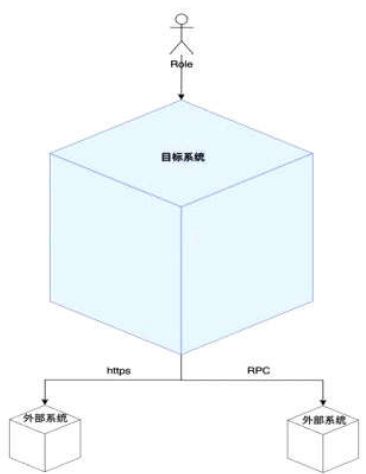


图2 设计

3 开发视图

如图3所示：为了保证代码模型的一致性，应为每一层的组件（或库）定义通用的代码结构，如此就能“向团队成员传递统一的设计元语，潜在地给出了架构的设计原则与指导思想”。倘若能做到同一层中每个组件的代码结构都从一个模子里铸出来，代码的阅读者就能做到窥一斑而知全豹，减少许多理解的成本。虽说如此会抹杀“个性”，但这就是软件工程技艺的一面，更不用

说“一致”带来的美感，其实也是一种艺术。通用的代码模型通常只能满足一部分场景^[4]。场景不同，决策也会不同。这就需要在开发视图中明确架构原则，从而给开发人员留出可以灵活调整的空间。

举例来说，假设项目引入了CQRS（Command Query Responsibility Speration）模式，针对不同的命令场景与查询场景，必然会定义不同的代码结构。开发人员需要理解做出这一区分的根由是什么，由此才能举一反三，在面对相似场景时，定义出合理的代码模型。

定义代码结构时，需充分考虑变化带来的影响，依据变化维度而非功能分类划分命名空间。以一个数据平台项目为例，它需要服务于全国的各个机场，并在机场的数据中心部署对应的Job。每个Job都由一到多个source、processor和sink组成。

有一些架构原则可能非常具体，例如处理异常的原则。可以规定：何时定义受控异常，何时定义非受控异常，异常在不同层次采用什么样的处理方式，捕获到不同类型的异常后该如何处理……这些内容都属于开发视图。

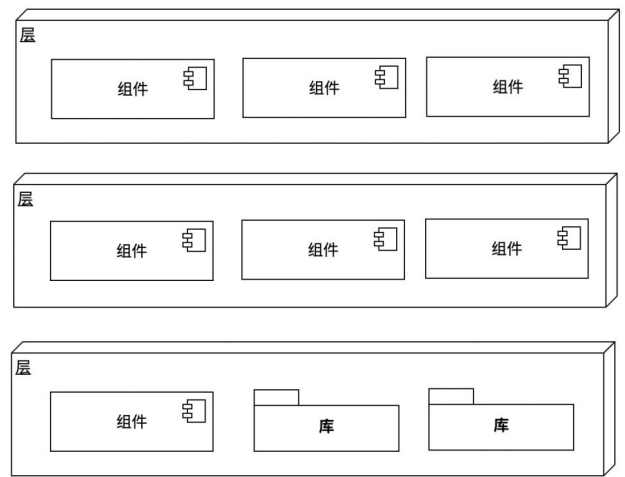


图3 开发视图

4 测试视图

测试人员和软件系统的架构有关吗？——当然有关！当测试人员在制订测试策略时，需要根据实际的架构决策确定该采用何种测试类型以及测试工具，不同类型测试的关系是什么，它们各自覆盖哪些因素，以保障和提升软件质量。

影响测试策略的决策主要来自设计视图的质量视图^[5]。以性能视图为例，某医疗系统需要针对医疗术语执行查询和替换操作。查询操作根据关键字进行查询，要求10s以内返回查询结果；替换操作的逻辑非常复杂，需要保证替换业务逻辑的正确性，遵循药品管理规则，执行时间较长，但不需要实时完成。为此，设计视图将查

找和替换划分为online和offline两个不同的场景，各自采用同步和异步机制。因此，就需要针对不同场景制订不同的性能测试策略。

如果我们将测试策略方案视为架构视图的一部分，则该视图可以称之为“测试视图”。

5 运维视图

为运维人员提供的架构视图，可以称之为”运维视图”。

运维人员可以分为两种角色：基础运维人员和应用运维人员。基础运维人员负责整个企业IT系统的运维工

作，提供企业范围内通用和基础的运维能力；应用运维人员则为具体某个应用（或独立的软件系统）提供运维服务。软件系统的运维视图主要针对后者。运维视图包括了目标系统的物理架构（或部署视图），这就需要明确目标系统各个组件（或库）的部署方式，各自需要的计算资源、网络资源和存储资源。定义这些资源时，还需要明确目标系统所要部署的底层基础设施究竟是物理机、虚拟机还是容器，如果需要上云，则需要确定是公有云、私有云还是混合云。这些决策都会影响到运维视图的内容。如下图4所示：

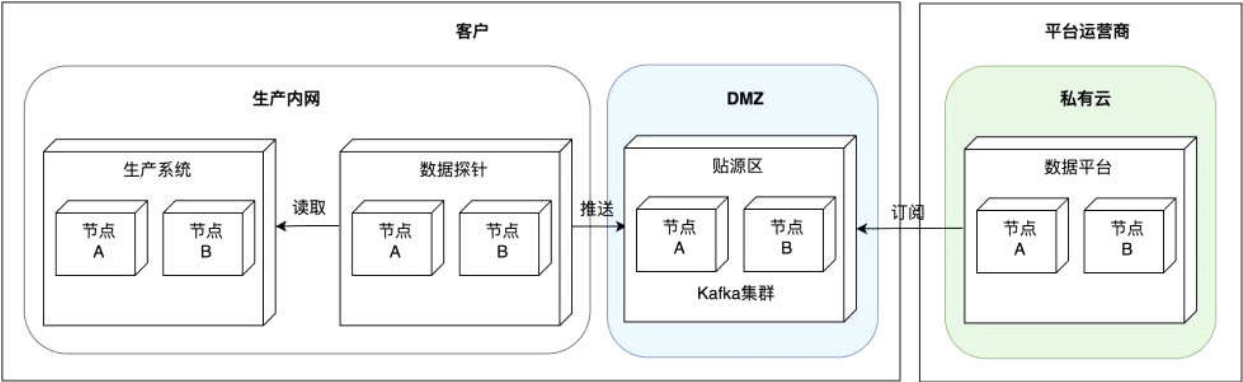


图4 运维视图

运维视图不只限于软件部署的物理架构，还需要考虑目标系统在运行过程中，对它的监控和追踪，并提供运行状态的分析和报告，以便于及时发现问题。“随着微服务架构的持续演进，应用和服务数量不断增加，调用关系越来越复杂。所以，从运维的角度来看，首要任务便是保持可观测性”。可观测性是云原生技术的重要组成部分。基于可观测性的要求，需要考虑引入支持日志功能的ELK、支持指标的Prometheus，以及支持链路追踪的ZipKin等可观测性工具，它们的部署方案也属于运维视图。

运维还牵涉到构建和交付的过程，属于CI/CD的内容。运维视图可以体现CI/CD流水线的设计，明确源代码仓库、持续集成服务器、测试环境、类生产环境（Staging）与生产环境之间的关系^[6]。针对分布式系统，还需要明确各个分布式部署的服务之间的调用关系，分支版本的管理等。

结语

架构视图为何要根据不同的角色划分不同的视图？因为架构视图是组成架构文档的主要内容，在设计架构视图时，若能考虑到不同角色需要关注的内容，就能写出真正有指导意义的架构文档。架构文档不是PPT，架构师也不是PPT架构师，那些形式好看、内容完美的架构图

未必真正能满足各个角色对目标系统的关注。

本文列出的架构视图仅仅给出了通用的架构视图内容，针对不同类型的软件系统，绘制出来的架构视图必然存在差异。例如，嵌入式系统的架构视图，以数据为核心的数据平台的架构视图，一定有别于大规模业务系统的架构视图。

参考文献

[1]陈晓阳.网约车上报监管平台系统架构演变研究[J].交通工程,2022,22(02):49-53.
[2]韩强,刘爱国.通信管理机软件系统设计与实现[J].智能建筑与智慧城市,2021,(05):79-81.
[3]林国富.论软件系统复合架构设计与应用[J].信息记录材料,2021,22(01):57-58.
[4]王俊.基于大数据的智慧物流软件系统架构设计及应用研究[J].中国战略新兴产业,2025,(20):44-46.
[5]肖逸,李俊业,方必武,等.基于微服务的大电网在线安全稳定分析系统数据中台与架构设计[J].中国电机工程学报,2022,42(S1):126-135.
[6]牟杨,栾江,许瑞平.AI时代下高校融合门户页面操作与语言交互并行的混合软件架构设计[J].科技创新与应用,2024,14(35):130-133.