软件系统设计

pp

---

1. 软件模式是将模式的一般概念应用于软件开发领域，即软件开发的总体指导思路或参照样板，是对软件开发这一特定“问题”的“解法”的某种统一表示

2. 软件设计原则：单一职责原则、开闭原则、里氏代换原则、依赖倒转原则、接口隔离原则、合成复用原则、最小知识原则（迪米特法则）

3. 设计模式按目的分为创建型模式用于创建对象、结构型模式用于处理类或对象的组合、行为型模式用于描述类或对象怎样交互和怎样分配职责。按范围分为类模式和对象模式。类模式处理类和子类之间的关系，这些关系通过继承建立，在编译时就被确定下来，是属于静态的；对象模式处理对象间的关系，这些关系在运行时刻变化，更具动态性

4. 创建型：工厂方法（类）、抽象工厂、建造者、原型、单例；结构型：类适配器（类）、对象适配器、桥接、组合、装饰、外观、享元、代理；行为型：模板方法（类）、策略、状态、命令、中介者、观察者、迭代器

5. 对象适配器可以适配多个类，能适配一个类及其子类，不能置换适配者的方法；类适配器只能适配一个类，可以置换适配者的方法，目标抽象类只能是抽象类

6. 代理模式：远程代理、虚拟代理、Copy-On-Write 代理、保护代理、缓冲代理、防火墙代理、智能引用代理

7. 防御式编程是通过预见到问题所在，断定代码中每个阶段可能出现的错误，并作出相应的防范措施，来防止类似意外的发生。进攻式编程不是防御式编程的反义词，是主动暴露可能出现错误的态度

8. 用错误处理代码来处理预期会发生的非正常情况；用断言来检查永远不该发生的情况。隔栏隔开了断言和错误处理，在隔栏之前用错误处理，在隔栏之后用断言

9. 如果设计模式那么好，为什么没有人构建一个库，这样我们就不用写了？（设计模式比库层级更高，告诉我们如何构建类和对象来解决特定的问题；我们的工作是适配这些设计来适合我们特定的应用程序）

10. 库和框架是不是设计模式？（框架和库不是设计模式，他们提供了链接到我们代码中的特定实现，有时，库和框架在实现中会使用设计模式，一旦理解了设计模式，就可以更快地理解围绕设计模式构建的API）

11. 软件架构是程序或计算系统的结构，由软件元素、这些元素外部可见的属性、这些元素的关系组成；是一个系统的基本组织形式，包含于它的组件、组件之间和软件与环境之间的关系、支配它的设计和演进的原则

12. 设计的步骤

    1. 分解
    2. 抽象
    3. 分治
    4. 生成和测试
    5. 迭代、改进
    6. 可重用元素

13. 4+1 模型

    - 逻辑视图：描述架构中重要的元素及其之间的关系
    - 进程视图：描述架构的并发和通信元素
    - 物理视图：描述主要过程和元素是如何被映射到应用程序硬件
    - 开发视图：捕获软件组件的内部组织
    - 架构用例：捕获架构的需求；与多个特定视图关联

14. 架构师职责：联络、软件工程最佳实践、技术领域深入理解、与设计和技术相关联的风险管控

15. 架构活动：创造系统的商业案例、理解需求、创造和选择架构、与包括开发者在内的涉众沟通架构、分析或评估架构、实现架构、确保架构符合要求

16. 架构的角色：描述如何实现需求，最难变更、最需要谨慎考虑，提供一个维护和更新决策的参考框架

17. 架构作用：提供交流手段；早期设计决定的表现；支配了开发维护的精力的组织结构；促进或阻碍了质量属性的达成；影响质量；会考虑潜在的变更；是可转移和可复用的抽象；是产品共通点的基础

18. 质量需求是整个系统的期望特征，是在功能需求之上的。约束是 0 自由度的设计决定，是预定义的设计决定。非功能性需求和架构需求是质量属性两个相互替代的形式，必须在设计的时候就考虑

19. 质量属性需要在开发的所有阶段考虑，超越了系统功能，架构约束了不同质量属性的实现

20. 总体场景是系统独立的场景，用来指导质量属性需求的规范。具体场景是系统特定的场景，用来指导一个特定系统的质量属性需求的规范，是总体场景的实例

21. 场景建模

    - 刺激：当其到达系统时需要考虑的状况
    - 刺激源：一个产生了刺激的实体（如人、系统等）
    - 响应：刺激到达后进行的活动
    - 响应度量：应对刺激的响应应该是能用某种方式进行度量的，这样就能够测试需求
    - 环境：当刺激发生时，系统的条件（如过载、运行中等）
    - 制品：需求应用于的整个系统或系统的一部分

22. 样式或模式通过应用战术来提供承诺的益处，一个战术是一个影响质量属性响应控制的设计决定，战术的集合被称为架构策略

23. 架构是设计决定的集合，设计决定的七个分类包括职责分配、协调模型、数据模型、资源管理、架构元素之间的映射、绑定时间的决定、技术选择

24. 可用性战术：故障检测、故障恢复、故障预防

25. 互操作性战术：服务定位、管理接口

26. 可修改性战术：减小模块体积、增加内聚、减少耦合、延迟绑定

27. 性能战术：控制资源命令、管理资源

28. 安全战术：攻击检测、攻击抵抗、攻击回应、攻击恢复

29. 可测试性战术：控制和观察系统状态、限制复杂度

30. 易用性战术：支持用户主动性、支持系统主动性

31. 架构重要需求是一种将在架构上有深刻影响的需求，如果没有这种需求，架构可能会有很大的不同。质量属性需求越困难、越重要，就越有可能对架构产生重大影响，因此成为 ASR

32. 识别 ASR 的方法：从需求文档收集、采访涉众、理解业务目标、从质量属性效用树中捕获

33. 架构模式是一组架构设计决定，这些决定适用于重复出现的设计问题，并被参数化以考虑出现该问题的不同软件开发上下文

34. 架构模式有分层模式、代理人模式、模型-视图-控制器模式、管道-过滤器模式、客户机-服务器模式、点对点模式、面向服务模式、发布-订阅模式、共享数据模式、映射-规约模式、多层模式

35. 模式与战术的关系：战术比模式简单，它们使用单一的结构或机制来处理单一的架构要求；模式把多个设计决定组合在一起；都是架构师的主要工具；战术是模式设计和创造的基石；大多数模式由几个不同的战术组成

36. 架构设计策略：分解、设计为 ASR 、产生和测试

37. 属性驱动设计的输入是需求，输出是软件元素、角色、职责、属性、关系

38. 质量属性驱动设计的步骤

    1. 确保有充足的需求信息
    2. 选择的一个系统元素进行分解
    3. 为选中元素识别 ASR
    4. 选择一个符合 ASR 的设计理念
        1. 识别设计关注点
        2. 为从属关注点列出可选的模式、战术
        3. 从列表中选择模式、战术
        4. 决定模式、战术与 ASR 之间的关系
        5. 捕捉初步架构视图
        6. 评估和解决不一致
    5. 实例化架构元素并分配职责
    6. 为实例化的元素定义接口
    7. 验证和细化需求，并使它们成为实例化元素的约束
    8. 重复直到所有 ASR 都被满足

39. 架构文档的作用：沟通和社会化架构设计决定，帮助理解和评估架构设计决定，刷新设计师对确定的决定的认识，训练人们设计架构，支持在地理上分布式的团队，用以架构设计分析、工作分解和分配、开发后维护，提供了一个用来维护和更新决定的框架

40. 样式的分类

    1. 它如何被构造成一组样式单元？模块样式
    2. 它如何被构造成一组有运行时行为和交互的元素？组件-连接器样式
    3. 它如何与它环境中的非软件结构关联？分配样式

41. 一个架构样式是“元素和关系类型的专门化，以及如何使用它们的一组约束”；一个架构模式是“软件系统中的基础结构组织方法的表达”。架构模式的一个重要部分是，它关注于问题和上下文，以及如何在这个

上下文中解决问题；架构样式侧重于架构方法，对于特定样式何时、是否有用这个问题有着更轻量级的指导。架构模式：「问题，上下文」➡架构方法；架构样式：架构方法

42. 架构视图是一组系统元素及其关系的表示，这些元素不一定是全部系统元素，而是特定类型的元素。视图让我们将系统实体划分成感兴趣和易于管理的系统表示。不同的视图支持不同的目标和用户，并凸显出不同系统元素和关系。不同视图在不同程度上展现不同的质量属性

43. 模块是提供一套连贯的职责的实现单元，模块视图包括分解视图、使用视图、概括视图、分层视图、方面视图、数据模型视图。组件-连接器视图显示一些运行时存在的元素，如进程、对象等组件，组件-连接器视图包括管道-过滤器视图、客户机-服务端视图、点对点视图、面向服务架构视图、发布-订阅视图、共享数据视图、多层视图。分配视图描述了软件单元到软件开发或执行环境元素的映射，分配视图包括部署视图、安装视图、工作安排视图、其他分配视图、开发视图、物理视图

44. 质量视图包括安全视图、性能视图、可靠性视图、通信视图、异常视图、错误处理视图

45. 选择视图的步骤：构建涉众-视图表；确定上图中的边缘视图，通过将一个视图的元素与另一个视图中的元素相关联，将每个边缘视图与另一个更具支持性的视图相结合；确定优先级和阶段

46. 软件架构文档包括

    1. 文档路线图：描述文档中有哪些信息以及在哪里可以找到这些信息
    2. 视图如何被文档化：描述视图的文档结构
        1. 主要展示：显示视图的元素及其关系，通常图形化
        2. 元素目录：详细描述元素及其属性、关系及其属性、元素接口和行为
        3. 上下文图：展示系统和其部分如何与环境关联
        4. 可变性指南：描述该视图如何应对未来架构的任何变化点
        5. 缘由：为什么这个视图反映了设计，提供一个令人信服的论据以说明它是 健全的
    3. 系统概览：概要地描述系统
    4. 在视图间映射：描述每种视图的相似和映射
    5. 缘由：描述最终选择的视图的原因
    6. 目录：索引、词汇表、缩略词表

47. 风险、敏感点、权衡点

    1. 风险：可能对所需质量属性产生负面影响的架构决定；增加备份数据库导致性能下降；用户的简单密码是安全性的风险
    2. 敏感点：对于特定质量属性敏感的架构决定；可靠性对于增加备份数据库敏感用户的简单认证降低安全性
    3. 权衡点：影响多个质量属性的架构决定；增加备份数据库让可靠性提升，让性能下降

48. ATAM 过程每个阶段的涉众、过程和输出

    1. 合作与准备：评估团队领导和主要项目决策者
        1. 谁：涉众的初步名单
        2. 逻辑：何时？何地？如何？
        3. 评估报告何时交付给何人
        4. 评估报告包含何种信息
    2. 评估 1：评估团队和项目决策者，展示 ATAM、展示业务驱动因素、展示架构、确定架构方法、生成效用树、分析架构方法
        1. 架构的简短展示

    2. 业务目标的表达（驱动因素）
    3. 作为场景实现的特定质量属性需求的优先级列表
    4. 效用树
    5. 风险和非风险
    6. 敏感点和权衡点
3. 评估 2：评估团队、项目决策者和架构涉众，展示 ATAM 和结果、头脑风暴和定场景优先级、分析架构方法、展示结果
    1. 涉众社区的优先级场景列表
    2. 风险主题和受到威胁的业务驱动因素
4. 后续行动：评估团队和主要涉众
    1. 最终评估报告

49. 总结 ATAM 的输出

    - 架构的简短展示
    - 业务目标的表达
    - 由质量属性场景表达的定优先级的质量属性需求
    - 效用树
    - 一组风险和非风险
    - 一组风险主题
    - 从架构决定到质量需求的映射
    - 一组确定的敏感点和权衡点
    - 最终评估报告

50. 架构的来源：非功能性需求、架构重要需求、质量需求、涉众、组织、技术环境、商业因素

51. 架构的关注点：需求、质量属性、特性请求、担忧、风险、复杂度、适合于环境

52. 架构的涉众：架构师、需求工程师、开发人员、测试人员、集成人员、维护人员、需要互操作的其他系统的设计人员、质量属性专家、经理、产品线经理、质量确保团队、客户、终端用户、产品线应用程序构建者、分析师、基础设施支持人员

53. 面向服务架构是一个分布式组件的集合，这些组件为其它组件提供服务，或者消费其它组件所提供的服务，而无需知道其它组件的实现细节，服务自身高内聚、服务间松耦合，最小化开发维护中的相互影响，良好的互操作性，符合开放标准，模组化，高重用性，服务动态识别、注册、调用，但是系统复杂度高，难以测试验证，各独立服务演化不可控，中间件易成为性能瓶颈

54. 面向服务架构实现原则

    - 服务解耦：服务之间的关系最小化，只是相互知道接口
    - 服务契约：服务按照描述文档所定义的服务契约行事
    - 服务封装：除了服务契约所描述内容，服务将对外部隐藏实现逻辑
    - 服务重用：将逻辑分布在不同的服务中，以提高服务的重用性
    - 服务组合：一组服务可以协调工作，组合起来形成定制组合业务需求
    - 服务自治：服务对所封装的逻辑具有控制权
    - 服务无状态：服务将一个活动所需保存的资讯最小化

55. 微服务架构风格是一种将单一应用程序开发为一组小型服务的方法，每个服务运行在自己的进程中，服务间通信采用轻量级通信机制，这些服务围绕业务能力构建并可通过自动部署机制来独立部署。微服务架构本质上仍然是一种分布式架构，也是面向服务架构的一种扩展

56. 微服务架构特点：通过服务组件化、围绕业务能力组织、内聚与解耦、去中心化、基础设施自动化、高可用性和可变更与演化的服务设计、服务颗粒化、责任单一化、运行隔离化、管理自动化

57. 微服务核心模式即针对采用微服务系统在特定场景下的特定问题，所使用的成熟的架构解决方案集合，包括服务注册与发现、API 网关、熔断器