**实验十 软件体系结构设计（二）**

**实验目的：**

1. 体系结构风格和视图特点

2. 研究经典软件体系结构案例

3. 继续补充和修改自己项目的SAD

**实验内容：**

**1. 对比书上各种软件****体系结构风格和视图特点，思考自己项目属于哪种设计风格？**

**网上搜索最新的软件体系结构资料，如MVC、Kruchten 4+1视图等。**

**表 1 软件体系结构风格**

|  |  |  |  |
| --- | --- | --- | --- |
| 软件体系结构 | 特点 | 优势 | 应用场景 |
| 管道和过滤器(Pipeline and Filter) | 此结构将处理过程分解为一系列的过滤器，每个过滤器对数据进行处理并传递给下一个过滤器，通过管道相连。每个过滤器都是一个独立的组件，只关注自己的转换任务，这促进了模块化和重用。 | 高度解耦，易于添加、修改或移除过滤器；适合数据处理和转换任务。 | 数据流处理、文本处理、图像处理等。 |
| 客户-服务器(Client-Server) | 系统被分为两部分，客户端负责用户界面和发起请求，服务器负责处理请求、执行业务逻辑和存储数据。 | 集中管理数据和资源，易于维护和升级服务器端；提高了系统的可伸缩性。 | Web服务、数据库访问、企业应用等。 |
| 对等网络(Peer-to-Peer, P2P) | 网络中的每个节点（对等体）既是客户端也是服务器，能够直接与其他对等体交流，没有中心服务器协调。 | 去中心化，提高了系统的健壮性和可扩展性；资源共享效率高。 | 文件共享（如BitTorrent）、分布式计算、区块链网络等。 |
| 发布-订阅(Publish-Subscribe) | 在此模型中，发布者发布消息到特定主题，订阅者订阅这些主题以接收消息。中间件（如消息队列）负责匹配和分发。 | 解耦了消息生产者和消费者，支持一对多通信，易于扩展。 | 实时消息系统、物联网(IoT)、事件驱动架构等。 |
| 信息库(Repository) | 强调的是集中存储和管理共享数据的结构，信息库作为核心组件，为其他组件提供数据访问服务。 | 数据一致性好，便于统一管理和控制数据访问权限。 | 内容管理系统(CMS)、配置管理系统等。 |
| 分层(Layered / Hierarchical) | 将系统划分为多个层次，每个层次承担特定的功能，上层依赖下层的服务，但各层之间相对独立。 | 清晰的模块划分，易于维护和扩展，支持技术多样性。 | Web应用程序、操作系统架构、大多数企业级应用等。 |

**表 2 软件体系结构视图**

|  |  |  |
| --- | --- | --- |
| 体系机构视图 | 关注点 | 用途 |
| 分解视图 | 此视图关注系统的逻辑组织，展示系统的组件（如模块、类、包等）以及它们之间的关系。它描述了系统的功能结构，包括组件的职责、接口、以及它们如何协同工作来满足系统的需求 | 用于指导系统的设计和开发，帮助理解系统的核心功能模块划分。 |
| 依赖视图 | 强调组件或模块间的依赖关系，显示哪些部分依赖于其他部分，以及这些依赖的方向和性质（如编译时依赖、运行时依赖等）。 | 有助于管理复杂性，确保修改一个部分不会意外影响到其他部分，同时也是持续集成和测试策略的基础。 |
| 泛化视图 | 描述组件或模块之间的继承、接口实现等泛化关系，展现了系统的抽象与具体实现之间的层次结构。 | 有助于理解系统的可扩展性和可维护性，支持软件复用和灵活的设计变更。 |
| 执行视图 | 关注系统的并发和同步机制，描述了在运行时系统中各个进程或线程如何协作执行，以及资源管理和通信模式。 | 对于性能优化、并发控制和故障隔离至关重要，特别是在分布式系统和实时系统中。 |
| 实现视图 | 细节化地展示系统组件的物理实现，包括编程语言选择、库的使用、API集成等。 | 指导编码工作，帮助开发者了解如何将逻辑设计转化为实际代码。 |
| 部署视图 | 描述系统如何部署到硬件上，包括服务器、网络配置、中间件等物理资源的布局。 | 为系统管理员提供安装、配置和运维的指导，确保系统的可操作性和可靠性。 |
| 工作分配视图 | 虽然不是所有架构框架中的标准视图，但它关注的是项目团队成员如何分工，哪些团队或个人负责系统的哪个部分。 | 在大型项目中特别重要，用于项目管理和协调，确保团队成员明确自己的责任范围，促进有效的沟通和协作。 |