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？

软件体系结构视图

|  |  |  |
| --- | --- | --- |
| 体系结构视图 | 特点 | 用途 |
| 分解视图 | 概念性，不涉及具体实现细节，用例，用框图、包、类、组件表示，可能用箭头指向调用虚线表示依赖关系。 | 展示软件的高层模块或组件及其相互关系。通过模块间的关系，体现功能的分解，展示模块化设计，帮助理解系统架构的宏观组织。 |
| 依赖视图 | 动态，明确构建、运行时依赖，有助于识别耦合、管理版本冲突，用实线表示直接依赖关系，有方向箭头指向依赖。 | 显示组件间依赖关系，包括库、模块、接口、服务、库、框架间的依赖。 |
| 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 | 时间和顺序、通信、同步、异步进制，助于理解系统性能分析、并发问题。用箭头、状态、活动节点、边表示流程，可能有并发线程或同步机制。 | 描述软件运行时的并发、交互和进程、线程或线程的序列，动态视图。 |
| 实现视图 | 细，具体，详细，面向开发、维护、测试。用类图、对象表示，箭头、接口、实线表示实现。 | 细实层面，源码模块、类、接口、对象、方法间的映射关系，微观结构。 |
| 部署视图 | 静态，运行环境，物理、网络配置，定位、部署规划，用服务器、网络设备、数据库、存储表示，箭头、网络连接。 | 系统件、软硬件、网络、配置、物理架构、基础设施的布局。 |
| 工作分配视图 | 任务分配明确、依赖关系和协调、沟通工具支持 | 工作分配图在项目管理和团队协作中扮演关键角色，它展示了任务、活动或工作项如何分配给各个团队成员，以及他们之间的依赖关系。 |

软件体系结构风格

|  |  |  |  |
| --- | --- | --- | --- |
| 软件体系结构风格 | 概述 | 优点 | 缺点 |
| 管道和过滤器 | 构件被称为过滤器，负责对数据进行加工；连接件被称为管道，是输入数据流和输出数据流之间的通路。每个过滤器是一个独立的个体元素，各个过滤器互不相关；每个过滤器独立完成自己的任务，不同过滤器之间不需要交互。数据输出的最终结果与各个过滤器的执行顺序无关。 | 简单性、支持复用、系统具有可扩展性和可进化性、系统并发性、便于系统分析 | 系统处理过程时采用批处理方式；不适合用来设计交互式应用系统；因为没有通用的数据传输标准，导致每个过滤器都要花费时间进行添加和去除因解析输入数据和合成数据产生的标记，降低了系统性能，增加了过滤器设计的复杂性；难以进行错误处理 |
| 客户-服务器 | 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。 | 1.客户机构件和服务器构件分别运行在不同的计算机上，利于分布式数据的组织和处理  2.构件之间的位置相互透明，不必考虑对方的实际存储位置  3.客户机侧重数据的显示和分析,服务器则注重数据的管理  4.构件之间彼此独立且隔离,软硬件环境的配置灵活性高  5.系统功能扩展性强  6.将大规模业务逻辑分布给网络连接的低成本计算机以降低系统开销 | 发成本较高。  大部分开发工作都集中在客户机程序的设计,增加了设计的复杂度。  信息内容和形式单一 ；维护费用高 |
| 对等网络 | 每一个构件都只执行它自己的进程，，且对于其他同级构件，每个构件本身既是客户端又是服务器。每个构件都有一个接口，该接口不靠定了该构件所提供的服务，而且指定了它向其他同级构件所请求的服务。端与端之间通过彼此发送诸求的方式来实现通信。 | 易于安装,复杂性更低,成本更低 | 无集中管理,不是很安全,不可扩展 |
| 发布-订阅 | 在发布-订阅的体系结构中，构件之间通过对事件的广播和反应实现交互。如果一个构件对某  事件感兴趣则可订阅该事件，一旦该事件发生了，另一个构件则进行发布来通知订阅者。发布-证所隐含的基础结构将负责注册订阅事件以及向合适的构件传达发布的内容。 | 这种系统为系统演化和可定制性提供了强有力的支持。因为所有的交互都是通过事件来合实现的，任何的发布-订阅构件都能添加到系统中去，并且能够在注册的时候不影响到其他构件。  同理，我们能够在其他事件驱动的系统中轻松地复用发布-订阅系统的构件。 | 在宣布事件的时候，构件能够传输数据。但是如果构件需要共享固定不变的数据，系统则须包含一个信息库来支持这种交互，而这种共享可能会减弱系统的可扩展性和可复用性。发布-订阅系统不易于测试，因为发布构件的行为取决于监视事件的订阅者。因此，我们.能独立地测试这种系统的构件，而要在一个集成系统中才能推断出构件的正确性。 |
| 信息库 | 信息库风格的体系结构由两类构建组成：中心数据存储以及与其相关的访问构件。共享数据存放于数据存储之中，而数据存取器是一个计算单元，它负责存储、检索以及更新信息。设计这样的系统是一个挑战，因为我们必须决定这两种类型的构件将如何进行交互。 | 数据一致性好，因为所有应用程序都访问相同的数据仓库。  易于实现数据共享和集成。 | 可能导致数据访问冲突和并发问题。  数据仓库的维护和管理可能比较复杂。 |
| 分层 | 分层系统(layered system)将系统的软件单元按层次化组织，每一层为它的上层提供服务，同时又作为下层的客户。在一个“纯粹的”分层系统中，各层中的软件单元只能访问同层中的其他单元和相邻低层的接口所提供的服务。但是为了提高性能，在一些情况下这些条件也可能会放宽松，可以允许一个给定层访间所有低层的服务，这称为层次桥接(layerbrdging)。 | 结构清晰，易于理解和维护。 提高了软件的可重用性和可扩展性。  便于进行单元测试和集成测试 | 不是每个系统都适合采用分层结构。  层次之间的交互和通信可能比较复杂 |

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

在网上搜索关于软件体系结构的最新资料，特别是提及的"4+1视图模型"，以下是一些关键信息的总结：

（1） Kruchten 4+1视图模型概述

Philippe Kruchten在19995年提出的"4+1视模型"，是软件体系结构的经典框架，广泛应用于软件工程中。该模型通过五个不同的视图来全面描述软件架构，以确保系统的所有关键方面得到考虑，便于不同利益相关者理解：

逻辑视图：关注重于系统功能结构，如何系统分解成模块、类、组件，以及它们如何交互来实现系统功能。

进程视图：说明并发和线程如何交互，系统中活动的执行，关注性能和响应时间。

物理视图：映射软硬件和部署，系统如何映射到物理环境。

场景视图：用例，特定的交互序列，展示系统如何在关键场景下如何工作。

（2）在Web上的资源和更新

查找该模型的最新资料，推荐访问权威的学术期刊、专业网站、技术博客和会议论文库，如ACM、IEEE Xplore、Springer、ACM Digital Library等。直接搜索"4+1视图"或"Kruchten 4+1模型"获取最新研究进展、案例研究、应用实例、扩展和实践。

（3）教程与工具

YouTube和在线教育平台\*\* 上也有很多视频教程和课程深入解释4+1模型，例如Coursera、Udemy、YouT上搜索"4+1视图"。这些资源通常会提供实例解析模型的应用方法、工具如ArchiMate、Enterprise Architect、Visio的使用教程。

（4）社区和论坛

参与技术社区\*\*如Stack Overflow、Reddit（r/programmers、CSD、Quora上关于架构话题标签）、GitHub上的讨论，可以获得实践者的见解、问题解答和最新趋势。搜索"4+1视图"查看同行交流。

综上，通过这些渠道，可以找到关于4+1视图模型的最新研究、应用实例、工具、教程和社区讨论，以及如何将其应用于现代软件架构设计和分析。

3. 参阅课本和网上资料，研究经典软件体系结构案例KWIC。

An Introduction to Software Architecture，4.1节

On-the-Criteria-To-Be-Used-in-Decomposing-Systems-into-Modules（Example System 1）

http://www.cs.cmu.edu/~ModProb/index.html

An Introduction to Software Architecture

软件架构是软件工程的核心，它定义了系统组成、组件如何组织、交互、以及这些组件如何协同工作以实现系统整体目标。一个良好的架构设计不仅关注技术细节，也考虑了业务需求、未来扩展性、可维护性、安全性等非功能性需求。

On-the-Criteria-Be-Used-in-Decomposing-Into-Modules（Example System）

模块化是架构的关键，决定如何分解系统的原则：

功能内聚类：相关性，确保模块内高内紧密度高，模块间低。

独立性：降低耦合度，模块间相互依赖，便于替换、修改。

可重用性：抽用，模块能在不同情境，减少重复代码。

理解性：易懂，模块清晰命名、结构简单。

该资源可能提供了更深入的案例分析、教学材料，通过实例讲解架构原则应用。这可能涵盖实际软件设计的步骤、架构模式（如MVC、分层、微服务架构）、架构风格（如Kruchten 4+1模型）等，以及如何解决特定挑战如性能优化、安全、扩展、部署等。

针对KWIC和自己项目，参考课本ch5 表5-3，小组成员每人给几种不同的体系结构风格设计打分，评最佳。

4. 补充和修改自己项目的SAD

记录项目及小组每个人工作的进度、里程碑、工作量的跟踪图或表，将其保存到每个小组选定的协作开发平台上，每周更新。