**实验十 软件体系结构设计（二）**

**夏思晓 202100031020**

**一、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

**二、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？**

表 1 软件体系结构风格特点汇总表

|  |  |  |  |
| --- | --- | --- | --- |
| **软件体系**  **结构风格** | **概述** | **优点** | **缺点** |
| 管道和  过滤器 | 数据通过一系列的处理阶段（即“过滤器”）进行转换，每个阶段之间的数据传递通过“管道”进行。 | 1、简单。  2、支持复用。  3、系统具有可扩展性和可进化型。  4、系统并发性（每个过滤器可以独立运行，不同子任务可以并行执行，提高效率）。  5、便于系统分析。 | 1、系统处理工程是批处理方式。  2、不适合用来设计交互式应用系统。  3、由于没有通用的数据传输标准，因此每个过滤器都需要解析输入数据和合成数据。  4、难以进行错误处理。 |
| 客户-服务器 | 分布式计算模式，其中客户端向服务器请求服务，服务器处理请求并返回结果。 | 1、客户机构件和服务器构件分别运行在不同的计算机上，有利于分布式数据的组织和处理。  2、构件之间的位置是相互透明的，客户机程序和服务器程序都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、构件之间是彼此独立和充分隔离的。  5、将大规模的业务逻辑分布到多个通过网络连接的低成本的计算机，降低了系统的整体开销。 | 1、开发成本较高。  2、在开发C/S结构系统时，大部分工作都都集中在客户机程序的设计上，增加了设计的复杂度。  3、信息内容和形式单一。  4、如果对C/S体系结构的系统进行升级，开发人员需要到现场来更新客户机程序，同时需要对运行环境进行重新配置，增加了维护费用。  5、数据安全性不高。 |
| 对等网络（P2P） | 分布式应用架构，其中每台计算机都作为网络服务的请求者和提供者，无主从之分。 | 1. 规模易于扩展。 2. 数据被很多同级构件复制并且分布在其间，对于构件和网络的故障有很好的容错性。 3. 网络成本低，因为不需要专门的服务器硬件。 4. 网络配置和维护简单，因为每台计算机都可以独立运行 | 1. 网络性能较低，因为每台计算机都需要处理自己的请求和响应。 2. 数据保密性差，因为每台计算机都可以访问网络中的资源。 |
| 发布-订阅 | 事件驱动的消息传递模式，其中发布者发布消息，订阅者接收并处理感兴趣的消息。 | 1. 系统演化和可定制性提供了强有力的支持。 2. 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。 3. 解耦度高，发布者和订阅者之间不需要直接通信或知道对方的存在。 4. 灵活性强，可以动态地添加或删除订阅者。 | 1. 可能导致消息传递的延迟和丢失，因为发布者无法确保消息一定会被所有订阅者接收。 2. 管理和维护订阅关系可能比较复杂。 3. 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。 4. 数据共享可能会减弱系统的可扩展性和可复用性。 |
| 信息库 | 基于共享数据仓库的软件体系结构，其中多个应用程序通过访问共享的数据仓库来交换信息和协作。 | 1. 数据一致性好，因为所有应用程序都访问相同的数据仓库。 2. 易于实现数据共享和集成。 | 1. 可能导致数据访问冲突和并发问题。 2. 数据仓库的维护和管理可能比较复杂。 |
| 分层 | 将软件系统划分为多个层次的结构，每个层次都为其上层提供服务，并作为下层的客户。 | 1. 结构清晰，易于理解和维护。 2. 提高了软件的可重用性和可扩展性。 3. 便于进行单元测试和集成测试。 | 1. 不是每个系统都适合采用分层结构。 2. 层次之间的交互和通信可能比较复杂。 |

表 2 软件体系结构视图特点汇总表

|  |  |  |
| --- | --- | --- |
| **软件体系结构视图** | **特点** | **用途** |
| 分解视图 | 分解视图呈现为较为明晰的分解结构。它用软件模块勾划出系统结构，形成层次化的结构。 | 帮助开发人员理解系统的整体结构，以及各个模块之间的层次关系。 |
| 依赖视图 | 展现了软件模块之间的依赖关系。如果一个软件模块调用了另一个软件模块或依赖其产生的数据，那么这两个模块之间就存在依赖关系。 | 帮助开发人员识别和理解模块之间的依赖关系，从而确保模块之间的正确交互和协作。 |
| 泛化视图 | 展现了软件模块之间的一般化或具体化的关系，如面向对象分析和设计方法中类之间的继承关系。这种关系不仅限于继承，还包括其他形式的一般化或具体化关系。 | 帮助开发人员理解类之间的继承层次和关系，从而确保代码的可重用性和可扩展性。 |
| 执行视图 | 展示了系统运行时的时序结构特点，如流程图、时序图等。执行视图中的每个执行实体（通常称为组件）都是唯一的，相似的执行实体会被合并。 | 帮助开发人员理解系统的运行流程和时序关系，从而确保系统的正确性和性能。 |
| 实现视图 | 描述了软件架构与源文件之间的映射关系。它关注于软件的静态组织结构，特别是在开发环境中。 | 帮助开发人员理解软件的实际实现方式，包括源代码的组织结构、模块之间的交互方式等。 |
| 部署视图 | 描述了软件到硬件的映射关系，反映了分布式特性。它关注于如何将软件部署到硬件上，以及各个硬件之间的通信和协作关系。 | 帮助开发人员理解软件的部署方式和运行环境，从而确保软件的正确性和可靠性。 |
| 工作分配视图 | 将系统分解成可以分配给各项目团队的工作任务。 | 有助于项目管理人员跟踪各个团队的工作进度，而且有利于计划和分配工程资源。 |

小组项目的线上人才招聘系统采用的是组合体系结构风格，包含客户-服务器风格、发布-订阅风格、信息库风格和分层风格等，具体分析如下：

**·** 线上人才招聘系统通常包含一个或多个服务器，用于处理用户请求、存储和管理数据。客户端（如Web浏览器或移动应用）用于与用户交互，并向服务器发送请求。具体应用表现为用户通过客户端浏览职位、提交简历、参加面试等，而服务器则处理这些请求，如检索数据、验证用户身份、发送通知等。

**·** 在招聘系统中，发布-订阅风格可以应用于职位发布和订阅功能。当有新的职位发布时，系统可以将这些信息发布给订阅了该职位的求职者。具体应用表现为求职者可以订阅感兴趣的职位类别或关键词，当有符合其订阅条件的职位发布时，系统会通过邮件、短信或应用内通知等方式提醒求职者。

**·** 信息库风格适用于需要集中存储和管理大量数据的应用场景，如招聘系统中的职位数据、用户数据、简历数据等。具体应用表现为系统使用一个或多个数据库来存储这些数据，并通过数据访问层（如ORM框架）来提供数据的增删改查功能。这样可以确保数据的一致性、安全性和可维护性。

**·** 分层风格将系统划分为多个层次，每个层次都负责特定的功能，如表示层、业务逻辑层、数据访问层等。具体应用表现为在线上人才招聘系统中，分层风格可以确保系统的模块化和可扩展性。例如，表示层负责与用户交互和展示数据，业务逻辑层负责处理业务规则和逻辑，数据访问层负责与数据库交互以获取和存储数据。

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

**MVC架构：**

MVC架构是一种广泛使用的软件设计模式，它将应用程序的逻辑、数据和用户界面分离为三个核心部分：模型（Model）、视图（View）和控制器（Controller）。近年来，MVC架构已经发展出了许多变体，以适应不同的应用程序需求和技术环境。

变体架构：基于MVC架构的成功，产生了许多变体架构，如MVVM（Model-View-ViewModel）、MVP（Model-View-Presenter）和MVW（Model-View-Whatever）等。这些变体架构根据特定的需求和技术环境进行了一些调整和改进。

前端框架和库：随着Web应用程序的兴起，各种前端框架和库如Angular、React和Vue.js等的出现，为MVC架构的实现提供了更好的支持和工具。这些框架和库通常与MVC架构或其变体相结合，以提高开发效率和应用程序的性能。

RESTful架构：MVC架构与RESTful架构相互配合，形成了一种常见的Web应用程序设计方式。模型部分可以与RESTful API进行交互，通过HTTP协议进行数据传输，实现前后端分离和可扩展的架构。

响应式架构：随着移动设备的普及和用户体验的要求提高，出现了响应式架构的概念。响应式架构注重于在不同设备和屏幕尺寸上提供一致的用户体验，这与MVC架构相结合，可以创建出既美观又实用的应用程序。

**Kruchten的“4+1”视图模型：**

“4+1”视图模型是由Philippe Kruchten在1995年提出的，用于从多个角度描述软件体系结构。该模型包括五个主要的视图：逻辑视图、过程视图、物理视图、开发视图和场景视图。

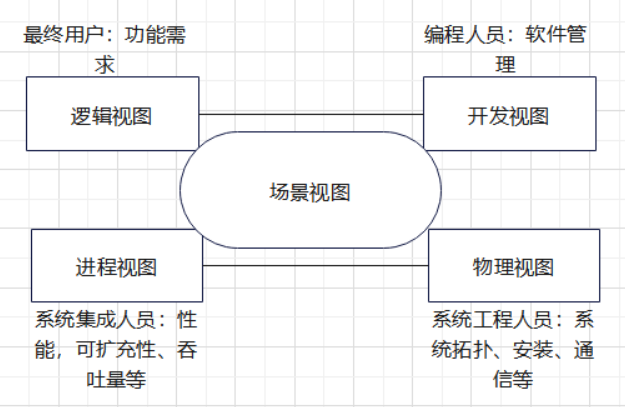
逻辑视图：设计的对象模型，使用面向对象的设计方法时。它描述了系统的功能需求，包括类、接口、以及它们之间的关系。

过程视图：捕捉设计的并发和同步特征。它关注于系统的动态行为，包括进程、线程、以及它们之间的通信和同步机制。

物理视图：描述了软件到硬件的映射，反映了分布式特性。它关注于系统的物理部署和配置，包括硬件设备、网络拓扑、以及软件组件在硬件上的分布。

开发视图：描述了在开发环境中软件的静态组织结构。它关注于系统的开发和管理，包括源代码组织、构建过程、以及版本控制等。

场景视图：综合所有的视图，通过用例和场景来描述系统的行为。它提供了一个全局的视角，帮助开发人员理解系统的整体需求和功能。



更多软件体系结构参考资料：

① [软件体系结构——4+1视图-CSDN博客](https://blog.csdn.net/lixiangTrad/article/details/102850708)

② [详解系统架构的“4+1”视图 (zhihu.com)](https://www.zhihu.com/tardis/zm/art/352590602?source_id=1005)

**2. 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**

**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。**

**KWIC（上下文中的关键词）索引系统**接受一组有序的行：每一行都是一组有序的词，并且每一个词都是有序的字符集合。任何一行都可以通过反复地将第一个词移到行的末尾来进行“循环移位”。KWIC索引系统按照字母顺序输出所有行的循环移位列表。课本中提出了几种不同的体系结构风格，包括：

① **共享数据（Shared Data）**：组件通过共享内存区域进行通信。

② **数据抽象（Data Abstraction）**：使用抽象数据类型来封装数据和行为。

③ **隐含调用（Implicit Invocation）**：基于事件的系统，组件通过事件进行通信。

④ **管道和过滤器（Pipes and Filters）**：数据流通过一系列过滤器，每个过滤器执行特定的操作。

表 3 体系结构风格设计打分表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **小组成员** | **共享数据** | **数据抽象** | **隐含调用** | **管道和过滤器** |
| 陈煜 | 3 | 2 | 4 | 2 |
| 姜世慧 | 3 | 4 | 4 | 3 |
| 夏思晓 | 3 | 4 | 5 | 3 |
| 阎笑 | 3 | 3 | 4 | 1 |
| 郑家彤 | 3 | 2 | 5 | 1 |
| 总计 | 15 | 15 | 22 | 10 |

评分标准说明：

易于改变算法：5分表示该风格非常灵活，可以轻松适应算法的变更。

易于改变数据表示：5分表示该风格支持数据表示的轻松修改，不影响其他组件。

易于改变功能：5分表示可以轻松添加或修改功能，而不需要大规模重构。

好的性能：5分表示该风格优化了性能，可以处理高负载。

有效的数据表示：5分表示数据在系统中以最高效的方式表示和存储。

易于复用：5分表示组件可以在不同的上下文中重复使用，具有高度的通用性。

针对KWIC和团队项目，评选出的最优体系结构风格为**隐含调用**。

**3. 补充和修改自己项目的SAD**

（详细内容见小组项目文件《软件体系结构设计说明（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。