计算机学院软件工程与实践课程实验报告

|  |  |  |
| --- | --- | --- |
| 实验题目：软件体系结构设计（二） | | 学号：201920130238 |
| 日期：2022-5-5 | 班级： 计科19级1班 | 姓名： 陈文盼 |
| Email：2220533881@qq.com | | |
| 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 | | |
| 实验软件和硬件环境：  Windows10操作系统 | | |
| 实验步骤与内容：  团队名称：没有bug软工实践团队  团队成员（排名不分先后）陈文盼 201920130238，史子涵 201900130126，刘诗婷201918130217， 赵一帆 201905130197，郑晓旭 201900130107。  任务分配如下：  在2022-5-5晚上7时，我们小组举行本周第一次的会议，初步分配了三个任务：分工调研各种软件体系结构风格和视图特点；分工协作，继续完善自己所负责的SAD部分；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。  经过讨论：陈文盼负责管道和过滤器，客户-服务器两个结构；史子涵负责组合体系结构风格，MVC体系结构这两个结构；刘诗婷负责Kruchten"4+1"视图模型；赵一帆负责对等网络，发布-订阅两个结构；郑晓旭负责仓库风格，分层的体系结构风格。各自完善所负责的SAD,最后大家一起打分。   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。   在2022年5月8日星期日20时，我们小组举行本周第二次的会议，集中讨论自己的软件体系结构风格和视图特点的调研情况，具体的调研结果在本篇报告的问题讨论中有所体现。结合调研结果以及我们小组全体对我们所需制作的软件系统的分析，我们最终决定采用Kruchten 4+1作为我们软件体系结构风格  二、每人给几种不同的体系结构风格设计打分，评最佳。 | | |
| 结论分析与体会：  本周的实验重点在于软件体系结构风格和视图特点的分析与对KWIC几种不同的体系结构风格设计打分。软件的结构风格和视图特点主要是客户机、服务器之间的关系。通过这次实验，我们小组对相关的体系结构和视图特点有了一定了解，还了解了保证质量属性的相关策略，这对我们小组后面的设计有很大作用。 | | |
| 就实验过程中遇到的问题及解决处理方法，自拟1－3道问答题：  介绍各种软件体系结构风格和视图特点？  管道和过滤器    过滤器：输入数据通过过滤器的数据转换部件而生成输出数据。不同的过滤器之间相互独立。  管道：简单地将数据从上一个过滤器的输出端传递到下一个过滤器的输入端。  客户-服务器    服务器：提供服务。服务器不知道正在向哪个用户提供服务，也不知道正在向几个用户提供服务。服务器仅是简单地对用户发起的请求进行处理。  客户：通过请求-应答协议访问服务。客户知道正在向哪一个服务器发起请求。  对等网络：  每个构件都只执行它自己的进程，并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，该接口不仅指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请求的方式来实现通信。该方式很类似于我们在客户-服务器体系结构中的请求/应答方式，但不同的是，本方式中任意一个构件都可以向其他同级构件发送请求。  发布-订阅  在该结构中，构件之间通过对事件的广播和反应实现交互。如果一个构件对某个时间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-订阅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。发布-订阅构件对其他构件的存在一无所知，相反，发布者只是简单地宣布事件，然后等待反应；订阅者只是简单地对事件通知做出反应，而不管事件是如何发布的。  仓库风格  构件：  中央数据单元，代表系统当前的各种状态  相对独立的构件集合，对中央数据单元进行操作  两种控制策略的选取产生两个主要的子类：  输入数据流中的事务指令触发系统相应进程的执行:则仓库是传统数据库体系结构  中央数据结构的当前状态触发系统相应进程的执行:则仓库是一个黑板体系结构;黑板体系结构是仓库体系结构的特殊化  黑板系统    分层的体系结构风格  典型范例（内、外分层）    构件：组织为层次结构，每一层给外层提供服务，又作为它内层的客户；某些系统中，内层只对相邻的层可见  连接件：层间的协议，定义了层次间的交互方式    TASK1：查阅组合体系结构风格和MVC的相关资料  1.组合体系结构风格  在我们的显示开发过程中，很少有使用单个体系结构风格的情况，实际上，多数情况下，我们都是将不同风格的体系结构组合使用。选择并调整该风格中的某些问题用来解决整个系统中的一部分问题。  1.1体系结构风格的若干组合方式  1.1.1分解系统的层次，不同级别使用不同风格。举个例子，我们的系统属于客户端-服务器的结构，但是我们就单纯来看客户端或者服务器，我们就可以将其分解成若干层：或者某抽象层上构建之间的连接在更底层上却是若干构件和连接器的结合。例如，我们可以对发布-订阅体系结构的交互进行分解，详细体现用以管理事件订阅，事件发布的通知机制。  1.1.2体系结构可以使用一个混合的风格来为不同的构件或者构件间不同的类型的交互建模。例如下图的范例所示。    客户构建之间使用发布-订阅的通信方式进行交互。另一方面，这些相同的构件都通过请求/应答协议来使用服务器构件，这些服务器构件又和一个共享数据信息库进行交互。在这个例子中，通过允许一个构件可以担任多种角色（如客户、发布者和订阅者）、可以有多种交互方式，该体系结构将多种风格集成为单个的模型。  •当体系结构风格之间可以互相兼容时，风格的集成将会更加容易，比如，所有要组合的风格都和运行时构件或代码单元有关。我们还可以创建以及维护不同体系结构的视图，这和建筑工程师所做的有异曲同工之处（如，布线视图、管道视图、供暖和通风视图等）。在如下情况下这种方法会非常合适：各种视图集成后过度复杂，构件之间有多种交互方式（如，构件之间同时使用隐含调用和显示方法调用），或者各视图构件之间的映射过于混乱（也就是说,形成了多对多的关系）  MVC体系结构  MVC即Model、View、Controller即模型、视图、控制器。  MVC是经常被Web应用程序体系结构采用的设计模式。该模式植根于Smalltalk。在Smalltalk中，应用程序常使用复杂的技术通过接口(视图)来查看业务数据(模型)，每当业务对象改变状态时，控制器对象就会自动更新和管理接口。  MVC包括三类对象，model是应用对象，view是视图，controller是控制器，它定义用户界面对用户输入的响应方式。  View层是界面，Model层是业务逻辑，Controller层用来调度View层和Model层，将用户界面和业务逻辑合理的组织在一起，起粘合剂的效果。所以Controller中的内容能少则少，这样才能提供最大的灵活性。MVC要实现的目标是将软件用户界面和业务逻辑分离以使代码可扩展性、可复用性、可维护性、灵活性加强。    MVC的优缺点：  优点：分层，结构清晰，耦合性低，大型项目代码的复用性得到极大的提高，开发人员分工明确，提高了开发的效率，维护方便，降低了维护成本。  缺点：简单的小型项目，使用MVC设计反而会降低开发效率，层和层虽然相互分离，但是之间关联性太强，没有做到独立的重用。  Kruchten"4+1"视图模型，从 5 个不同的视角包括包括逻辑试图、处理视图（过程视图）、物理视图、开发视图、场景视图来描述软件体系结构。每一个视图只关心系统的一个侧面，5 个试图结合在一起才能反映系统的软件体系结构的全部内容。    逻辑试图主要是用来描述系统的功能需求，即系统提供给最终用户的服务. 在逻辑视图中，系统分解成一系列的功能抽象、功能分解与功能分析，这些主要来自问题领域。在面向对象技术中，通过抽象、封装、继承,可以用对象模型来代表逻辑视图，可以用类图来描述逻辑视图。  开发视图主要用来描述软件模块的组织与管理。服务于软件编程人员，方便后续的设计与实现。它通过系统输入输出关系的模型图和子系统图来描述。要考虑软件的内部需求：开发的难易程度、重用的可能性，通用性，局限性等等。开发视图的风格通常是层次结构，层次越低，通用性越好。  处理视图侧重系统的运行特性，关注非功能性的需求。服务于系统集成人员，方便后续性能测试。强调并发性、分布性、集成性、鲁棒性、可扩充性、吞吐量等。定义逻辑视图中的各个类的具体操作是在哪一个线程中被执行。  物理试图主要描述硬件配置。服务于系统工程人员，解决系统的拓扑结构、系统安装、通信等问题。主要考虑如何把软件映射到硬件上，也要考虑系统性能、规模、可靠性等。可以与进程视图一起映射。  场景用于刻画构件之间的相互关系，将四个视图有机地联系起来。可以描述一个特定的视图内的构件关系，也可以描述不同视图间的构件关系。文本、图形表示皆可。 | | |