**2022年春（2020级软卓）**

**《软件体系结构》**

**课程总结报告**

姓名：赵庆举 学号：202024100738 班级:软卓一班

**郑州大学计算机与人工智能学院**

**2022年 6 月 7日**

目录

[《软件体系结构》 1](#_Toc105543709)

[一、说说你对软件体系结构的认识。 3](#_Toc105543710)

[二、软件体系结构这门课有哪些内容？ 3](#_Toc105543711)

[软件结构视图：软件体系结构的4+1视图(RUP4+1) 3](#_Toc105543712)

[软件质量属性： 4](#_Toc105543718)

[软件架构风格： 5](#_Toc105543719)

[三、关于这门课程，你做了什么？有什么收获？ 10](#_Toc105543720)

[四、大学期间你开发过哪些软件系统？请分别简单介绍。 10](#_Toc105543721)

[1:小区管理系统 10](#_Toc105543722)

[2:学生管理系统 10](#_Toc105543723)

[五、选择一个你开发过的软件系统，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。 11](#_Toc105543724)

[六、选择一个开源软件系统，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。 12](#_Toc105543725)

[七、选择一个你使用过的非开源软件，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。 12](#_Toc105543726)

# 一、说说你对软件体系结构的认识。

软件体系结构通常也称为架构，是能够预先构件或者重构的软件框架。软件体系结构包括：构件，连接件和约束。连接件是一类特殊的构件，是将构件连接在一起的软件构造体。约束是指按照什么标准或要求将构件连接起来。三者的关系可以表述为：按照系统的性能约束或者功能约束，用连接件将构件组装成软件系统。

软件体系结构是软件设计过程中的一个层次。也可以说软件体系结构是一个抽象的系统规范，主要包括用其行为来描述的功能构件和构件之间的相互连接、接口和关系或者软件体系结构是一个程序/系统各构件的结构、它们之间的相互关系以及进行设计的原则和随时间进化的指导方针，迄今为止还没有一个被大家公认的定义。

软件体系结构能够简化应用开发的流程，让一些大型的应用不必从零开始开发，而是只需要在已有的框架中填入所需的东西，使之能够实现自己的目的即可。而且现有的框架经过大量的使用和升级，结构相比于在有限期限内开发的应用程序可能更加合理，有较高的运行效率，同时也能节省大量的研发资金，节省大量的时间。

# 二、软件体系结构这门课有哪些内容？

关于软件体系结构的概述：软件体系结构这门课包括了软件结构与视图、软件质量属性、软件架构构模式/风格和软件开发框架。

### 软件结构视图：软件体系结构的4+1视图(RUP4+1)

逻辑视图：通常使用面向对象风格，通过抽象、封装和继承，用对象模型来代表逻辑视图。保持单一的哪句的对象模型贯穿整个系统，主要支持系统提供给最终用户的服务，不涉及具体的编译即输出和部署，通常在UML中用类图，交互图，时序图来表述，类似与我们采用OOA的对象模型

开发视图（模块视图）：主要侧重于软件模块的组织和管理，描述软件在开发环境下的静态组织，从程序实现人员的角度透视系统。考虑软件内部的需求（软件开发的容易性、重用和通用性），充分考虑由于开发工具的不同而带来的局限性。开发视图通过系统输入输出的模块图和子系统图来描述。

进程视图：侧重于系统的运行特性，主要关注一些非功能性的需求，主要是进程以及相关的并发、同步、通信等问题。可以描述为多层抽象，每个级别分别关注不同的方面。它可看成一系列独立的，通过逻辑网络相互通信的程序，他们是分布的，通过总线或局域网、广域网等硬件资源连接起来。与逻辑视图不同，开发视图比较关注的正是这些运行时单元的交互问题，在UML中通常用活动图表述。

物理视图：主要考虑如何吧软件映射到硬件上，是从系统工程师解读系统，关注软件的物流拓扑结，以及如何部署机器和网络来配合软件系统的可靠性、可伸缩性等要求。软件运行于不同的节点上时，视图中的构件对应于系统的不同节点上，对映射有较高的灵活性，当环境改变时，对系统其他视图的影响最小。

场景视图：关注最终用户需求，可以看作是那些重要系统活动的抽象，使四个视图有机联系起来，是最重要的需求抽象。场景可以用文本表示，也可以用图形表示，通常用UML用例图和活动图描述。

对于不同的软件系统来说，侧重的角度也有所不同。

### 软件质量属性：

可用性：对一个软件运行程度的评估。可用性的评估公式是(正常运行时间/(正常运行时间+平均修复时间))。对软件而言，需要时刻关注软件的故障。一个软件的故障越少，发生故障后修复的时间越短，正常运行的时间越长，其可用性就越高。

可修改性：可修改性实施满足局部化修改，防止连锁反应、延迟绑定时间。其针对的方面很广。不管是底层架构还是界面开发都必须强调可修改性这一属性。在可修改性的实施中需要强调设计模式是使用，软件设计模式的宗旨是高内聚、低耦合，多以来组合，少继承。

性能：软件中功能的反应时间。在软件开发过程中需要考虑资源需求、资源管理、资源仲裁。为每一个资源进行分配，计算其效率。在程序中优先级分配，最常用的就是时间轮转以及先到先优先原则。若出现资源不可用，则可以设置资源间的时间间隔，分析每个请求所消耗的资源。

安全性：安全性是衡量系统保护数据和信息不受未经授权访问的能力，同时仍能提供对授权人员和系统的访问。身份验证、限制访问，对数据机密性及完整性的维护等都是对攻击抵抗的策略。对检测攻击就是需要基本经验的积累，就是来攻击者与防护数据库对比，通过检测就能知道攻击的类型以及程度，方能及时做出方案反击。常见的安全性技术有防火墙技术、对用户授权。

扩展性：反应软件适应变化的能力。在软件开发过程中，可能遇到需求、设计的变化，算法的改进，程序的变化等等。现代软件开发通常采用“增量开发模式”，不断推出新的版本，获取增值利润，可扩展性非常重要。

伸缩性：系统处理负载增加而不降低性能的能力，或者是迅速增加负载的可能性。。如果系统允许水平拓展，增加拓展所需要的时间（以秒为单位），拓展限制：服务器或网络容量的数量以及拓展的可能性：事务数量的增加或内容量。

### 软件架构风格：

调用返回风格：

调用返回风格包括：主程序/子程序；面向对象风格，层次结构

结构化的程序，符合顶层分解的逻辑，自上而下的功能设计。每一层抽象都会通过向下调用的方法来实现他所定义的功能，逐层分解，从功能的观点设计系统。主程序/子程序由构件、连接器、拓扑结构组成，分别是主/子程序、调用—返回机制、层次化结构。经大系统分解为若干模块，主程序调用这些模块实现完整的系统功能。这种方法已被证明是成功的设计方法，可以被用于较大的程序。但随着程序太大，开发变慢，测试越来越进行困难。

面向对象风格中系统被看作对象的集合，每个对象都有一个自己的功能集合。数据及作用在数据上的操作呗封装成抽象数据类型，被称为对象。面向对象风格的构件是：类和对象；连接件是：对象之间通过功能与函数调用实现交互。其优点是与人类习惯的思维方式方法一致，按照人类习惯的思维方法建立问题域的模型，开发出尽可能直观、自然地表现求解方法的软件系统。有很好的稳定性和可重用性，容易开发出大型软件产品，软件比较容易理解，容易修改，易于测试和调试。面向对象的缺点是需要管理大量的对象，需要确立大量对象的结构。并且需要知道对象的身份（而在管道—过滤器风格的系统中则无需知道其他过滤器的任何消息），并且继承会引起复杂度。

数据流风格：

数据流风格包括：管道—过滤器；批处理

在数据流风格的系统结构中，数据在各处理之间有序移动，出具的可用性决定着处理<计算单元>是否执行。在纯数据流系统中，处理之间除了数据的交换，没有任何其他的交互。数据流风格的构件、连接器、拓扑结构分别为：数据处理、数据流、拓扑结构的图。管道—过滤器风格中，每个过滤器封装了一个处理步骤，其中间的管道连接一个源和目的过滤器作为连接件，连接器定义了数据流图，形成拓扑结构。过滤器中无上下文信息，无保留状态，对其他过滤器无任何了解，可以使用数据缓冲区临时保存数据流，类比于：蓄水池。管道作为连接器在过滤器之间传送数据，不同的管道中流动的数据流，具有不同的数据格式，但都为单向流。

批处理把任务分解成为一系列固定顺序的计算单元，壁纸之间只通过数据传递交互，相比于管道—过滤器，批处理整体传递数据，并且构件粒度较大，缺点有延迟高，实时性差，无并发。数据流风格中各构件设计具有良好的隐蔽行和高内聚、低耦合的特点。并且容易进行复用和扩展，任何两个过滤器都可以连接起来，新的过滤器可以方便的添加到现有的系统中来。但可能会导致系统系统进程成为批处理的结构，整体性能不高，不适合交互应用。

事件驱动风格：

事件驱动风格分为显式调用和隐式调用。

显式调用中各个构件之间的互动是由显性调用函数或程序完成的，调用过程与次序是固定的、预先设定的。隐式调用（基于时间的系统）不直接调用一个过程，而是触发或广播一个或多个时间。当事件被触发，系统自动调用在这个事件中注册的所有过程，这样一个事件的触发就导致了另一模块中的过程的调用。时间驱动系统的连接器为事件—过程绑定。在某些情况下，一个事件也可能触发其他事件，形成时间链，这种链接机制称为，事件—事件绑定。

事件系统能很好的支持交互式系统，系统中的操作可以异步执行，不必同步等待执行结果。促进了软件复用和系统动态演化，对事件的并发处理可以提高系统性能，其中的某个构件出错将不会影响其他构件，使系统有很强的健壮性。事件驱动风格的分布式控制方式使得系统的同步、验证和调试变得异常困难，构件放弃了对系统计算的控制，也不能保证过程调用的顺序。相比于直接连接方式的连接，隐式调用增加了中间层必要的消耗，会降低时间的响应速度。

层次风格：

在层次系统中，系统被组织成若干个层次，每个层次由一系列构件组成，下层构件向上层构件提供服务，上层构件被看作下层构件的客户端。层次系统的基本构件、连接件、拓扑结构分别为：各层次内部包含的构件、层间的交互协议、分层。层次风格支持基于可增加抽象层的设计，允许讲一个负载问题分解成一个增量步骤序列的实现。不同的层次处于不同的抽象级别，越靠近底层越通用、越靠近顶层越具体。每一层最多影响两层，只要给相邻层提供相同的接口，允许每层用不同的方法实现，为软件复用提供了强大的支持。一般情况下，某一层中的构件只能与同一级别中的对等实体或较低级别中的构件交互，这有助于减少不 同级别中的构件之间的依赖性。两种通用的分层方法：严格分层、松散分层。严格分层系统要求严格遵循分层原则，限制一层中的构件只能与对等实体以及与它紧邻的下面一层进行交互。严格分层修改时较为简单，当某一层被加入或被替换时， 它的影响范围很有限，容易管理，但其效率相对较低。过度的严格分层导致效率低下，因此在很多实际系统里，并不遵循严格分层的原则。

层次软件体系结构风格侧重于把软件中的相关功 能和组件分成独立的逻辑分组，即逻辑层 (Layer)，从而组织成一个层次结构。层有助于区分组件进行的不同类型的任务，任何系统在逻辑架构层面都可以看做是把一组相互协作的组件分组成逻辑层。常见的逻辑层包括：表现层（面向用户的功能，负责管理用户和系统之间的交互；通常包含：用户界面组件、表现逻辑组件）、业务层（实现系统的核心功能并且封装了相关业务逻辑。通常包含：应用程序外观、业务逻辑组件）、数据层（抽象了访问底层数据存储的必要逻辑。通常包含：数据访问组件、服务代理）。物理层(Tier) 描述的却是如何把功能和组件从物理上部署 到独立服务器、计算机、网络或远程位置。逻辑层和物理层之间的映射包含三种情况：1对1、1对多、多对1；多层次的系统中可以有由上而下和由下而上两种方案，在某些网络/通讯系统里，信息必须在两个方向都可以传递（双向分层）。

客户服务器风格：

客户服务器是一个物理上分布的逻辑整体，是由客户机、服务器和连接部分组成。客户机是客户服务器系统中的核心部分，是一个面向最终用户的接口设备或应用程序。客户机可向其他设备或应用程序提出请求，再向用户显示得到的信息。C/S可以实现资源共享，可以快速的进行信息处理，当一项任务提出请求处理时，可以在所有可能的服务器间均衡地分布该项任务的负载。这样，在客户端发出的请求可由多个服务器来并行进行处理，为每一项请求提供了极快的响应速度和较高的事务吞吐量。 B/S是三层C/S风格的一种实现方式，其表现层、逻辑层、数据层分别为：浏览器、Web服务器，应用服务器、数据库服务器。基于B/S体系结构的软件，系统安装、修改和维护全在

服务器端解决，系统维护成本低，客户端没有任何业务逻辑，用户仅需要一个浏览器就可以运行全部的应用模块，所以具有良好的灵活性和可扩展性，对于环境和应用条件经常变动的情况，只要对业务逻辑层实施相应的改变，就能够达到目的。在B/S架构的系统中，客户应用程序不能直接访问数据，具有较好的安全性，三层模式是真正意义上的瘦客户端。客户浏览器在每一次请求服务器就要刷新一次页面，受限于HTTP协议基于文本的数据交换的限制，在数据查询等响应速度上远远低于C/S体系结构。数据提交以页面为单位，数据的动态交互性不强，不利于在线事务处理应用。

表示分离风格：

表示分离包括：MVC风格、MVP风格

MVC风格经用户和应用之间的交互分为三个角色：模型(业务逻辑)，视图(用户接口)，和控制器(用户输入)。其构件连和连接件分别为：模型、视图、控制器和显式调用、隐式调用或其他调用机制。MVC通过以下机制来实现：1、将核心功能从UI行为中分离出来；2、建立改变传播机制；3、设计和实现视图；4、设计并构建控制器；5、建立视图和控制器之间的关系；6、启动MVC。

MVP的基本思想是一种“变种MVC”，视图吸收控制器功 能，增加一个主持者层(Presenter)，主持者P能够直接 访问视图和模型，模型-视图之间的关系仍然存在，但视 图并不直接使用模型，所有的交互都发生在Presenter内部。模型与视图之间的交互类型，他把用户行为分类 为选择、执行命令和触发事件。

MVC组主要的优点是数据从视图和控制器中解耦出来、三个模块相互独立，改变其中一个不会影响其他两个，控制器可以用来连接不同的模型和视图去完成用户的需求。但这样会导致需要构建并维护更多的构件，增加了系统的复杂度。魔性的改变也会同时需要控制器和视图的改变。由于试图和模型数据的分离，可能会产生低效的数据访问。

SOA风格：

SOA系统把系统按照实际业务，拆分成刚刚好大小的、合适的、独立部署的模块，每个模块之间相互独立。SOA能达到的目的：1.实现业务与IT的一致性；2. 创建更灵活的反应更敏捷的IT基础设施；3. 简化集成实现；

面向服务体系结构是开发分布式应用软件的新型体系结构，将应用程序的不同功能单元描述为服务，这些服务通过接口联系起来。这些接口独立于实现服务的硬件平台、操作系统和 编程语言，这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。同传统的面向对象的模型相比较，面向服务的模型是松耦合、粗粒度和无状态的模型。

Restful风格：

主要用于客户端和服务器交互类的软件。基于restful风格设计的软件可以更简洁、更有层次、更易于实现缓存等机制。REST 指的是一组架构约束条件和原则。满足这些约束条件和原则的应用程序或设计就是 RESTful。使用一个URL地址表示一个唯一的资源，然后把原来的请求参数加入到请求次元地址中，原来请求的增，删，改，查操作，改为使用HTTP协议中请求方式GET、POST、PUT、DELETE表示。优点：1.适合开放性高的API。这几年的由于移动互联网流行使得前端设备多样化，业界急需一种统一的机制来规范API设计，使得API适用于各种各样的前端设备，REST符合这种需求。2.行为和资源分离，更容易理解。3.提出使用版本号（例如v1、v2），更加规范。

# 三、关于这门课程，你做了什么？有什么收获？

对曾做的管理系统（C语言程序）重新分析，理清其中的函数调用关系，画出函数调用图，简单的优化函数的结构。在面向对象风格的语言中重新实现管理系统中的功能。

比较不同的软件架构风格，调试分析KWIC。对于抽象数据类型（Abstract Data Type）、黑板风格（Blackboard Solution）、隐式调用风格（Implicit Invocation Solution）、Main\_Subroutine、管道过滤器风格（Pipe\_and\_Filter Solution）。画出每种风格对应的静态结构图。对于每个程序，分别输入1500行左右的纯文本数据，测试每一种风格程序的性能。

通过petclinic案例，了解Spring framework和Spring boot项目的运行原理和二者之间的差别和联系。导入数据库，连接数据库，配置tomcat，通过本地网络用不同设备访问本地主机。

# 四、大学期间你开发过哪些软件系统？请分别简单介绍。

### 1:小区管理系统

使用面向过程语言编写。该软件模拟疫情下小区居民的出入大门管理，并记录居民的住址信息和健康状态。居民可以通过软件提交外出申请或取消申请，修改信息。管理人员通过该软件对居民提出的申请进行审核。也可以查看用户信息和登记体温等健康状况。

### 2:学生管理系统

使用面向对象语言编写。记录的信息包括学生姓名、学号、性别等个人信息，和学科，单科成绩。该系统可以对所有的信息进行修改，也可以增加或删除一条学生记录，但必须保证所有输入的数据符合定义的数据类型和值域，系统拥有检查输入数据的功能。

# 五、选择一个你开发过的软件系统，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。

小区管理系统：

软件结构：

一、小区居民端：在使用账号密码登陆（函数type1）成功后，可以分别进行登记外出、上报体温、修改个人信息等操作。

二、医护人员端：同样需要登陆（函数type2），进行居民日常健康检查、对登记外出的居民进行测量体温、定时自测体温等操作。这些数据上传之后系统会自动判断是否符合健康标准并给异常的用户打上标签。

三、管理员端：（type3）身份验证成功之后，可以进行批量查看用户信息，审核批准用户外出申请的操作。

软件质量属性：安全性、三类用户的操作所涉及的函数完全分离，不会相互影响操作；易用性、系统中包含的操作较少，并且如果一旦发生错误会立即返回上一步并发出错误警告，即使是初次使用也可在提示下完成操作；持续可用性、在每一次完成操作后，会直接返回主函数或主函数直接调用的子函数，并释放掉后面调用的子函数所占用的内存，回到系统刚启动时的状态，因此系统可以长时间稳定运行；可伸缩性、系统所使用的记录数据的文件并无标准、无法进行拓展，可伸缩性较差；互操作性、在本系统开始运行并调用某个子函数时，子函数所使用的记录文件会被锁定，无法与其他系统进行交互；可靠性、每次运行完一个事件就会完全释放内存，理论上在一定时间内程序不会出现故障。

软件体系风格：主程序/子程序层次结构的调用返回风格，自上而下的调用方法。

# 六、选择一个开源软件系统，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。

Android系统是采用“堆层”的体系结构来搭建整个系统的运行流程的，整个系统可以分为Android应用程序层、Android应用程序框架层、外部库和运行时库层及Linux内核层。1·应用程序层主要体现在用户界面方面，通常以Java或Kotlin语言进行编写，资源文件放置在项目目录res中。2·外部库和运行时库相当于一般嵌入式系统的中间件层次，在Android系统中分为库和系统运行环境。3·框架层为应用程序开发者提供了大量API，实际上即为应用程序的框架。包含UI程序中需要的各种空间，例如：Views(视图组件)，包括Lists(列表)、Grids(网格)、TextBox(文本框)、Buttons(按钮)等控件，甚至包含有一个重要的嵌入式的网络浏览器(webkit内核集成)。4·Linux内核，硬件驱动程序模块。

软件质量属性：安全性、Android系统对应用的权限有严格限制，对危险操作可以及时制止；易用性、拥有图形化界面和流畅的动画，用户的操作直接反馈到主屏幕上，让用户了解每一步操作带来的结果，并且可以不断试错；持续可用性、在Activity成为后台程序时，系统会限制其活跃度，足够长的时间之后甚至会结束该Activity，直至用户重新选择该界面系统会恢复期活跃度或者重启Activity，以此来保持系统有足够的内存空间来保持长时间稳点运行；可伸缩性、为开发者提供了大量的接口；互操作性、Android系统开放程度非常高，可以和其他系统交互；可靠性、在短期内不会出现系统崩溃。

软件体系风格：分层系统。

# 七、选择一个你使用过的非开源软件，从软件结构、软件质量属性、软件体系结构风格三个维度进行描述。

QQ主要分为登录注册,消息,聊天,联系人,动态,侧边栏,设置等几大模块。

软件质量属性：安全性、QQ使用密码，手机号辅助验证，手机令牌等手段阻止非授权用户登陆账号，服务器有腾讯开发的防火墙进行保护；易用性、拥有图形界面，将常用的功能设置为导航栏，简单易懂；持续可用性、作为及时通信社交软件，拥有24小时不间断在前台或后台运行的能力；可伸缩性、运行时使用的数据库可以通过其他应用进行访问，可延展性强；互操作性、可以从其他系统接收数据，也可将数据分享到其他系统；可靠性、可以长时间运行。

软件体系风格：事件驱动风格