### 1.引言

**1.1目的及范围**

通过本项目的系统分析，可以深入了解软件系统的开发内容和开发过程。将所学的框架开发技术运用到本项目中去，同时开发过程要求运用软件工程方法和Case工具。

**1.2 文档结构**

第一部分 引言

第二部分 架构背景

第三部分 视图

第四部分 视图之间的关系

第五部分 软件体系结构学习报告

第六部分 项目故障树及相应割集树

### 2.架构背景

**2.1系统概述**

电影订票系统是便于电影院和购买者之间交易的网络平台系统。实现客户浏览上映电影，搜索上映电影；客户注册会员，修改密码，会员修改信息、购买电影票、销售排行（最受欢迎排行榜）。

开发软件名称：电影订票系统

项目的提出者：各大电影院

开发方名称：。。。。

开发者：。。。。。 小组

开发负责人：张树祥

用户：会员、管理人员、游客

**2.2架构需求**

2.2.1 技术需求

前端采用的技术栈为webpack+vue全家桶：

webpack：项目构建打包

vue：一个类MVVM的渐进式JavaScript框架

vue-router：单页应用前端路由

vue-material：UI框架

vuex：应用数据状态管理

服务器端的技术栈为Flask+MySQL+Redis+Gunicorn+Nginx：

Flask：轻量级 Web 应用框架

MySQL：最流行的关系型数据库管理系统

Redis：基于内存亦可持久化的Key-Value数据库

Gunicorn：轻量高效的Python WSGI Server

Nginx：高性能的负载均衡服务器

2.2.2 功能需求

（1）系统中有多个电影院，系统管理员进行电影院的维护

（2）系统管理员可以为每个电影院指派一个电影院管理员

（3）电影院管理员可以维护本电影院即将上映的电影信息

（4）网民可以进行用户注册和登录

（5）用户可以根据时间、电影名称、电影院名称进行电影查询，购买积分，使用积分购买电影票

（6）系统能够对指定时间上映电影进行统计分析，显示最受欢迎的电影排行榜在页面中

2.2.3 质量属性需求

(1)数据正确性：

结合现实情况：购买积分使用浮点数存储，保证充值金额正确记录，每次充值判断是否积分达到上限，达到上限系统返回充值失败。

对于多用户并发购买电影票，系统的电影票剩余量修改操作保证原子性，电影票剩余量修改操作同一时刻只能执行一个事务。通过锁机制实现修改函数的互斥使用。

通过数据库备份保证当主数据库存储出现问题的时候，可以通过使用备份数据库进行服务，提高用户的使用体验，增加数据正确检验，通过检查多份数据库中的数据避免因为一个数据库的错误信息导致的数据错误，提供一定的纠正空间。

（2）性能：

性能属性描述了系统速度和容量上的特点，包含以下几项内容：

（1）响应时间：软件对请求做出反应的速度；

（2）吞吐量：软件每分钟可以处理多少请求；

（3）负载：在响应时间和吞吐量变糟糕之前，软件可以支持多少用户使用。

为了保证软件的性能，我们选择的第一个方法是提高资源的利用率，比如，一个购票系统可以同时收集多个用户的业务请求，鉴别用户身份信息，以及向客户确认请求的业务，这样，服务器只接受经过确认和鉴定后的请求，然后处理，提高了服务器的吞吐量。

第二个方法是有效的管理资源分配，我们根据具有的高低优先级来处理请求，保证了重要请求能被快速处理，同时，动态地增加被搁置的请求的优先级，保证一旦某个请求的服务开始，那么它的处理就一定会全部完成。

（3）易使用性：

易理解性：

本项目分为网页、服务器、数据库三部分。结构清晰明了，功能、逻辑、概念、接口没有歧义。文档语言简练，前后一致，易于理解。逻辑层次上清晰直白，描述细致，充分考虑普通用户的使用体验。

易学习性：

网页界面一目了然，用户进入界面后能够轻松明确网页的各个功能，方便完成操作。每一步操作步骤细致，对操作处理错误的处理也十分明确。

易操作性：

本系统的人机界面友好、设计科学合理，操作简单。各个业务可以互相关联其他相关业务的数据，页面事先预定设定合理的默认值，避免了手工操作失误可能产生的严重后果。如果操作失败本系统可以及时给出反馈和错误处理。

（4）安全性：

猿眼电影订票系统采用B/S架构，客户端软件主要是Web页面。

**准则**

1)   确保最薄弱环节的安全

2)   纵深防御,避免单点失效

3)   失效安全：进入安全状态、阻止信息非授权访问

4)   最小化准则：范围最小化、权限最小化

5)   通过冗余和多样性降低风险

6)   验证所有输入

**安全性可分为: 数据或功能访问控制安全和页面访问控制安全。**

**数据或功能访问控制安全**

数据或功能访问安全性可保证在预期的安全性情况下，不同授权的用户只能访问特定的功能，或者只能访问有限的数据。例如，普通用户可进行输入数据、创建新账户等操作而不能随意删除数据或账户，只有管理员才能删除数据或账户。

**页面访问控制安全**

(1)页面登录。B/S 架构软件必须测试登录用户名和密码的有效性、输入大小写的敏感性、用户登录是否有次数限制; 而且要测试是否可以不登陆而直接浏览某个页面、IP地址登录是否有限制等;

(2) 超时限制。如果用户登陆系统后在一定时间 内没有进行任何页面操作，应进行超时判断，强制用户 重新登陆后才能正常使用系统。

(3) 日志文件。日志文件是保证 Web 应用系统安全性的重要的工具。B/S 架构软件需要测试日志文件 记录信息的完整性、各类操作的可追踪性。在服务器后台，还要检测服务器的日志记录是否正常进行。

(4) SSL。为保证信息在Internet 上传输的安全性，B/S架构软件会采用SSL技术。如果使用了SSL，测试人员需要测试加密的正确性、检测信息的完整性，确定是否有相应的替代页面。当用户进入或离开安全 站点的时候，是否有相应的提示信息。

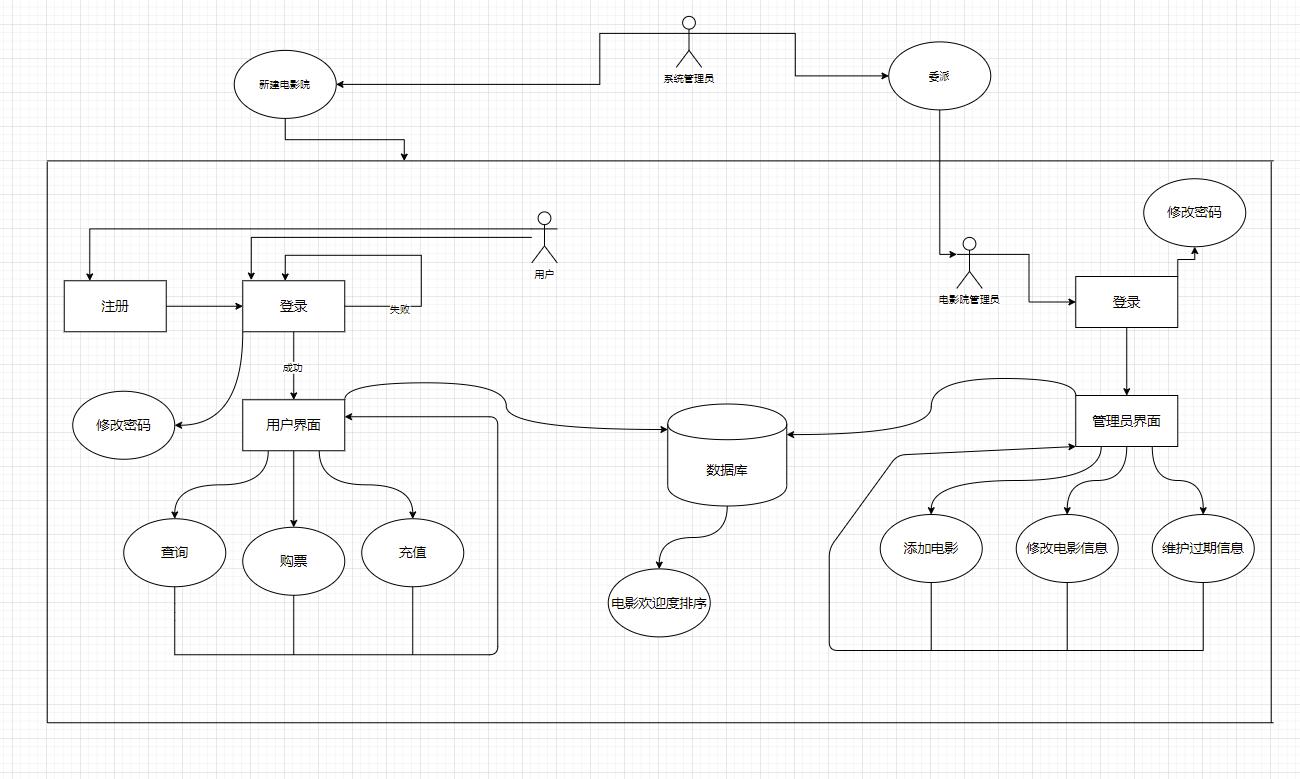
**安全性测试技术手段**

为确保数据访问安全而进行的**安全功能测试**和为保证页面访问安全而进行的**安全漏洞测试**。安全功能测试基于软件的安全功能需求说明，测试软件的安全功能实现是否与安全需求一致; 安全漏洞测试则站在攻击者的角度，以发现软件的安全漏洞为目的。

### 3.视图

**3.1 逻辑视图**

**3.1.1主表示**



**3.1.2构件说明**

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（Problem Definition)。 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（Class Diagram）来描述逻辑视图。

电影订票系统逻辑视图包括：系统管理员，电影院管理员，用户，数据库四个部分。

系统管理员：可以建立电影院，以及设立及指派每个电影院的电影管理员。

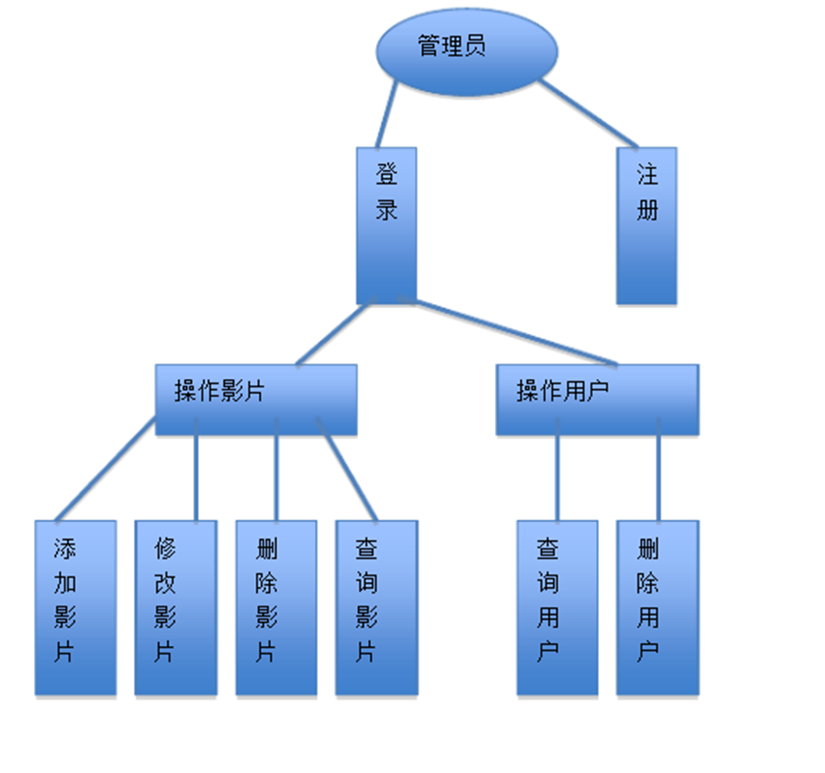
电影管理员：通过系统管理员获取管理员账号，电影管理员可以修改密码，登录后通过管理员界面对管理的电影院添加电影，修改电影信息，维护电影信息等。

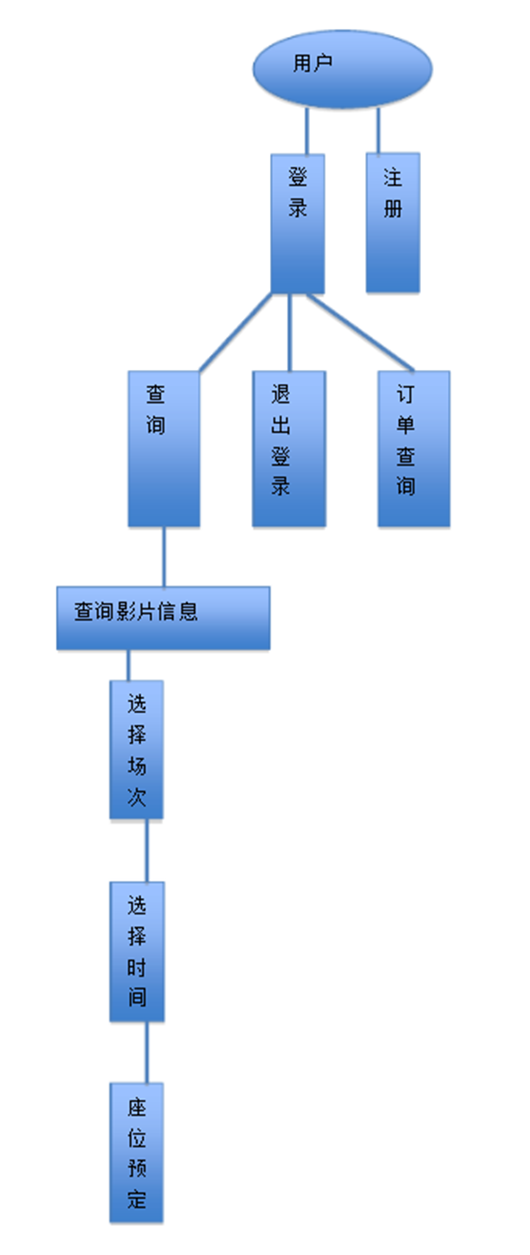
用户：游客通过注册成为系统的用户，用户可以修改自己账号密码，同时可以通过用户界面查询和购买电影票，用户购买电影票需要通过充值的积分来进行购买。

数据库：数据库存储用户和电影院管理员和系统管理员的账号和登录密码，存储用户的个人信息，上映电影票信息以及用户的订单信息和充值积分。系统还能通过自动检索用户订票信息展示一定时间内的电影欢迎度排行榜，为用户提供一定参考。

**3.2 开发视图**

**3.2.1主表示**

****

****

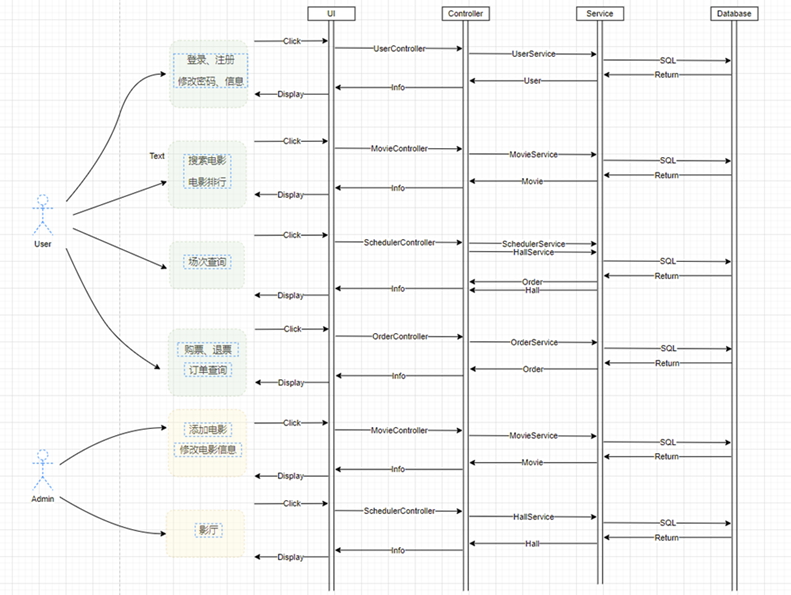
**3.2.2构件说明**

用户：用户只有在登录之后才能进行其他一系列操作，且在登录前必须先进行注册，之后便可以查询电影的放映信息，再通过选择合适的场次和时间，完成预定。除此之外，用户还可以查询自己的订单和历史消费。

管理员：管理员可以对系统进行的操作主要分为操作影片和操作用户两类，管理员添加、修改、删除、查询所有的电影，从而对系统进行及时的维护和跟新；还可以查询和删除用户，便于对用户信息进行管理

**3.3 运行视图**

**3.3.1主表示**

****

**3.3.2构件说明**

UserController：UserController构件向UI界面提供了登录、注册、修改密码、更新信息等接口收到请求后调用UserService提供的接口来实现相应的功能。

MovieController：MovieController构件负责的是电影管理模块，为用户提供了诸如电影搜索、排行

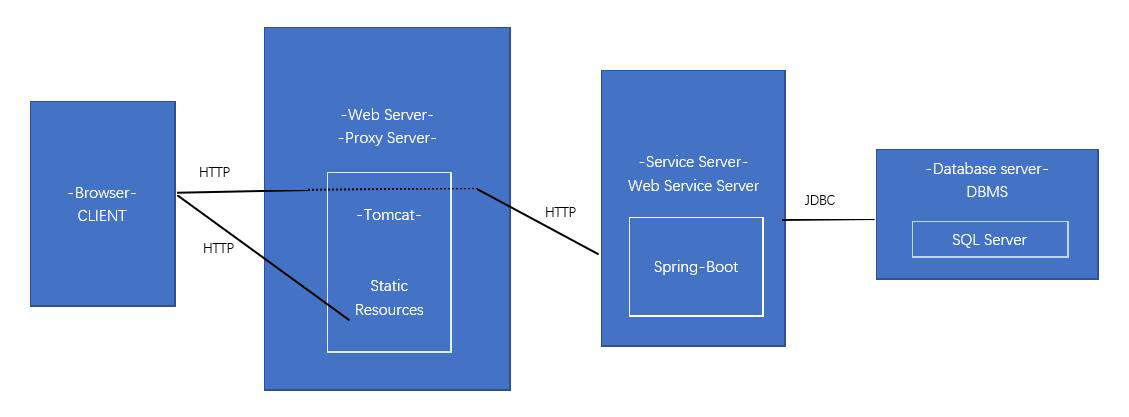
OrderController：OrderController构件主要完成的是订单管理的功能，例如查看订单、支付等等、依赖于 OrderService、MovieService 、ScheduleService等构件。

ScheduleController：主要负责的是电影上映场次的管理功能。提供了通过名称、状态、时间来查找场次以及增加场次的主要功能，主要通过调用ScheduleService、HallService构件的结构来实现。

UserService、MovieService、OrderService、HallService、ScheduleService这五个构件向上为相应的Controller提供相应的接口，以供其调用从而完成相应的行为，向下将自己提供的接口对应的功能转化为相应的SQL语句，从而转变为在数据库上的操作，对数据库进行增删改查，对数据库中的信息进行读写。

**3.4 部署视图**

**3.4.1主表示**



**3.4.2构件说明**

Web Server 服务器：一台Web服务器预装操作系统，用来部署Web。

Web访问量分流设备：根据网站流量，自动定位客户访问流量小的服务器。

Proxy Server服务器：介于浏览器和Web服务器之间的一台服务器，当你通过代理服务器上网浏览时，浏览器不是直接到Web服务器去取回网页，而是向代理服务器发出请求，由代理服务器来取回浏览器所需要的信息，并传送给你的浏览器。可提高访问速度，增强安全性。

DB Server服务器：数据库服务器。

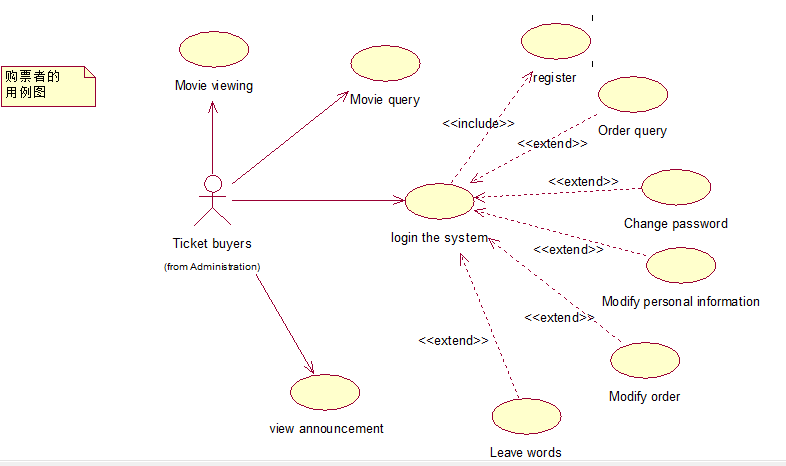
User Client：用户个人PC，使用安装在它之上的浏览器即可。

Service Server：SpringBoot内置服务器，并装备启动类代码，可以快速开启一个Web容器。

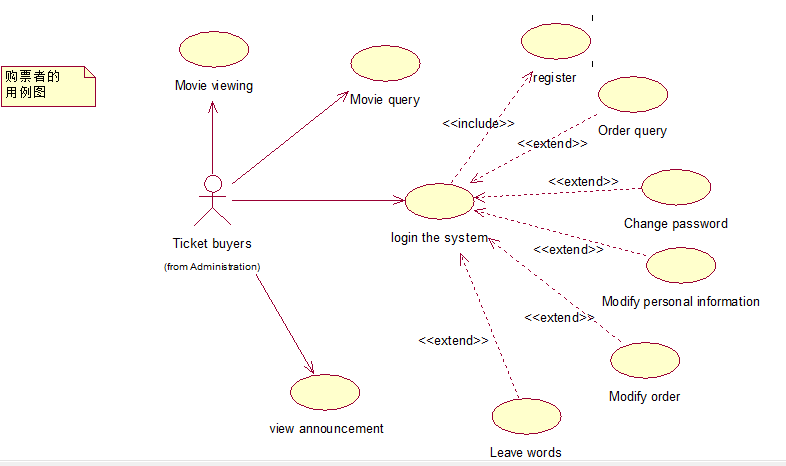
**3.5 用例视图**

**3.5.1主表示**

该部分是对软件架构的用例视图的描述，重点关注的是软件功能性需求。



购票者的用例图



管理员用例图

**3.5.2构件说明**

**购票者：**

**Login the system (登录系统)**  Ticket buyers（购票者）只有在登录系统之后才可以进行其他的工作，而且登录和注册是必须住册之后才可以登录的。

**Movie viewing（电影浏览）** 购票者可以在本系统中，浏览所有发布的电影集。

**Movie query（电影查询）**购票者可以在本系统中，查询自己想要看得电影，而且还可以按照自己喜欢的电影类型进行查询。

**Register（注册）** 只有先注册之后才可以登录系统进行进一步的操作。

**Order query （订单查询）** 购票者在登录系统之后，选择自己想要观看而且在那个时间段有播的电影进行预定，用户确定购买之后就可以查询自己的订单。

**Change password （修改密码）** 购票者在登录系统之后，如果想要修改自己密码，可以通过这个进行密码的修改。

**Modify personal information （修改个人信息）** 购票者在登录系统之后，如果个人信息有写错的可以进行修改。

**Modify order （修改订单）** 购票者在登录系统之后，对自己购买的电影票有想要修改的，可以通过点击修改订单，增加电影票或换个电影等等一系列操作。

**View announment（查看公告）** 购票者在系统的主页面， 查看管理员发布的最新的电影活动或者最新上映的电影等等。

**Leave words（留言）**购票者在登录系统之后，登录成功后，可以进入到网站的留言板块进行留言等。

**管理员：**

系统管理员（Administration）可以对系统进行的操作：

1. Increase the film（增加电影）管理员登录就可以增加新出的电影。
2. Remove the film（删除电影）管理员登录就可以删除那些已经过时的电影，来增加一些新片
3. Delete user（删除用户）管理员登录就可以删除一些用户注册的错误的用户或一些经常不登录的用户。
4. Delete movie ordering information（删除电影预定信息）管理员登录就可以删除那些用户订购但最终取消的订单。
5. Announcement（公告）管理员登录就可以发布最新的电影信息或一些新出的电影套餐，方便购票者更快捷的进行选择。

### 4.视图之间关系

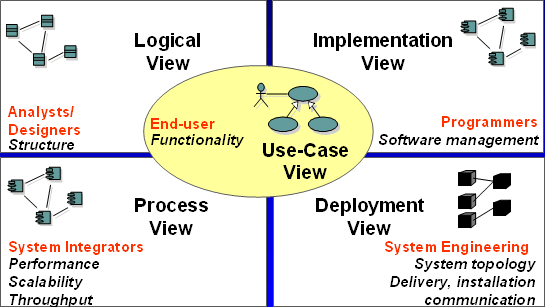
视图之间关系说明：

通过4+1视图，我们可以形成一个系统的抽象描述，组织中的所有成员，都要围绕着这个抽象进行设计、实现、验证，并在系统演进中不断完善修正它们。由于4+1视图并不能覆盖系统的所有内容，比较系统中使用了哪些关键的技术，或是系统中关键的数据结构和表的设计，在这里都没能体现出来。这需要我们再结合其它的设计技术来完善补充，才能让我们的系统设计更加完整。

用例视图即4+1中的1，其余4个视图都是围绕着用例视图为核心的。用例视图作为4+1视图的核心，确定了系统边界，系统用户以及系统的功能和场景，而其它4个视图都是需要围绕着这些信息进行设计。

通过逻辑视图、开发视图加部署视图，我们可以知道系统中每一个逻辑架构元素、每一份代码，最终会运行在什么位置上。反向也可以通过运行环境上，找到所有其上运行的逻辑架构元素和代码。它们描述的都是系统的静态信息。

运行视图则是用来描述系统中的动态信息的，在运行视图中，除了要关注组件间的交互关系，通常还需要考虑并发、抢占、关键资源（比如锁）访问等。



### 5. 软件体系结构学习报告

**5.1 张树祥**

什么是软件体系结构

软件体系结构的定义：是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件、和连接构件。

处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。

（1）从工程的角度来看：

软件体系结构是软件设计过程中的一个层次，这一层次超越极端及过程中的算法设计和数据结构设计。软件体系结构处理算法与数据结构之上关于整体系统结构设计和描述方面的一些问题

（2）从审视角度看，软件体系结构有四个角度：

概念角度：描述系统的主要构件以及它们之间的关系

模块角度：包含功能分解与层次结构

运行角度：描述了一个系统的动态结构

代码角度：描述了各种代码和库函数在开发环境中的组织

（3）从重用的角度看：

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、结构和关系。

（4）从维护角度看：

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演化的指导方针。

（5）从工程的作用看：

软件体系结构包括一个软件和系统构件、互联及约束的集合；一个系统需求说明的集合；一个基本原理用以说明这一构件、互联和约束能够满足条件。

（6）从质量角度看：

软件体系结构包括一个或一组软件构件、软件构件的外部的可见特性及其相互关系。

**5.2 刘永豪**

体系结构发展

一个好的架构能确保软件满足诸如性能、可移植性、可扩展性和可互操作性等关键需求。人们逐渐认识到一个合理的架构是系统设计、开发成功的关键。当前软件体系结构的研究有了很大进展，一个工程的软件架构的基本轮廓逐渐变得清晰，但仍然面临许多挑战和未知。

过去，体系结构的描述大多依赖于非形式化的箱线图，软件体系结构北非一个正式环节，人们逐渐意识到需要一个更严格的方法，于是号召架构设计作为一个需要关注的领域，进而出现了软件架构师。行业有两股潮流，一是认识到在构建复杂软件系统时对方法、技术、模式的共享。设计师用抽象来使整个复杂系统便于理解。第二是对在特定领域利用共性来提供一系列产品的重用框架的关注。这种想法基于相关系统的共同方面可以被提取出来，通过实例化共享的设计来使每个新系统的成本相对较低。

如今，软件开发中体系结构设计越来越重要和明确化，体系结构设计的技术基础已经大为改善，三个重要进步是：（1）体系结构描述语言和工具的发展：非形式化的图不能用于一致性、完整性或正确性分析。对此，人们提出用形式化的符号（正规的表示法）来表示和分析体系结构设计，就是常说的体系结构描述语言（ADLs），典型的有UniCon，Darwin，Rapide，C2，SADL，Wright。（2）产品流水线生产：一个产品线代表一组具有公共的系统需求集的软件系统，将可重用构件与系统独有的部分集成而得到。（3）架构标准的产生：体系结构设计的书籍的出版及有关课程的开设。这些书籍和课程的一个共同主题是使用标准体系结构风格。

软件体系结构如今有了更坚实的基础，但它尚未作为一门学科在整个软件行业普及。该领域的自然发展将导致稳定的进步。网络为中心的计算对软件体系结构有很大影响，在网络无所不在的环境中，系统可能不会集中控制，这样一组新的软件体系结构挑战出现。首先是架构的需要，适应网络的大小和变化。虽然许多相同的架构模式可能会运用，其实现的细节和规范将需要改变。

**5.3 王杰**

部分软件系统结构及其特点

软件体系结构在软件工程领域有着广泛的影响，但当前仍未形成一个统一的、标准的定义。目前国内外普遍认可的看法是软件体系结构包含构件、连接件和约束。其中约束描述了体系结构配置和拓扑的要求，确定了体系结构的构件与连接件的连接关系。这样就可以把软件体系结构写成

软件体系结构(software architecture)=构件(components)+  
　　连接件(connectors)+约束(constraints)  
　　构件是软件体系结构的基本元素之一。一般认为，构件是指具有一定功能、可明确辨识的软件单位，并且具备语义完整、语法正确、有可重用价值的特点，然而目前对于构件的具体结构及构成并没有一个统一的标准，而且一些主要的构件技术也没有使用相同的构件类型。另外，当前被广泛接受的构件定义并不包含具体的软件构件模型(software component model)。例如，Szyperski等人给出了软件构件一个很有名的定义:软件构件是一个仅带特定契约接口和显式语境依赖的结构单位，它可以独立部署，易于第三方整合。但是关于软件构件模型有一个被普遍接受的观点是:软件构件是一个具有服务提供和服务请求功能的软件单元。  
　　连接件是软件体系结构另一个基本的构成元素，是用来建立构件间交互以及支配这些交互规则的构造模块。连接件最先是由Shaw提出来的，她建议把连接件作为软件体系结构中第一类实体，用来表示普通构件之间的交互关系。目前对于连接件尚未形成统一的认识，尽管在软件体系结构中强调了连接件存在的必要性，但是关于连接件模型的研究还很少，连接件的实际应用还不成熟。  
　　面向方面软件体系结构在传统软件体系结构的基础上增加了方面构件单元。通常认为，方面构件是封装了系统横切关注点的一类特殊的构件。目前关于方面构件模型的研究还处于起步阶段。  
　　由于传统软件体系结构模型包含构件、连接件和约束，而面向方面软件体系结构是在传统软件体系结构的基础之上扩展了方面构件，所以面向方面软件体系模型结构包含构件、连接件、方面构件和约束。其中约束描述了面向方面体系结构配置和拓扑的要求，确定了体系结构的构件、连接件和方面构件之间的连接关系，而构件、连接件、方面构件是它的三个基本的构成单元。以下对这三个构成单元的模型进行详细的设计

**5.4 王英凡**

软件体系结构的内容

软件体系结构是具有一定形式的结构化元素，即构件的集合，包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工，数据构件是被加工的信息，连接构件把体系结构的不同部分组合连接起来。当下为我们所熟知的是三层结构。

客户层：用户接口和用户请求的发出地，典型应用是网络浏览器和胖客户（如Java程序）。

服务器层：典型应用是Web服务器和运行业务代码的应用程序服务器。

数据层：典型应用是关系型数据库和其他后端（back-end）数据资源, 如 Oracle和SAP、 R/3等。

三层体系结构中，客户（请求信息）、程序（处理请求）和数据（被操作）被物理地隔离。三层结构是个更灵活的体系结构，它把显示逻辑从业务逻辑中分离出来，这就意味着业务代码是独立的，可以不关心怎样显示和在哪里显示。业务逻辑层处于中间层，不需要关心由哪种类型的客户来显示数据，也可以与后端系统保持相对独立性，有利于系统扩展。三层结构具有更好的移植性，可以跨不同类型的平台工作，允许用户请求在多个服务器间进行负载平衡。三层结构中安全性也更易于实现，因为应用程序已经同客户隔离。应用程序服务器是三层/多层体系结构的组成部分，应用程序服务器位于中间层。

**5.5 唐灿**

**Definition of *Software Architecture***

网上查阅资料，搜到了[这篇文档(Software Architecture Guide)](https://martinfowler.com/architecture/)，文档作者对软件架构说明了 3 个定义：

* a better view of architecture was **the shared understanding that the expert developers have of the system design.**
* “the design decisions that need to be made early in a project”, but Ralph complained about this too, saying that it was more like **the decisions you wish you could get right early in a project**.
* His conclusion was that **“Architecture is about the important stuff. Whatever that is”**.

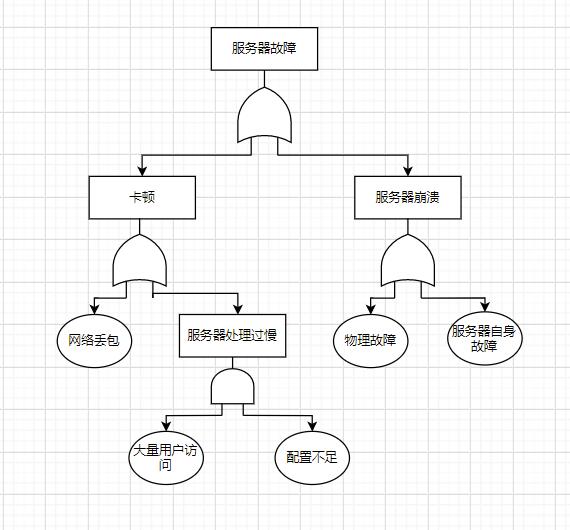
也就是说软件架构没有太明确的定义，有的人认为是专家对整个系统的设计，也有观点认为就是整个项目开始前这些设计理念的设计者认为的该有的“打开方式”，结论就是：不管是啥，就是跟重要的事情有关的东西。

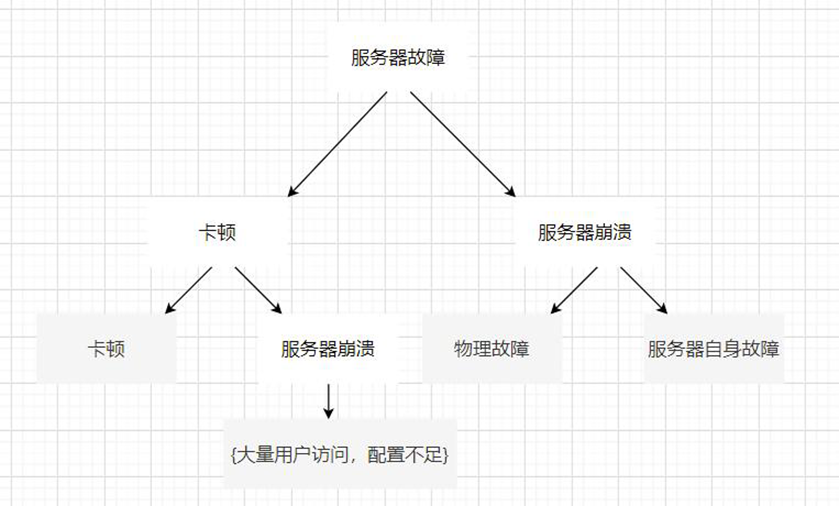
我也就理解为 something important you should decide and design before the whole project, whatever comes to your mind.

在文档下面有个关于 [GUI Architectures](https://martinfowler.com/eaaDev/uiArchs.html) 的链接，虽然说 2006 年的可能有点老，毕竟 User Interface 的设计和审美随时间会不断变换，但是这篇文章不讲美观，只讲架构，还是值得深入学习的。

### 6. 项目故障树及相应割集树

**6.1**





**6.2**

