软件体系结构设计(SAD)

——人事管理系统的软件体系结构设计

组员：杨逸，黄德业，刘钊，谢骁飏，闫威

指导老师：余仲星

目录

1 引言

1.1 标识

标识号：PMS-2303

软件名称：人事管理系统

缩略词语：personnel management system（PMS）

版本号：v1.1.1

发行号：20230314

1.2 系统概述

本项目是一个基于软件工程原理和技术开发的人事管理系统，旨在为企业或机构提供一种高效、便捷的员工信息管理、工资管理和数据分析平台。系统将采用先进的技术和架构，以提高系统的稳定性、可靠性和安全性。

该系统的主要功能包括员工信息管理、工资管理和数据分析三大模块。员工信息管理模块包括员工基本信息的录入、修改、查询、统计和打印等功能；工资管理模块包括工资、奖金、扣款等信息的管理和网上查看个人工资条等功能；数据分析模块可以按照各种条件查询和统计员工信息，以支持企业或机构的管理决策。

此外，系统还将具备数据备份、恢复、权限管理等常用功能，以保障系统的可靠性和安全性。

1.3 文档概述

软件体系结构设计（SAD）概述提供了软件系统设计的高层次视图，包括系统架构、模块设计、接口设计和数据结构设计等内容。它是在需求分析和系统设计的基础上编写的，并为软件开发人员、测试人员和维护人员提供了清晰、详细和可执行的软件系统设计文档。

软件体系结构通常包含以下内容:

1.系统架构设计：描述系统的整体架构、模块划分、模块功能、模块之间的关系和数据流等。

2.模块设计：对系统中的各个模块进行设计和描述，包括模块的功能、接口、输入输出、数据结构、算法和程序流程等。

3.接口设计：描述模块之间的接口规范，包括接口类型、接口参数、接口传递的数据类型和格式等。

4.数据结构设计：描述系统中所使用的各种数据结构，包括数据结构的类型、结构定义、数据元素的类型和结构、数据的存储和访问方式等。

5.安全设计：描述系统的安全性设计，包括系统的保密性、完整性、可用性等。

6.测试设计：描述系统的测试策略和测试计划，包括测试方法、测试环境、测试数据和测试用例等。

7.部署设计: 描述系统的部署设计和实施方案，包括系统的安装、配置和部署等。

1.4 基线

设计基线是软件工程中的一种基础文档或参考线，它是软件开发过程中的一个重要里程碑。设计基线是在软件体系说明(SAD)和软件架构设计文档的基础上，建

立起来的软件设计方案，是软件开发过程中的一个基础文档或参考线，可以在后续的软件开发、测试、维护和升级过程中，作为设计方案的参考和指导，确保软件的一致性、可维护性和可扩展性。

①系统设计文档：该模块需要包括员工信息的增删改查功能，可以参考 SRS 中的员工管理需求进行模块划分和接口设计。具体的实现可以采用 MVC 架构模式，将模型层、视图层和控制层进行分离。

②数据库设计文档：该模块需要涉及员工信息的存储和管理，可以设计一个名为“employees”的表来存储员工信息。该表需要包括员工号、员工姓名、员工年龄、手机号码等字段。同时，为了提高查询效率，可以在姓名字段上建立索引。

③用户界面设计文档：该模块需要提供用户友好的订单管理界面，可以设计一个员工列表页面。员工列表页面需要展示员工的基本信息和当前状态，并提供搜索和过滤功能。员工详情页面需要展示员工的详细信息，同时提供编辑和删除功能。

④系统架构设计文档：该模块需要与其他模块进行协同工作，可以通过定义接口来实现模块之间的交互。同时，为了保证系统的可扩展性和可维护性，可以采用面向对象的设计原则，将系统划分为多个独立的模块，并采用设计模式来解决常见的软件设计问题。

⑤系统测试文档：该模块需要进行单元测试、集成测试和系统测试等多个阶段的

测试。单元测试需要测试每个模块的基本功能和边界条件，集成测试需要测试模块之间的接口和交互，系统测试需要测试整个系统的性能、稳定性和安全性。

2 引用文件

[1]杨丹,戴玉敏.基于计算机软件工程的数据库编程技术[J].电子技术与软件工程,2018(09):154

[2]钟睿.基于计算机软件工程的数据库编程技术[J].计算机产品与流通,2018(02):32.

[3][美]Shari Lawrence Pfleeger、[加]Joanne M·Atlee 软件工程(第4版·修订版)

3 CSCI 级设计决策

3.1 设计决策

设计决策：采用分层架构设计模式来组织人事管理系统的软件结构。

决策理由：分层架构是一种常用的设计模式，它将系统划分为多个层次，每个层次具有特定的职责和功能，以提高系统的可维护性、扩展性和重用性。

决策细节：

1.用户界面层：设计一个用户界面层，负责与用户进行交互，并展示系统的功能和数据。该层使用用户友好的界面元素和布局，以提供良好的用户体验。

2.应用逻辑层：设计一个应用逻辑层，处理用户界面传递的请求，并根据业务规则进行处理。该层包括业务逻辑、数据验证和处理用户输入等功能。

3.数据访问层：设计一个数据访问层，负责与数据库进行交互，包括数据的读取、写入和查询等操作。该层使用适当的数据访问技术，如 ORM（对象关系映射）

或 SQL 查询，以提供对数据的持久性访问。

4.数据库层：设计一个数据库层，用于存储和管理系统的数据。该层包括数据库的设计和建模，表结构定义以及数据关系的管理。

5.第三方集成层：如果需要与其他系统或服务进行集成，设计一个第三方集成层，负责处理与外部系统的通信和数据交换。该层可以使用适当的协议和接口，如 API（应用程序编程接口）或 Web 服务。

3.2 质量保证决策

1.决策：进行系统功能测试以确保其正确性和一致性。

2.决策理由：为了确保人事管理系统的功能符合用户需求并且工作正常，需要进行系统功能测试来验证系统的正确性和一致性。

3.决策细节：执行以下测试活动：单元测试：对各个模块和组件进行单独测试，以验证其功能和逻辑的正确性。

4.集成测试：将各个模块和组件组合起来进行测试，确保它们能够正确地协同工作。

5.系统测试：对整个人事管理系统进行综合测试，验证其功能、性能、可用性和安全性等方面的要求。

6.用户验收测试：由用户参与的测试活动，以确认系统是否满足其需求并能够实际应用。

7.异常情况测试：测试系统在异常情况下的行为和处理能力，例如输入错误、网络故障等。

8.性能测试：评估系统在不同负载和压力条件下的性能表现，包括响应时间、吞

吐量和并发用户数等指标。

3.3 工程管理决策

1.决策：采用敏捷开发方法进行项目管理和团队协作。

2.决策理由：敏捷开发方法可以提高开发团队的灵活性、效率和响应能力，适应需求变化和快速交付可用的软件产品。

3.决策细节：执行以下工程管理决策措施：敏捷开发方法：采用敏捷开发方法 Scrum 进行项目管理和团队协作。这种方法强调迭代开发、持续集成和团队合作，以快速适应变化和提供高质量的软件。

4.用户参与：积极地与用户进行沟通和合作，以确保对用户需求的准确理解和及时反馈。通过持续的用户参与和反馈，可以及时调整项目方向和优先级，确保交付符合用户期望的产品。

5.迭代开发：将开发过程划分为多个迭代周期，每个周期交付一部分可用的功能。每个迭代周期都包括需求分析、设计、开发、测试和部署等活动，以增量方式构建和完善系统。

6.持续集成：通过自动化测试和持续集成工具，确保团队成员的代码能够快速集成和测试，减少集成问题和软件缺陷，并提高团队的协作效率。

7.进度管理：使用项目管理工具和技术，如甘特图、看板和迭代计划等，对项目进度和任务分配进行监控和管理，确保项目按时交付和资源合理利用。

8.风险管理：识别、评估和管理项目中的风险，并制定相应的风险应对策略。持续跟踪和监控项目中的风险，并及时采取措施以降低风险对项目的影响。

9.团队协作：通过良好的沟通、协作和团队建设活动，促进团队成员之间的合作

和有效的知识共享，提高团队的效率和工作质量。

4 CSCI 体系结构设计

4.1 体系结构

4.1.1 程序(模块)划分

员工信息模块:

程序标识符：EMP

功能：这一部分主要完成与数据库的交互，基于 SQL 提供一些基本操作的接口，可以完成数据库中信息的查询、修改、插入和删除等，为后面的一些功能提供了基础。

源标准名：HR-XML

人事调动程序:

程序标识符：HRM

功能：可以利用员工信息模块提供的接口来完成人事调动，比如人事调动后修改相应员工的人事信息，新员工入职后在数据库为其创建对应的记录，以及员工离职后选择性地对信息进行删除或保留。

源标准名：HR-XML

薪资管理程序:

程序标识符：PAY

功能：根据员工信息模块的接口完成对员工薪资信息的修改，还可以根据员工的表现来进行奖惩，在薪资的基础上额外发奖金或罚款。

源标准名：ISO 20022

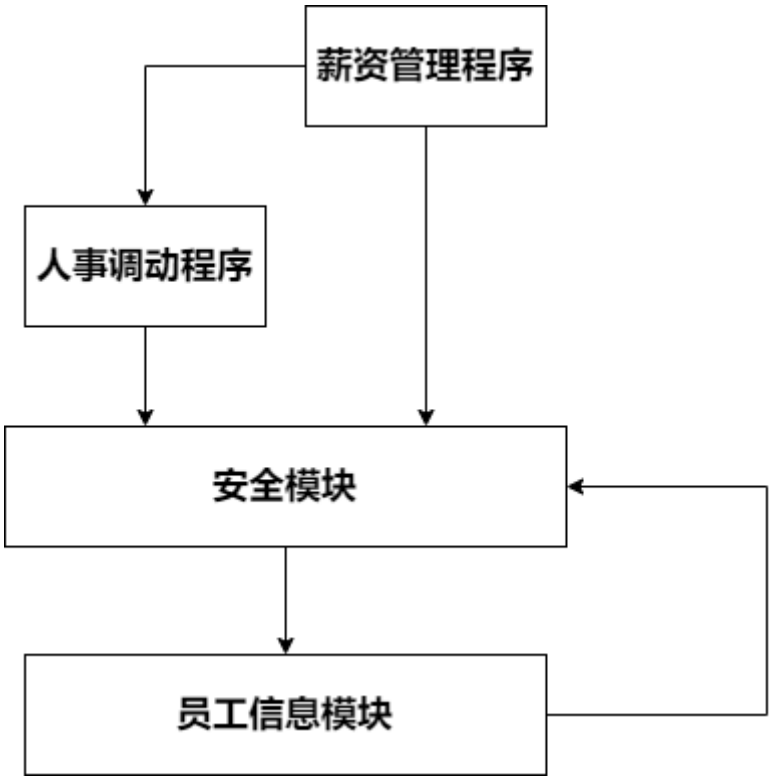
安全管理模块：

程序标识符：SEC

功能：这一模块主要是为了提高系统的安全性，会记录使用者近期的操作，这样如果出现了误操作，也可以及时的进行修改，并且该模块还会定期对数据库进行备份，避免出现数据库数据受损或丢失造成巨大损失的局面。

源标准名：LDAP

对应程序图如下：



4.1.2 程序(模块)层次结构关系

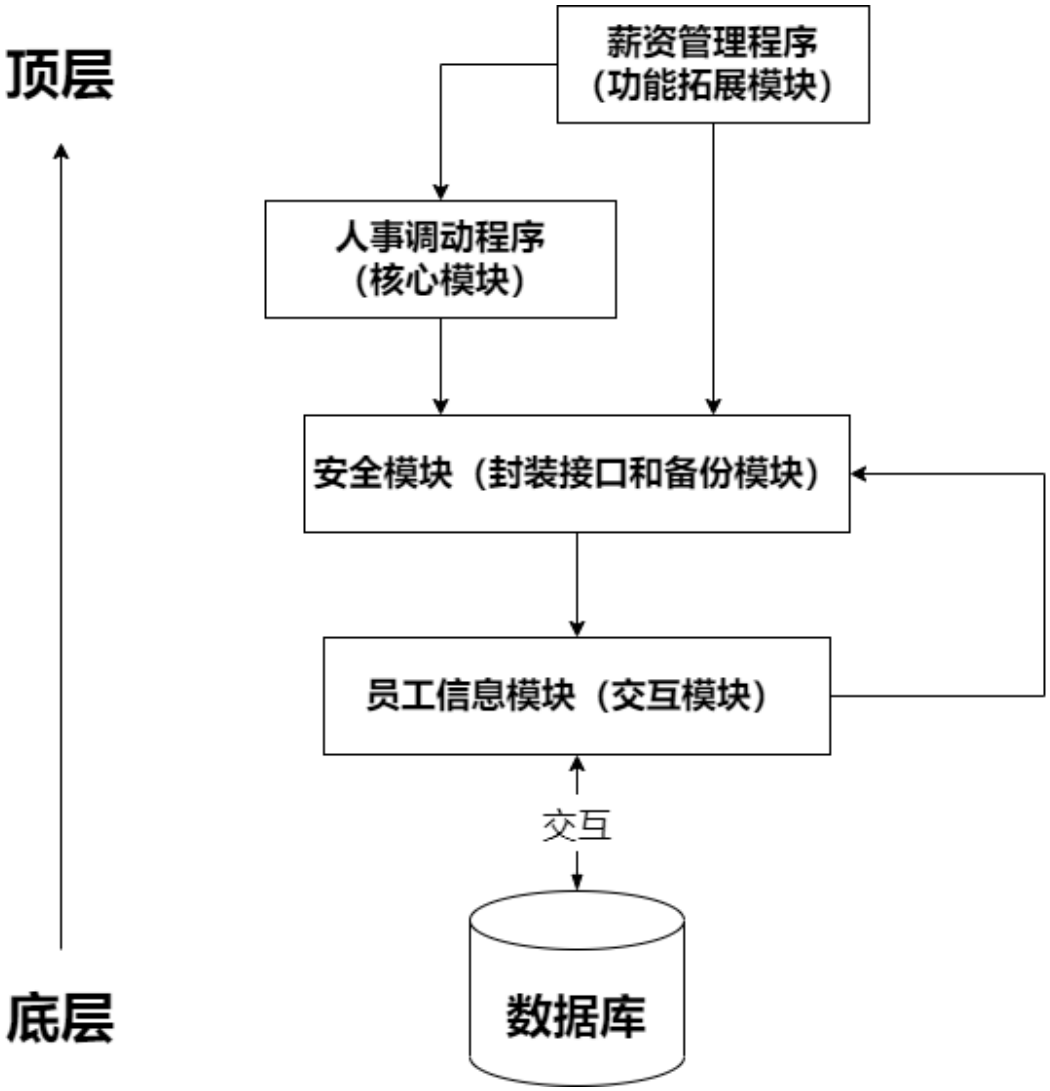
在模块的层次结构设计上:

员工信息模块:直接与数据库进行交互的部分,也是系统中最底层的部分,是人事管理系统的基础。

人事调动程序:是该系统的核心,系统设计的最主要的目的就是为公司的人事管理提供便利,该系统需要调用员工信息模块提供的接口,间接地对数据库进行操作,同时,该程序在运作期间也会调用安全模块,对使用者的操作进行保存,以确保误操作可以被回溯。

薪资管理程序:是系统的功能拓展部分,在人事管理的基础上对员工的表现、薪资信息等进行记录和管理,也需要使用员工信息模块提供的接口间接对数据库进行操作,并且同样地,也会调用安全模块确保系统的安全性。

安全管理模块：为了系统安全性而设立的模块，调用员工信息模块的接口，以实现定期对数据库信息的备份以及核验；同时在人事调动程序和薪资管理程序运行时都会被调用，对操作进行记录。



4.1.3 架构描述

基于上述的体系结构和调用关系，我们的系统有如下架构：

用户界面(User Interface)：这是系统的前端。用户将在此输入/查看他们的员工信息、发起调动请求、IP 登录等。

数据库(Database)：这是存储所有关于员工和公司的数据的地方。数据库设计需

要遵循最佳实践并考虑数据安全性和可扩展性。

员工信息模块(Employee Information Module)：这个模块允许管理员/HR 来管理员工相关的信息。这些信息包括员工基本信息、工作历史、培训记录、出勤记录、福利计划等。在这个模块中，管理员/HR 可以更新员工信息，如果员工变化工作职位或部门，管理员/HR 还可以使用这个模块进行调动员工的工作内容。

人事调动系统(Human Resource Relocation System)：这个模块会支持 HR 和管理层优化员工调配，从而使公司最大限度地发挥员工的潜力。人事调动系统可以生成员工汇总、技能矩阵、组织图等各种数据类型，让 HR 找到合适的员工摆正位置，解决内部缺陷和瓶颈区，提高生产力和企业竞争力。

薪资管理系统(Payroll Management System)：这个模块管理着员工的薪资、奖金、总收入等，也会包括公司提供的其他福利计划，如医疗补偿和税务信息，保险等。薪酬信息需要与其他功能相互关联，确保每笔支付都是准确、及时、合法的。

安全模块(Security Module)：由于人事管理系统中包含重要的员工和公司数据，应该加强安全运营，例如实施身份验证/授权、访问控制和审计追踪等内容。

4.1.4 软件单元

基于系统的体系结构模块和软件架构，给出具体的软件单元：

员工信息管理软件单元：用于管理员工信息和员工档案。它可以包括员工基本信息、工作历史、培训记录、出勤记录、福利计划等。

人事调动管理软件单元：用于管理和控制员工变化的职位和部门，帮助 HR 找到合适的员工来填补空缺，并优化公司的结构构架。这个模块也会生成以技能矩阵描述员工信息，以帮助 HR 快速找到最适合工作职位的员工或者未来的成功装

备和发展对象。

薪资管理软件单元：用于管理员工的薪资、奖金、总收入等数据。非常重要的是，这个模块需要与其他功能相互关联，确保每笔支付都是准确、及时、合法的。

安全管理软件单元：提供必要的身份验证、授权、访问控制和审计追踪等安全保障措施以保护员工和公司信息不被泄露、篡改、破坏或滥用。

用户界面软件单元：这是系统的前端界面，用户将在此输入/查看他们的员工信息、发起调动请求等。

数据库管理软件单元：用于存储所有关于员工和公司的数据。它需要遵循最佳实践，并考虑数据安全性和可扩展性。

员工自助服务单元：这个模块允许员工通过在线平台进行请求、查询或更新信息，例如查看工资单、选择医疗补偿福利等。

培训管理单元：这个模块记录员工参加培训的历史和未来计划，并管理课程的安排。它可以支持在线培训，预约教室等并管理培训费用的支出。

绩效管理单元：这个模块跟踪员工绩效评估的进度、结果和报告，帮助公司了解员工表现和对企业业务目标的满足程度。此模块可包括定期/临时考核、自我评估、直接主管评论、360 度反馈、互动评测等方式。

招聘管理单元：该组件提供一种标准化过程，管理公司的所有招聘活动。它可以包括发布岗位要求吸引应聘者，收集应聘者简历、联系面试、邀请入职测试，签约入职等等程序。

社交媒体单元：通过社交媒体平台快速发展的现代人类交往方式，将有助于人事部门寻找新渠道，与潜在的候选人和员工之间建立联系。

4.1.5 架构观点

当我们考虑解决一个较大的问题时，通常会将其拆分为更简单的小问题。SOA 的基本思想就是将业务功能组件化并以松耦合的方式构建。这些组件是独立的，它们通过定义的接口进行相互交互。

实现 SOA 体系结构意味着要将这些独立的服务部署在不同的计算机上，互相交互来完成整个系统的任务。每个服务都独立于其他服务，且应当只关心自己的本职责任。因此，每个服务都可以定期进行版本升级、改进性能、修复漏洞等操作，而不会影响到整个系统的稳定性和可用性。

SOA 体系结构的另一个核心优势是，服务是松散耦合的。也就是说，由于服务只与它们所连接的组件息息相关，因此单个服务的故障或出现运营问题不会造成整个系统的崩溃或失效情况。高度耦合的设计很容易导致可靠性低下和不利于维护，而 SOA 中松散耦合的设计措施迭代更新，一旦固化了之后扩展新功能模块需要获得支持变得更容易。

SOA 体系结构还具有可重用性，可以将开发的服务在其他应用程序和系统中再次使用，而不必担心与新环境或流程是兼容的。这提高了效率并减少了成本和风险。 最后，SOA 体系结构削减了部署和管理技术的复杂性，因为每个服务组件都独立工作，所以更容易维护和控制。

4.2 全局数据结构说明

4.2.1 常量

1. 使用到的常量数据文件名称及其所在目录：

tax_rates.dat - 存储税率信息的数据文件，可能位于系统根目录下的 data 目录中。

holiday_dates.txt - 存储节假日日期的文本文件，可能位于系统根目录下的 config 目录中。

employee_types.xml - 存储员工类型信息的 XML 文件，可能位于系统根目录下的 data 目录中。

user_roles.json - 存储用户角色信息的 JSON 文件，可能位于系统根目录下的 config 目录中。

2. 用到的常量

MAX_EMPLOYEE_NUM：定义公司可以拥有的最大员工数量；

MAX_DEPARTMENT_NUM：定义公司可以拥有的最大部门数量；

MAX_SALARY_LEVEL：定义公司可以设置的最高薪资等级；

MIN_SALARY_LEVEL：定义公司可以设置的最低薪资等级；

MAX_BONUS_PERCENTAGE：定义公司可以设置的最高奖金比例；

MIN_BONUS_PERCENTAGE：定义公司可以设置的最低奖金比例；

TAX_RATE：定义公司设置的所得税率；

SOCIAL_SECURITY_RATE：定义公司设置的社会保险费率；

HOUSING_FUND_RATE：定义公司设置的住房公积金缴存比例。

这些常量可以用于限制系统中各种数据的范围，并提高系统的可维护性。

4.2.2 变量

1. 可能涉及到的数据文件名称及其所在目录:

   ①员工信息数据文件:

   文件名称：employee.dat

   存放目录：/data/employee/

   ②工资信息数据文件:

   文件名称：salary.dat

   存放目录：/data/salary/

   ③权限信息数据文件:

   文件名称：permission.dat

   存放目录：/data/permission

   ④数据备份文件:

   文件名称：backup.dat

   存放目录：/data/backup/

具体使用的数据文件名称和目录结构可能因实际情况而异。在设计和实现过程中，应根据具体需求来确定所需的数据文件及其存放位置。

2. 用到的变量

departmentList: 部门列表，包括每个部门的名称、编号等信息。

positionList: 职位列表，包括每个职位的名称、编号、所属部门等信息。

salaryScaleList: 工资等级列表，包括每个等级的名称、编号、基本工资等信息。

deductionList: 扣款项列表，包括每个扣款项的名称、编号、扣款标准等信息。

bonusList: 奖金项列表，包括每个奖金项的名称、编号、奖金标准等信息。

userRoleList: 用户角色列表，包括每个角色的名称、编号等信息，用于权限管理。

logList: 系统操作日志列表，包括每个操作的时间、操作人员、操作内容等信息，用于日志记录和审计。

backupList: 数据库备份列表，包括每个备份的时间、备份人员等信息，用于数据恢复。

4.2.3 数据结构

1. Employee (员工信息)

功能说明：用于存储员工基本信息的数据结构。

具体数据结构说明：包含员工编号、姓名、性别、年龄、联系方式、部门、职位等信息。

定义：

```
struct Employee {
    int empId; // 员工编号
    string name; // 姓名
    string gender; // 性别
```

```
    int age; // 年龄
    string phone; // 联系方式
    string department; // 部门
    string position; // 职位
};
```

2.Salary (工资信息)

功能说明：用于存储员工工资、奖金、扣款等信息的数据结构。

具体数据结构说明：包含员工编号、基本工资、津贴、奖金、扣款等信息。

定义：

```
struct Salary {
    int empId; // 员工编号
    float basicSalary; // 基本工资
    float allowance; // 津贴
    float bonus; // 奖金
    float deduction; // 扣款
};
```

3. Analytics (数据分析)

功能说明：用于存储员工信息的数据结构，以支持企业或机构的管理决策。

具体数据结构说明：包含部门、职位、基本工资、津贴、奖金、扣款等信息。

定义：

```
struct Analytics {
    string department; // 部门
    string position; // 职位
    float basicSalary; // 基本工资
    float allowance; // 津贴
    float bonus; // 奖金
    float deduction; // 扣款
};
```

4. Backup (数据备份)

功能说明：用于存储系统数据备份的数据结构。

具体数据结构说明：包含备份日期、备份时间、备份文件名等信息。

定义：

```
struct Backup {
    string backupDate; // 备份日期
    string backupTime; // 备份时间
    string backupFileName; // 备份文件名
```

};

5. Permission (权限管理)

功能说明：用于存储系统权限管理的数据结构。

具体数据结构说明：包含用户类型、用户权限等信息。

定义：

```
struct Permission {
    string userType; // 用户类型
    string userPermission; // 用户权限
};
```

4.3 CSCI 部件

a.其 CSCI 软件配置项如下：

数据库：包括员工信息、人事调动信息和薪资信息等。

用户界面：提供交互式的操作界面，使得用户可以方便地进行各类操作。

人事管理模块：实现员工档案、考勤管理、员工奖惩等功能，可实现员工的全生命周期管理。

薪资管理模块：实现工资计算、社保代缴、五险一金等功能，保障员工的权益和公司的法律合规。

系统管理模块：实现系统权限管理、日志记录、安全措施等功能，保证系统的安

全性和稳定性。

b.本系统的软件配置项静态关系如下:

用户界面依赖于人事管理模块和薪资管理模块,因为用户需要在界面上进行的操作需要通过这两个模块来实现相应功能。

人事管理模块和薪资管理模块都依赖于数据库,因为它们需要对其中存储的信息进行读取、修改、删除等操作,以实现员工全生命周期管理和工资计算、社保代缴等功能。

系统管理模块则是独立模块,它不依赖其他配置项的功能,但包括它自己的功能和数据会被其他配置项使用和调用。

总体而言,本系统采用了模块化设计的思想,实现了各模块之间的松耦合,易于维护和扩展。

c.以下是该人事管理系统中各软件配置项的用途:

数据库:作为存储所有员工信息、人事调动信息和薪资信息的核心组件,提供数据访问服务,支持对数据进行查询、修改、删除等操作。

用户界面:提供可视化的交互式操作界面,让用户可以直观地管理员工档案、员工考勤、员工调动、员工工资等方面的信息。它与其他系统模块交互,把用户输入输出传递给底层模块来完成相关操作。

人事管理模块:主要负责员工档案管理、考勤管理、员工奖罚记录、员工调动等功能,实现员工全生命周期管理,维护企业用工安排。

薪资管理模块:主要负责计算员工工资、发放工资、社保费用代缴等工作,确保

员工利益，防止因失误导致的纠纷和法律风险。

系统管理模块：包括系统权限管理、日志记录、安全措施等，确保系统的稳定性和安全性，防止未经授权或错误操作造成的系统故障及信息泄露等问题。同时，能够追踪管理人员及员工在系统上的操作状况，方便出现问题后的处理和调查。

通过这五个软件配置项的协同配合，该人事管理系统可以完成全面、高效，安全的员工管理以及企业内部信息的统计分析等功能。

d.以下是该人事管理系统中各软件配置项的开发状态/类型：

数据库：开发完成，会根据需要进行调整和优化。

用户界面：正在开发，在 UI 设计、实现及测试等方面持续改进，并且需要与其他配置项进行充分融合测试。

人事管理模块：已开发完成，并已适当测试。它是该人事管理系统的核心部分之一，但在未来可能需要进行调整以满足不断变化的需求。

薪资管理模块：已开发完成，并已经进行充分的测试。它是该人事管理系统的重要组成部分之一，能够对企业内部的工资管理工作提供重要帮助。

系统管理模块：已经开发完成，经过充分测试。它包括安全策略和日志记录等功能来确保系统的稳定性和安全性。

总体来说，该人事管理系统已经具备了较好的基础功能，但可能在未来会进行各方面的升级和调整以应对时代变化。

e.以下是该人事管理系统中各软件配置项计划使用的计算机硬件资源：

数据库：由于需要存储大量员工信息和相关数据，因此需要一台高性能的服务器

来处理数据库的读写操作。其中 CPU、内存、磁盘空间等方面的要求可能会根据数据量的变化而有所调整。

用户界面：主要运行在计算机的图形用户界面（GUI）上。对于 GUI 运行环境的要求相对较低，通常通过一个普通的桌面或者笔记本电脑就可以满足要求。

人事管理模块：可能需要一个中央处理单元（CPU）速度较快、内存充足的计算机环境，以确保其正常运行和快速响应。

薪资管理模块：它需要进行大量的计算和数据读写，因此需要高性能的 CPU 以及大量的内存来完成工资计算、记录管理等操作。

系统管理模块: 需要具备较高的安全可靠性, 因此对于硬件资源也有较高的要求。例如，需要稳定的电源保障、自动备份机制等。

总体来说，在不同的组件中，根据功能、需求和实现方式的不同，其计算机硬件资源需求是有所区别的。决定哪些硬件资源需要使用以及使用的规格，需要根据具体情况进行灵活调整。

4.4 执行概念

执行控制流：CSCI 之间的执行控制流可能包括顺序执行、条件分支、循环等结构。例如，在员工信息管理模块中，用户可以选择录入、修改、查询、统计或打印员工信息等操作，系统将根据用户的选择执行相应的控制流。

数据流: CSCI 之间的数据流通常用于传递信息和状态, 以实现不同模块的交互。例如，在员工信息管理模块中，当用户选择录入员工信息时，录入界面将接收用户输入的信息，并将其传递给后台数据处理模块。

动态控制序列：CSCI 之间的动态控制序列用于实现复杂的交互行为，例如一些高级操作或工作流程。例如，在工资管理模块中，用户可以提交加班或请假申请，这需要经过审批和审核等一系列流程才能最终生效。

状态转换图：状态转换图描述 CSCI 的状态以及状态之间的转换关系，通常用于描述状态机或者有限状态自动机。例如，在员工信息管理模块中，一个员工信息可能有不同的状态，例如已录入、已审核、已发布等，这些状态之间可能存在一些转换关系。

时序图：时序图描述 CSCI 之间的时序关系，即各个配置项之间的交互顺序和时间。例如，在数据分析模块中，用户可以按照不同的条件查询员工信息，查询结果将在一定时间内返回并展示给用户。

配置项之间的优先关系：配置项之间可能存在一些优先关系，即某些配置项需要在其他配置项执行之前执行。例如，在数据备份模块中，备份操作必须在数据恢复操作之前执行，以保证数据的完整性和可靠性。

中断处理：中断处理通常用于处理一些异常情况，例如系统崩溃、用户取消操作等。在这些情况下，系统可能需要执行一些特殊的处理程序，以保证系统的安全和稳定性。

时间/序列关系：CSCI 之间可能存在一些时间或者序列关系，例如某些操作需要在特定的时间或者特定的顺序下执行。例如，在工资管理模块中，每月的工资发放必须在特定的时间点执行，以保证工资的及时性和准确性。

异常处理：异常处理用于处理一些不可预期的异常情况，例如输入错误、

4.5 接口设计

1.用户接口：

用户应该能够以友好的方式与系统进行交互。

用户应该能够通过图形用户界面（GUI）输入、修改、删除和查询员工信息。

用户应该能够方便地浏览员工信息，并能够根据不同条件进行排序和过滤。

用户应该能够通过菜单或快捷键访问系统的各个功能模块。

用户应该能够方便地打印和导出员工信息的查询和统计结果。

系统应该提供帮助文档和用户手册，以便用户能够理解和使用系统。

2.硬件接口：

系统应该能够运行在 Windows 或 Linux 等常见操作系统上。

系统应该能够支持标准的输入和输出设备，如键盘、鼠标、打印机、显示器等。

系统应该能够支持数据库服务器，如 MySQL、Oracle 等。

系统应该能够支持网站服务器，如 Apache、Tomcat 等。

3. 软件接口：

系统应该能够与数据库服务器进行通信，并能够使用 SQL 语言进行数据操作。

系统应该能够使用第三方组件或库，如 JQuery、Bootstrap 等。

系统应该能够使用标准的 API，如 RESTful API 等。

系统应该能够与其他系统进行数据交换，如 ERP 系统、HR 系统等。

4. 通信接口的需求：

系统应该能够支持 HTTP、HTTPS 协议进行数据传输。

系统应该能够支持 SMTP 协议发送邮件。

系统应该能够支持 FTP 协议进行文件传输。

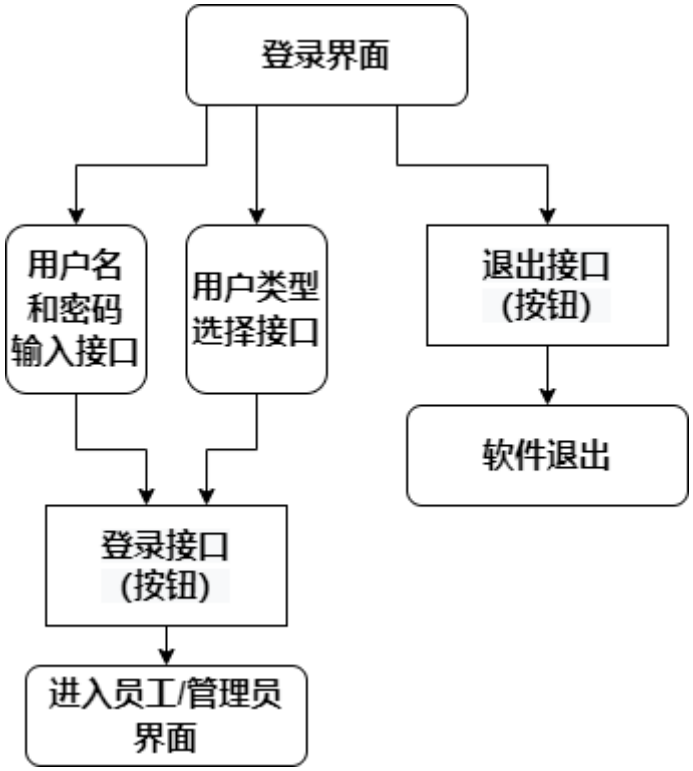
系统应该能够支持 SOAP 协议进行 Web 服务调用。

4.5.1 接口标识和接口图

本条标识所需的 CSCI 外部接口，也就是 CSCI 和与它共享数据、向它提供数据或与它交换数据的实体的关系。该标识应说明哪些实体具有固定的接口特性(因而要对这些接口实体强加接口需求)，哪些实体正被开发或修改(从而接口需求已施加给它们)。可用一个或多个接口图来描述这些接口。

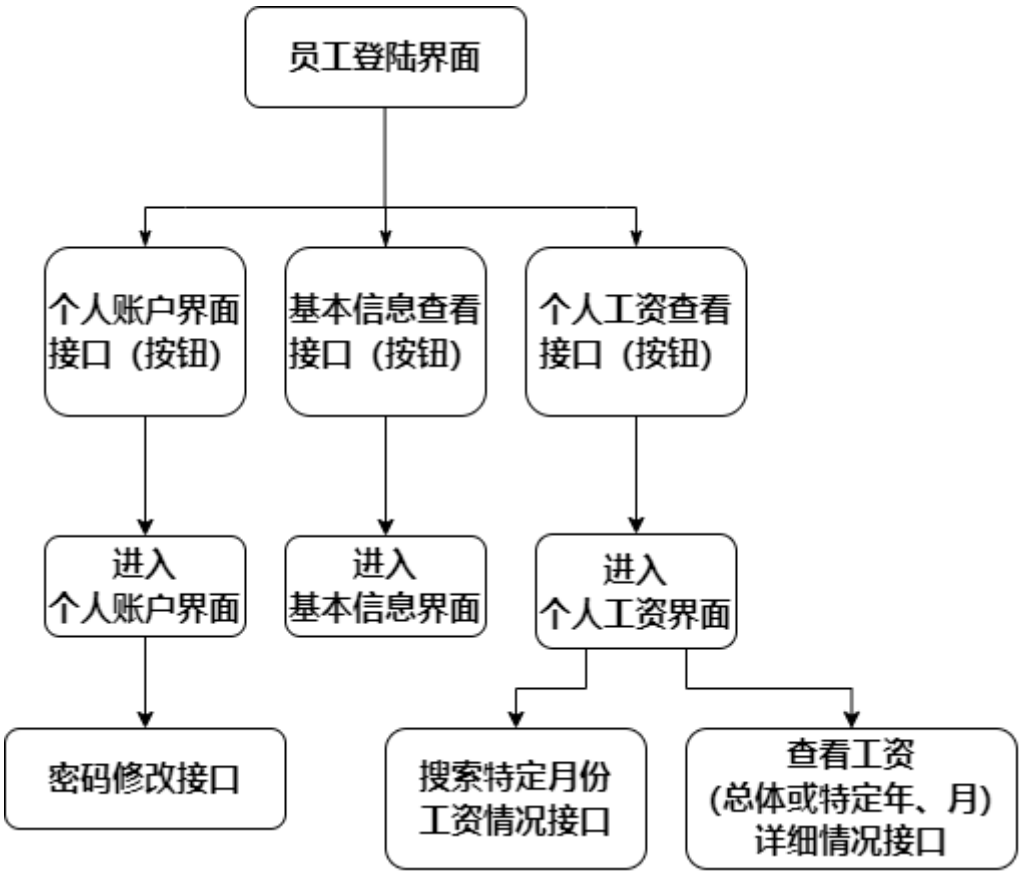
4.5.2 登陆界面的接口图

登陆界面的接口图如下：



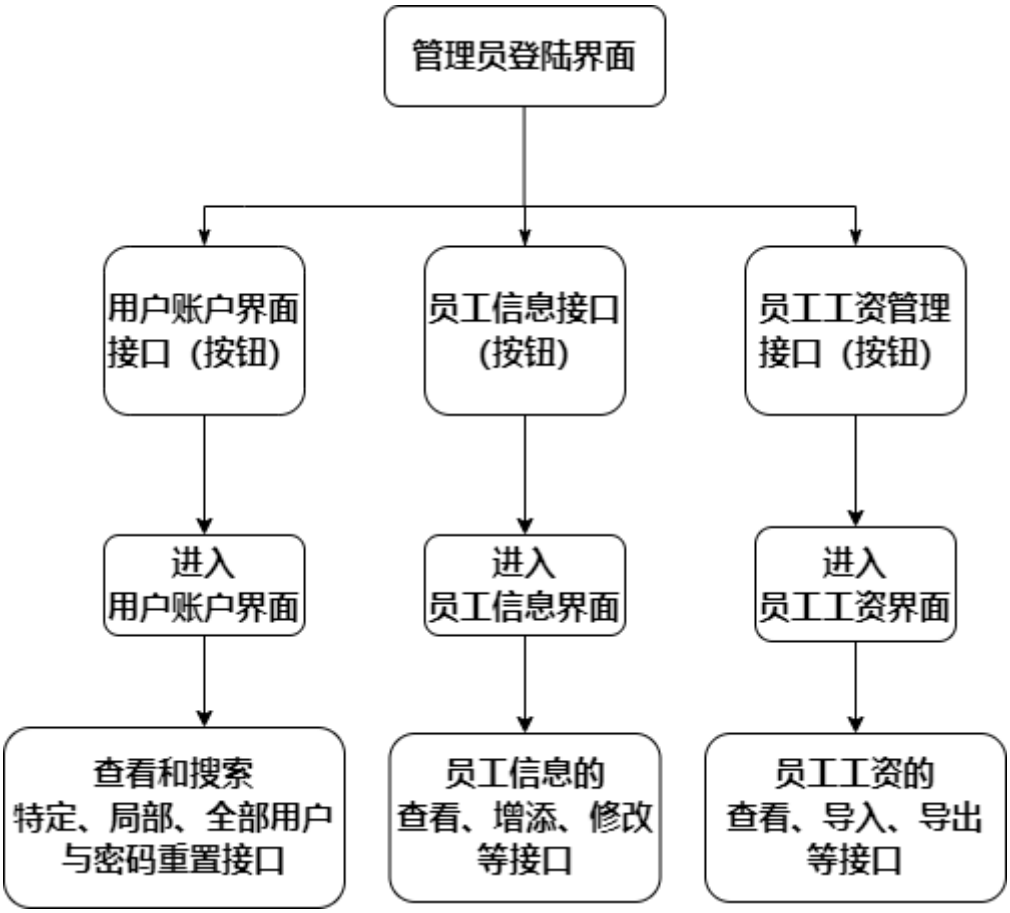
4.5.3 员工界面的接口图

员工界面的接口图如下:



4.5.4 管理员界面的接口图

管理员界面的接口图如下:



5 CSCI 详细设计

5.1 体系结构框架和体系结构描述语言

以下是该人事管理系统的体系结构框架：

Presentation Layer：用户界面，包括用户登录和操作界面。

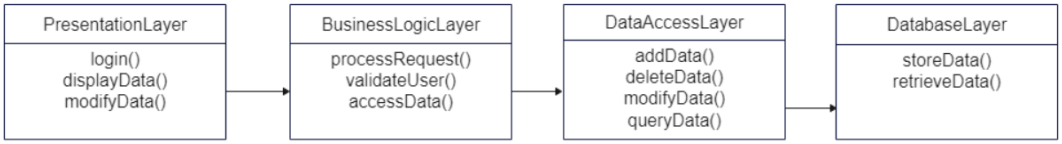
Business Logic Layer：业务逻辑层，处理用户请求，调用数据访问层来访问数据库。

Data Access Layer：数据访问层，负责与数据库交互，包括增删改查等操作。

Database Layer：数据库层，存储所有的数据。

体系结构描述语言采用 UML（Unified Modeling Language），使用类图、时序图、活动图等来描述体系结构。

一个简单的例子如图所示：



5.2 详细设计

a. 本软件配置项的设计决策包括使用现代的 C++开发技术来实现系统的前端界面，同时采用后端框架和数据库来支持系统的后台业务逻辑。

b. 客户端软件配置项需要满足以下约束和限制：
客户端需要连接到服务器进行数据交互
客户端需要实现界面交互，如输入员工信息、查询员工信息、显示员工列表等
客户端需要保证数据传输的安全性和完整性

c. C++是该 CSCI 所指定的编程语言，其具有较高的性能和可扩展性，同时也具有较好的面向对象编程支持。可以提高系统的开发效率和稳定性。

d. 客户端软件配置项中包含过程式命令和一些命令行工具或者脚本来辅助管理和操作系统，如：
查询员工信息的 SQL 语句
更新员工信息的 SQL 语句
显示员工列表的 GUI 构造器的输入
用户权限管理命令：包括添加用户、修改用户信息、删除用户、用户权限管理等命令。
系统配置管理命令：包括修改系统参数、配置数据备份、数据恢复、系统维护等命令。
数据库备份工具：用于备份人事管理系统所使用的数据库，包括员工信息、薪资信息等。

数据库恢复工具：用于恢复备份的数据库文件。
数据库初始化脚本：用于在系统首次启动时初始化数据库，包括创建数据库表、添加默认数据等。
数据库升级脚本：用于在系统升级时更新数据库结构，保证旧数据的兼容性。
系统日志脚本：用于记录系统日志，包括用户登录信息、操作日志等。

e. 人事管理系统的 CSCI 应包含以下数据元素：
员工信息：包括员工姓名、工号、性别、出生日期、联系电话、家庭住址、身份证号码、入职日期、部门、职位、薪资等。
员工考勤记录：包括员工出勤情况、迟到、早退、缺勤等记录。
员工请假记录：包括员工请假开始时间、结束时间、请假类型、请假原因等记录。
员工培训记录：包括员工培训开始时间、结束时间、培训类型、培训内容等记录。
员工奖惩记录：包括员工奖励时间、奖励类型、奖励原因、惩罚时间、惩罚类型、惩罚原因等记录。
部门信息：包括部门编号、部门名称、部门描述等。
职位信息：包括职位编号、职位名称、职位描述等。
薪资信息：包括薪资编号、薪资等级、薪资范围等。
系统设置信息：包括系统参数设置、用户权限设置、系统日志等记录。
客户端软件配置项包含输入、输出和其他数据元素，如：
输入数据：员工的基本信息，如姓名、性别、出生日期、联系方式等
输出数据：员工的基本信息，如姓名、性别、出生日期、联系方式等
其他数据元素：员工的薪资、职位、考勤记录等

f. 客户端启动时需要先连接服务器
客户端界面交互的响应包括：
输入员工信息
查询员工信息
显示员工列表
响应时间包括：

数据转换

数据传送操作

客户端运行期间的操作序列和动态控制序列包括：

a) 序列控制方法：事件驱动

b) 该方法的逻辑与输入条件：客户端交互事件

c) 数据在内存中的进出：通过网络连接与服务器进行数据交互

d) 离散输入信号的感知：客户端交互事件

异常与错误处理包括：

a) 输入数据格式错误或缺失：应该对输入数据进行格式检查，确保其满足数据类型和范围的要求。如果输入数据缺失，则应该提示用户提供必要的数据。

b) 数据库连接失败：如果连接数据库失败，则应该显示错误消息，并提示用户检查数据库设置和网络连接。

c) 数据库操作失败：如果数据库操作失败，例如插入或更新数据失败，或者查询数据不存在，则应该显示错误消息，并提示用户重新尝试或联系管理员。

d) 系统资源不足：如果系统资源不足，例如内存或磁盘空间不足，则应该显示错误消息，并提示用户释放一些资源或联系管理员。

e) 未知异常：如果发生未知异常，应该记录错误信息，并提示用户联系管理员。

6 需求的可追踪性

a. 从软件配置项到 CSCI 需求：

软件配置项：人事管理系统用户接口

CSCI 外部接口需求：接口必须在 Windows 和 Linux 平台下运行

CSCI 内部接口需求：用户登录接口必须与员工信息管理接口交互

CSCI 内部数据需求：需要在系统中存储和管理员工个人信息

适应性需求：用户界面必须易于使用，可自定义

保密性和私密性需求：员工的个人信息必须保密

CSCI 环境需求：系统必须在企业内网中运行

计算机软件需求：需要安装 MySQL 数据库和 Qt 框架

计算机通信需求：系统需要与企业内部服务器进行通信

软件配置项：员工信息管理模块

CSCI 能力需求：模块必须能够添加、修改和删除员工信息

CSCI 内部接口需求：模块必须与用户接口交互

CSCI 内部数据需求：需要在系统中存储和管理员工个人信息

适应性需求：员工信息必须可以按不同的类别进行管理

保密性和私密性需求：员工的个人信息必须保密

软件配置项：薪资管理模块

CSCI 能力需求：模块必须能够计算、审核和发放薪资

CSCI 内部接口需求：模块必须与员工信息管理模块交互

CSCI 内部数据需求：需要在系统中存储和管理薪资信息

适应性需求：薪资计算必须符合公司政策和法律法规

保密性和私密性需求：员工的薪资信息必须保密

b. 从 CSCI 需求到软件配置项：

CSCI 外部接口需求：接口必须在 Windows 和 Linux 平台下运行

软件配置项：人事管理系统用户接口

CSCI 内部接口需求：用户登录接口必须与员工信息管理接口交互

软件配置项：人事管理系统用户接口、员工信息管理模块

CSCI 内部数据需求：需要在系统中存储和管理员工个人信息

软件配置项：人事管理系统用户接口、员工信息管理模块、薪资管理模块

CSCI 能力需求：模块必须能够添加、修改和删除员工信息

软件配置项：员工信息管理模块

适应性需求：用户界面必须易于使用，满足不同用户的需求和能力
软件配置项：人事管理系统用户接口

保密性和私密性需求：员工信息必须加密存储，只有授权的管理员才能访问
软件配置项：员工信息管理模块、数据加密模块

CSCI 环境需求：系统必须在 Windows 10 和 Linux CentOS 7 操作系统上运行
软件配置项：人事管理系统用户接口、员工信息管理模块、薪资管理模块、数据加密模块、操作系统适配模块

计算机硬件需求：系统必须在 x86 架构的计算机上运行，至少需要 1GB 的内存和 500MB 的磁盘空间
软件配置项：人事管理系统用户接口、员工信息管理模块、薪资管理模块、数据加密模块、操作系统适配模块

计算机硬件资源利用需求：系统必须能够充分利用计算机的硬件资源，保证系统运行的高效性和稳定性
软件配置项：人事管理系统用户接口、员工信息管理模块、薪资管理模块、操作系统适配模块

计算机软件需求：系统必须使用 C++编程语言实现，需要使用相关的第三方库和框架，如 Qt
软件配置项：人事管理系统用户接口、员工信息管理模块、薪资管理模块、数据加密模块、操作系统适配模块

计算机通信需求：系统需要使用网络通信协议，实现远程访问和数据传输功能
软件配置项：员工信息管理模块、薪资管理模块、网络通信模块

7 注解

1.故障树分析法

故障树分析法(Fault Tree　Analysis，FTA)是在对系统的可靠性进行分析时最常用的方法之一。FTA 方法是指在系统设计或改进过程中，通过对可能造成系统故障的各种因素(包括硬件、软件、环境、人为因素等)进行分析，画出逻辑框图(即故障树)，从而确定系统故障原因的各种可能组合方式及其发生概率，并以此计算系统的故障概率，采取相应的措施，以提高系统可靠性的一种设计分析方法和评估方法。

2.故障树分析法对于数据库故障解决的意义

经过在实践和应用中的总结，故障树分析法作为一种分析方法和思路，同样适合数据库故障的分析和解决，如果扩展一步来说，这种方法作为一种思维方式，甚至适合生活中所有事件的分析和处理。

但是需要注意的是，故障分析实际上是一种事后分析的方法，当然我们不希望工作、生活中当事故、问题出现后再来分析，所以，我们一直提倡将故障树分析在事前实施，通过参考别人的经验、教训，将故障树引入事前，人类的学习特点应当能够使我们从学习中而不是亲身经历去获得经验。像 SQL 审核，就是一个例子，通过提前探知可能存在的隐患，给予意见和关注，避免出现问题。

通过实践我们发现，将应用于传统行业的故障树分析法引入到数据库故障分析及问题解决之中，可以极大地加快问题分析、处理和解决的速度，同时可以帮助我们发现系统的缺陷所在，从而通过实施有效的预防措施显著地提高系统的稳

定性和可靠性。

8 附录 1 学习笔记

附录中包括了每个人对应的学习笔记。

8.1 杨逸的学习笔记

该学习笔记主要是通过阅读 Software Architecture: a Roadmap 这篇论文后所写的学习笔记。这篇学习笔记根据论文所讲述的内容分为六个部分，第一个部分写的是内容提要，简要的介绍了这篇文章的主要内容；第二部分到第五部分则是根据该论文的内容，针对其每个小节所做的具体阐述；第六部分则是笔记作者阅读完这篇论文后的读后体会。

8.1.1 内容提要

本文讲述的核心是软件体系结构发展路线图，作者从软件体系结构的过去讲述到软件体系结构的现在，并对软件体系结构的未来做了一定的预测。最终我总结的该文的内容提要如下：

1. **文章背景：**在过去十年间，作为软件工程的一个重要领域，软件体系结构越来越受到关注。在设计、构造任何一个复杂软件时，一个关键问题就是它的体系结构。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，而一个坏的架构可能是灾难性的。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，以及在开发一个新产品时，

对过去设计的重用的重要性。

2. **现状与挑战：**虽然对于软件体系结构的研究有了很大进展，但这个领域相对来说还不成熟。尽管一个工程的软件架构的基本轮廓逐渐变得清晰，但仍面临许多挑战和未知。
3. **文章摘要：**本文考察了软件体系结构在研究和实践中的一些重要趋势。作者首先描述了体系结构在软件系统开发中的作用。然后总结了软件体系结构的过去和现在的发展情况。最后，对软件体系结构未来的发展及挑战做了预测。

8.1.2 软件体系结构的作用

小节总结：本小节首先指明了软件体系结构的定义和对应作用，**指明软件体系结构通常作为需求和实现之间的桥梁发挥着关键作用。**本小节指出，软件体系结构在软件开发的六个方面（Understanding，Reuse，Construction，Evolution，Analysis，Management）发挥作用 。**本小节的核心作用是指明软件体系结构的重要性，从而进一步证实了本论文关于软件体系结构对于过去和现在的总结，以及对未来发展预测的重要性。**

本小节简介：

1. 本小节首先指明软件体系结构的定义，即其定义的核心是系统的体系结构描述其总体结构的概念。同时，它指明了软件体系结构的作用，即**软件体系结构通常作为需求和实现之间的桥梁发挥着关键作用。**

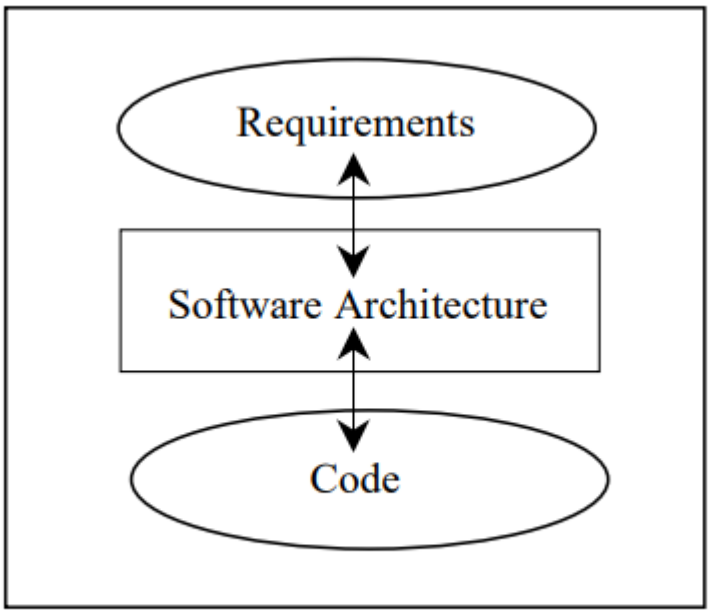


Figure 1: Software Architecture as a Bridge

2. 紧接着，本小节**进一步指明了体系结构的内核原理**，即通过提供系统的抽象描述，体系结构公开了某些属性，同时隐藏了其他属性。理想情况下，这种表示为整个系统提供了一个智能化的指导，允许设计者对系统满足某些需求的能力进行推理，并为系统构建和组成提供了蓝图。

3. 最后，本小节**详细描述了软件体系结构可以在软件开发的至少六个方面发挥重要作用**，即软件体系结构可以在 Understanding，Reuse，Construction，Evolution，Analysis，Management 六个方面对于软件开发起重要帮助。

8.1.3 软件体系结构的过去

小节总结：十年前的软件体系结构的描述大多依赖于非形式化的箱线图。**但是人们已经意识到架构设计是一个非常重要的领域**，并产生了两个突出其重要性的趋势，第一个是认识到在构建复杂软件系统时对方法、技术、模式的共享，第二个

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。

本小节简介：

1. **过去的背景：**十年前，体系结构的描述大多依赖于非形式化的箱线图。虽然不正式，但架构设计对整个系统设计是重要的。

2. **当时的观点：**人们意识到需要一个更严格的方法。早期的学者观察到某些统一的规则，开始号召架构设计作为一个需要关注的领域，并发明了软件架构师这个工作词汇。工具供应商开始考虑对架构设计的支持，语言设计者也开始考虑架构表现的符号。

3. **行业内的两种趋势**：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。常见例子：用户界面工具包和框架（提供可重用组件，如菜单和对话框）。

8.1.4 软件体系结构的现在

小节总结：本小节主要**详细讲述了当今时代软件体系结构在技术基础的三个进步**，即体系结构描述语言和工具的发展；产品流水线生产和架构标准的产生；关于体系结构设计专业知识的传播和编纂。

关于三个进步的详细描述如下：

1. 体系结构描述语言和工具的发展

①**形式化的符号：**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）。

②**现在的各种体系结构描述语言：**ADL 提供具体的语法与刻画体系结构的概念框架，通常还提供工具，用于分析、显示、编译或是模拟架构描述。已经提出若干适用于特定领域的 ADL，典型的有 UniCon，DarwinRapide，C2，SADL，Wright。C2 支持用户界面系统采用基于事件的风格描述。Darwin 支持对消息传递系统的支持。Rapide 允许模拟架构设计，并有工具对这些模拟结果进行分析。Wright 是一种结构描述语言，该语言基于一种形式化的、抽象的系统模型，为描述和分析软件体系结构和结构化方法提供了一种实用工具。Wright 侧重于描述系统的软件构件及其连接的结构、配置和方法。

③**继承架构基础工具和软件开发包：**最近 ADLs 的扩展能力促使一项调查把符号的方法和工具集合在一起。更加令人激动的是，一些研究人员开始关注集成架构基础工具的其他方法，有望弄出一个灵活的“ADL 工具包”，这允许从业者结合现有工具的能力来创建特定领域的体系结构设计环境。另一努力是在软件开发各方面，整合体系结构的开发工具和其他工具。此外，最近出现了一些试图展示如何将在日常生活活动中发现的概念可以直接映射到像 UML 的面向对象表示法的建议。

2. 产品线和标准（跨供应商的集成标准）

①**产品线的进步：**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。虽然产品线工程尚未普及，但我们对实现产品线所需的过程、工件有了更好的认识。现在已发表了一些产品线成功案例研究。此外，像软件工程学院这样的组织正努力提供具体的指导方针和流程，以供产品线方法使用。

②**跨供应商的集成标准：**类似产品线方法，跨供应商的集成标准要求一个允许系统开发人员通过实例化该框架来配置各种具体制度的架构框架。集成标准支持由多个供应商提供的零件的集成。前者一个很好的例子是高层体系结构的分布式仿真。这种体系结构是许多厂商生产的模拟的集成。同时，该标准规定模拟组件能表明在使用共享服务时，它们有什么能力，有什么限制。如 Sun 的企业级 Java-BeansTM（EJB）体系结构是为了支持分布式的，基于 Java 的企业级应用程序，如企业信息管理系统。此外，它定义了一个独立于供应商的接口信息服务，包括事务，持久性和安全性。

3. 编纂和宣传

①**编纂和宣传现状：**早期，阻碍体系结构设计作为一门工程学科出现的原因是缺乏知识架构。今天情况有所好转，部分原因是关于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。一种体系结构风格通常会指定一个设计词汇表，和一组约束。词汇表包含构件和连接件类型，约束指出如何将构件和连接件组合。

②**现有体系结构风格：**常见风格有管道—过滤器体系结构、客户端—服务器结构、基于事件的隐式调用体系结构和面向对象体系结构等。每种体系结构风格适合于某一用途，并不是能满足所有。例如：一个管道过滤器体系结构风格可能更适合一个信号处理应用程序，而不是一个对共享数据的并发访问有显著要求的应用程序。此外，对每种风格通常有相关联的分析。例如：因为系统延迟而分析一个管道过滤器系统是有意义的，而事物存取速度更适合分析存储为导向的风格。

8.1.5 软件体系结构的未来

小节总结：本小节主要**讲述了三个突出的趋势并预测其对软件体系结构领域的影响**，三个突出的趋势为改变内置对战平衡购买；网络为中心的计算；普适计算。

关于三个突出的趋势的详细描述如下：

1. 改变内置对战平衡购买

①**内置对战平衡购买的现状：**在整个软件工程历史中，系统开发的一个主要问题是决策系统的什么部分包含其他，什么部分内置。对外收购部分有节省开发时间的优势，但它们也有经常不完全满足需求的缺点。虽然问题本身没有改变，但经济压力减少上市时间大大改变了平衡。对越来越多的产品，提早一个月介绍产品是成功与失败的区别。这种情况下使用其他人编写的软件架构自己的系统成为唯一可行的选择。现在软件开发人员使用数千行外部代码开发自己的每行代码变得很寻常。事实上，许多公司发现自己处在系统集成位置，而不是软件开发人员位置。许多公司通过并购作为经济增长的主要途径，而不是购买单独的软件部件或产品，公司在吸收其他公司。现在的集成变得更加严重、棘手。

②**全行业标准的需求趋势：**这种趋势强调了对全行业标准的需求。共性越多，第三方软件系统一起工作的可能性越大。由此基于组件的软件开发人气高涨，通过选择组件，达成一个共同的体系结构框架。例如 COM、JavaBeans 和 CORBA，许多体系结构不匹配问题得到缓解。

2. 网络为中心的计算

①**网络为中心的趋势：**从个人 PC 为中心的计算模式到以网络为中心的计算模式有个明显趋势。传统计算机使用集中在个人 PC 上，需手动安装和更新本地应用程序和数据。但越来越多的 PC 和各种其他接口（手持设备、笔记本、手机）主要被用作用户接口，提供对远程数据的访问。这种趋势并不奇怪，因为以网络为中心的计算模式有显著优点。它提供更广泛的基础服务；它允许人们在几乎任何地方（办公室、家、汽车和工厂）使用便携式计算机访问一系列计算和信息检索服务。它通过随时随地的访问信息和服务促进了用户移动性。

②**对软件体系结构的影响：**这对软件工程，尤其是软件体系结构有很大影响。从历史上看，软件系统已经发展成封闭系统，在个别机构控制下开发和运作。然而，在网络无所不在的环境中，系统可能不会集中控制。互联网就是这样一个开放平台的例子，允许个人网站进行交互。这样一组新的软件体系结构挑战出现。**包括以下四个方面**。

③**需要架构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。例如：一个有吸引力的组成形式是隐式调用，这种

体系结构风格中，构件很大程度上是自治的，通过广播消息和其他构件进行交互。使用这种风格的大多数系统，对它的使用属性作出许多假设。例如：一个典型假设是事件传递是可靠的、消息的集中式路由是足够的。然而在以互联网为基础的设置上所有这些假设是值得怀疑的。

④**需要支持动态组成、基于特定任务的分布式资源的计算。**互联网承载了各种各样的资源：主要信息、沟通机制、可调用应用程序、协调资源使用的控制和服务等。这些资源都是独立开发、独立支持的。许多情况下，资源不必知道它们如何被使用，甚至是它们是否正在使用。组件的集成是困难的，因为很难确定每个组件在其操作环境下的假设是什么，更不用说一套组件是否会很好的交互操作，其组合功能是你所需要的。此外，存在许多有用资源但无法顺利整合，因为他们对组件交互做了不兼容假设。例如，很难将一个封装成通过远程过程调用进行交互的组件和一个在专有表示上共享数据进行交互的组件集合。

⑤**需要找到一个架构灵活满足商业应用服务。**在未来，每个桌面上应用程序将由本地和远程计算能力组成，要求支持计费，安全等体系结构。

⑥**需要开发允许终端用户做自己系统集成的体系结构。**随着互联网的快速发展，越来越多的用户都处在组装和定制服务的位置上。这些用户也许有起码的技术专长，但仍然希望能保证组成部分以他们期望的方式一起工作。

3. 普适计算

①**普适计算的趋势：**在计算宇宙填充了丰富的异构计算设备的普适计算，像烤面

包机，家用加热系统，娱乐系统，智能汽车等。这种趋势可能会导致我们当地的设备数量爆炸—从几十台设备到数百或数千设备。另外，这些设备可能是相当异构的，需考虑他们的物理资源和计算能力。

②**对软件体系结构影响：**这对软件体系结构提出很多挑战，**主要包括三个方面**。首先，我们将需要一个适合于系统中，资源的使用是一个关键问题的架构。第二，系统的体系结构将比今天更加灵活。特别是，设备可能以不可预知的方式运作。第三，就是需要体系结构能更好的处理用户的移动性。目前，我们的计算环境是手工配置的，由用户显示管理。我们需要找到一种体系结构来提供更多的自动化控制的计算机服务管理，这允许用户方便地从一个环境移动到另一个环境。

8.1.6 阅读体会

通过对该论文《Software Architecture: a Roadmap》的阅读，我对软件体系结构在软件开发的作用，特别是它作为需求和实现之间的桥梁的关键作用有了深刻的体会。同时，通过作者对软件体系结构的过去、现在的发展状况的描述以及对软件体系结构未来的发展及挑战的预测，我深入的理解和体会了软件工程体系结构的发展和未来的趋势，好似站在时间轴上一步一步跟着各位软件体系结构的学者和专家一起走过了软件体系结构几十年的发展历程。

总之，这篇文章不仅让我对软件体系结构的定义和作用有了进一步的认知，而且让我对软件体系结构的历史脉络，以及它的未来前景有了宏观上的认识，进一步激发了我对软件工程这门课的喜爱和兴趣。

8.2 闫威的学习笔记

8.2.1 《Software Architecture in Practice，3rd Edition》

《Software Architecture in Practice，3rd Edition》是一本软件架构实践的经典著作，作者为Len Bass、Paul Clements和Rick Kazman，本书介绍了软件架构的概念、方法和实践技术，旨在帮助读者更好地理解和应用软件架构。

本书主要分为四部分，分别是软件架构基础、软件架构分析、软件架构设计和软件架构实践。

第一部分：软件架构基础

第一部分主要介绍了软件架构的概念、架构与设计的区别、架构视图和架构风格等基础知识。其中，架构视图是软件架构的重要组成部分，它用于描述架构的不同方面，如组件、连接和数据流等。架构风格则是一种通用的架构模式，它提供了一种标准化的方式来描述和构建软件系统。本部分的学习让我更加清晰地认识到软件架构的基础知识和重要性。

第二部分：软件架构分析

第二部分主要介绍了软件架构分析的方法和技术。其中，质量属性是软件架构分析的重要概念，它用于描述和衡量软件系统的各种质量特性，如可靠性、可维护性、安全性等。软件架构评估则是一种重要的分析方法，它用于检查和评估软件架构的各种质量特性。本部分的学习让我更加深入地了解了软件架构分析的方法和技术，同时也认识到了软件质量属性的重要性。

第三部分：软件架构设计

第三部分主要介绍了软件架构设计的过程和技术。其中，软件架构设计的过程包括架构设计目标的确定、架构风格的选择、模块化设计、接口设计和架构文档的编写等。模块化设计是软件架构设计的重要技术，它用于将系统划分为一些独立的、可重用的模块，使得系统更加易于维护和扩展。本部分的学习让我更加系统地了解了软件架构设计的过程和技术，同时也认识到了模块化设计的重要性。

第四部分：软件架构实践

第四部分主要介绍了软件架构实践的各种技术和工具。其中，软件架构文档是软件架构实践的重要工具，它用于记录和传递软件架构信息。架构重构则是一种重要的实践技术，它用于改进和优化现有的软件架构。本部分的学习让我更加深入地了解了软件架构实践的各种技术和工具，同时也认识到了软件架构文档和架构重构的重要性。

总体来说，《Software Architecture in Practice，3rd Edition》是一本非常好的软件架构实践教材，它系统地介绍了软件架构的基础、分析、设计和实践等方面的知识，为读者提供了一个全面而深入的学习体验。在阅读过程中，我深刻认识到了软件架构在软件开发中的重要性，并对软件架构的设计、分析和实践等方面有了更深入的理解和认识。

8.2.2 会议 INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE 的相关文章

https://icsa-conferences.org/2023/call-for-papers/engineering-track-papers/

Sune Chung Jepsen, Bende Siewertsen and Torben Worm A Reconfigurable Industry 4.0 Middleware Software Architecture

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10092655

这篇论文介绍了一个基于可重构质量属性场景（QAS）开发的可重构工业 4.0 中

间件软件架构。文章探讨了灵活的工业 4.0 生产系统的重要性以及可重构中间件如何支持这些系统。论文的结构如下：第二部分概述了研究问题和研究方法；第三部分描述了该领域的类似工作以及作者的贡献；第四部分介绍了一个生产重新配置用例，并在此基础上提出了一个可重构性 QA 需求；第五部分介绍了提出的可重构中间件软件架构设计；第六部分对在 I4.0 实验室中使用实际设备评估所提出的中间件进行了评估，并根据所述 QA 需求对结果进行了分析。

工业 4.0 是指第四次工业革命，它是数字化和自动化技术在制造业中的应用。它包括物联网、云计算、大数据、人工智能等技术，旨在实现生产过程的智能化和自动化。 工业 4.0 对生产系统非常重要，因为它可以提高生产效率、降低成本、提高产品质量，并且可以使生产过程更加灵活和可持续。通过使用工业 4.0 技术，企业可以更好地适应市场需求的变化，并更快地推出新产品。可重构中间件软件架构可以支持灵活的生产系统，因为它可以使生产系统更容易地进行重新配置。在生产过程中，可能需要对生产线进行调整或更改以适应市场需求的变化。可重构中间件软件架构可以帮助实现这种重新配置，因为它具有灵活性和可扩展性，可以根据需要添加或删除组件，并且可以自动适应新的环境和要求。这样，企业就能够更快地响应市场需求，并且能够更加灵活地管理其生产过程。

其中第五部分介绍了所提出的可重构中间件软件架构设计。该架构设计旨在支持灵活生产系统，包括生产重构和重新配置。该架构由多个组件组成，这些组件可以根据需要进行重新配置和重组合。本文还提供了详细的架构视图和图表，以展示中间件组件之间的交互方式和关系。此外，本文还讨论了如何使用该架构来支持不同类型的生产系统，并提供了一些实例来说明如何使用该架构来实现灵活性和可重构性。

8.2.3 其它部分

软件体系结构的模式

在学习过程中，我还学习了许多软件体系结构的模式。这些模式包括分层、客户端-服务器、MVC（Model-View-Controller）等。学习这些模式可以帮助开发人员更好地理解软件体系结构的设计，以及如何在实际项目中应用它们。

其中 MVC（Model-View-Controller）是一种常用的软件体系结构，用于将软件系统的用户界面、业务逻辑和数据分离开来，以便于开发、维护和测试。下面详述 MVC 架构的三个部分：

模型（Model）

模型是 MVC 架构中的数据部分，表示应用程序的数据和业务逻辑。它独立于用户界面和控制器，可以处理和管理数据、查询、验证、更新等操作，通常包括数据库、数据模型、数据处理等。模型层也可以定义一些公共的接口，供视图和控制器进行调用。

视图（View）

视图是 MVC 架构中的用户界面部分，表示用户和应用程序的交互界面。它显示应用程序的数据和状态，响应用户的操作和输入，通常包括界面设计、页面布局、样式等。视图层不负责处理业务逻辑和数据存储，只负责展示数据，提供用户与系统的交互。

控制器（Controller）

控制器是 MVC 架构中的逻辑部分，表示应用程序的业务逻辑。它接受用户输入和事件，负责处理业务逻辑和数据流，然后根据处理结果选择合适的视图进行显示。控制器层通常是模型和视图之间的中介，通过将用户请求转发给模型和视图来协调系统的工作流程。

在 MVC 架构中，这三个部分之间是相互独立的，每个部分都有自己的职责和功能。模型负责数据的存储和处理，视图负责数据的展示和用户交互，控制器负责协调和控制整个应用程序的流程和数据传递。通过这种分离的方式，MVC 架构能够提高代码的可维护性、可扩展性和可重用性，使得软件开发更加灵活和高效。

MVC 一个著名的成功案例为 Django。Django 是一款基于 Python 编程语言的 Web 应用框架，也采用了 MVC 架构。该框架使用 ORM 作为模型层，将数据库和 Python 代码进行了很好的集成，同时采用了视图和控制器的分离，实现了快速开发和高

效管理 Web 应用程序的功能。Django 在 Python 社区中非常受欢迎，被广泛应用于 Web 开发、数据分析、人工智能等领域。

MVC 架构作为一种经典的软件体系结构，已经被广泛应用于各种领域的软件开发中。随着技术的发展和需求的变化，MVC 架构也在不断演进和发展。以下是 MVC 架构未来发展的一些方向：

前后端分离

前后端分离是一种新的软件开发方式，它将前端和后端进行了分离，使得 Web 应用开发更加灵活和高效。在前后端分离的架构下，MVC 架构可以将前端和后端进行更加清晰的分离，使得应用程序更加易于维护和扩展。

RESTful API

RESTful API 是一种基于 HTTP 协议的轻量级的 Web 服务架构，它将资源和操作进行了统一的抽象，并采用标准的 HTTP 方法进行操作。在 RESTful API 的架构下，MVC 架构可以更好地支持 Web 服务的开发和管理，使得应用程序更加易于扩展和维护。

云原生

云原生是一种新的软件架构方式，它将应用程序设计为基于云的分布式架构，使得应用程序可以更好地利用云的优势。在云原生的架构下，MVC 架构可以更好地支持应用程序的分布式部署和管理，同时还可以更好地支持应用程序的容错性和可伸缩性。

8.3 谢骁飏的学习报告

在阅读了 An Introduction to Software、Architecture Software Architecture in Practice、Software Architecture-a Roadmap 这三篇文章后，结合我们软件开发的具体工作，我做了一下几点总结:

a.软件架构的概念

软件架构是指软件系统设计的基础，它负责定义软件系统各部分之间的关系，包括软件组件的类型、接口、协议和数据结构等。简单来说，软件架构就是在实现软件系统之前，对软件系统进行组织和设计的过程。

软件架构的概念在计算机科学中非常重要。通过良好的架构设计，软件系统可以更容易地被开发、维护和扩展。同时，软件架构还可以帮助开发团队更好地理解和沟通软件系统的结构。

软件架构的组成包括:

软件组件：软件系统由多个组件构成，每个组件具有独立的职责和功能。

接口：不同的组件之间需要通过接口进行交互和通信。

协议：定义不同的组件之间交互和通信的规则。

数据结构：定义软件系统中使用的数据元素、数据类型和数据格式等。

此外，软件架构还需要考虑安全性（security）、可用性（availability）、可扩展性（scalability）、性能（performance）等方面的问题。

总之，软件架构是一种有效的、高质量的设计方法，能够帮助我们创建出更具可维护性、可扩展性和可移植性的软件系统。在软件开发的过程中，理解并遵循好软件架构设计原则是非常重要的一环。

b.软件架构的先进技术

当前，软件架构方面采用了很多先进的技术和工具，这些技术和工具可以帮助开发人员更好地实现系统的核心功能，提高开发效率、性能和稳定性。以下是其中一些比较常见的技术和工具:

微服务架构：将应用程序拆分成小型的、自治的服务单元，每个服务单元都可以

独立部署和扩展。这种模式可以在复杂系统中提供更好的可伸缩性和灵活性。

容器化技术：如 Docker 和 Kubernetes 等，使得开发者可以方便地打包和部署应用程序，提高了应用程序的移植性和可维护性。

无服务架构：将代码逻辑进行拆分，并通过云平台来自动处理应用程序所需的计算和存储资源，提高了开发效率和部署效率。

事件驱动架构：基于事件和消息来组合和协调各个系统组件的操作，使系统更加容易扩展和优化。

RESTful API 和 GraphQL：这些是目前非常流行的 Web API 设计模式，使得应用程序之间的通信更为简洁和灵活。

除了上述技术外，还有一些其他的工具和框架，如 Spring 框架、React.js 等，都可以提供先进的软件架构开发和设计模式。开发者可以根据实际具体需求选择最合适的技术和工具。

c.当前软件架构发展的方向和趋势

云原生：随着云计算技术的不断成熟和普及，云原生成为了当前软件架构的重要趋势。云原生将应用程序打包成可轻松部署的容器，并通过以 Kubernetes 为代表的容器编排工具进行管理，以更好地发挥云计算的优势。

微服务：微服务架构作为一种分布式系统设计方法被广泛使用。它将应用程序划分成小型的、自治的服务单元，每个服务单元都可以独立开发、测试和部署。微服务架构能够让开发团队高效协作、快速迭代新功能、提高系统的扩展性和灵活性。

无服务器：无服务器架构借助云端提供的功能，使得开发者无需关心基础设施的

维护和扩展问题，只需要关注核心业务逻辑实现即可。无服务器架构目前已经在许多领域得到了应用，包括 Web、Mobile、IoT 等。

事件驱动：事件驱动架构在异步、分布式系统中有广泛应用，每一个操作都由一个事件触发，并且事件之间以消息作为通信手段。它使得系统解耦、易扩展和高性能。

智能系统：人工智能在当前的软件架构中也得到了广泛应用，如机器学习和深度学习技术等。通过将人工智能引入系统设计中，可以提高系统的自动化和智能化水平，使得系统更加自适应和优化。

可观察性：可观察性是一个系统的实时运行状态对开发者、用户和管理者来说是可知的能力，这种能力就是通过各个维度或指标进行监测和分析实现的。日志收集、指标监控和分布式追踪等技术被广泛应用于系统的管理和诊断。

总之，软件架构领域发展快速，越来越注重提高系统的安全性、可用性、灵活性、性能和扩展性。未来软件架构将会更加注重云原生、可观察性、分布式一体化、超稳定性等方面的发展。

d.如何将软件架构的技术有效地应用到软件开发中

了解业务需求：首先需要深入了解业务需求和用户需求，以便确定所使用的架构模式和技术。从客户、用户、安全、性能、可维护性、扩展性等多个方面考虑。

设计架构：在了解业务需求之后，需要对系统进行设计，并选择适合的架构模式。例如，如果需要高度可靠性，可以选择集群部署；如果需要高可用性，可以选择微服务架构等。

实现开发：在架构设计确定之后，根据设计规范实现代码和系统构建。在这个过

程中一定要充分运用好所选用的架构模式和技术，确保符合架构设计的规范。

测试校验：当开发完成后，对系统进行一系列测试，如单元测试、集成测试、性能测试、安全测试等。通过测试，不仅可以发现问题并及时修复，还可以验证系统是否符合架构规范。

迭代升级：随着业务的发展和环境的变化，系统的需求也会改变，这时需要对架构进行持续迭代升级。同时需要对新技术和新平台保持关注，适时地引入和升级系统。

综上所述，要将软件架构的技术有效地应用到软件开发中，需要全面考虑业务需求和系统设计，主要就是要在三个方面进行落实：指导整个开发流程、规范代码编写和设计开发流程。可以帮助开发者更好地了解和公司内部其他同事协作。

8.4 刘钊的学习笔记

在阅读了 Software Architecture in Practice 并且查阅了新的有关软件体系结构的资料之后，我对事件驱动架构这一新的软件体系结构进行了学习并做了下面的学习笔记。

8.4.1 什么是事件驱动架构

事件驱动架构（Event-Driven Architecture，EDA）是一种基于事件的异步消息传递机制，通过事件驱动模型来实现软件系统的解耦、弹性和可扩展性的软件架构。它强调事件的产生和消费，事件可以是任何重要的状态变化，例如用户的行为、业务流程、系统性能等。

在事件驱动架构中，事件的产生和处理是异步的，即事件的产生者和消费者

不需要直接通信，而是通过事件总线来进行消息的传递和处理。事件总线是整个系统的核心，它负责接收和分发事件，将事件发送给相应的处理程序。

事件驱动架构的核心思想是解耦，将不同的模块解耦开来，使得系统更加灵活和可扩展。同时，事件驱动架构可以提高系统的弹性和容错能力，因为一个事件的处理失败并不会影响到整个系统，只会影响到相关的部分。

事件驱动架构的应用非常广泛，特别是在大型分布式系统和云原生应用中。它可以应用于各种不同的场景，例如实时数据处理、物联网、实时通信、日志处理、消息队列等。在实际应用中，可以使用多种技术来实现事件驱动架构，例如消息队列、流处理、复杂事件处理、分布式事务等。

8.4.2 事件驱动架构的例子

一个常见的事件驱动架构的例子是在线电商网站。在这个系统中，可以有多个事件，例如用户购买商品、取消订单、评价商品等。这些事件会被不同的服务所处理，例如订单服务、商品服务、评价服务等。每个服务会注册自己关心的事件类型，并将事件处理程序注册到事件总线上。

当用户购买商品时，订单服务会接收到购买事件，并将订单信息保存到数据库中。同时，商品服务会接收到购买事件，并减少商品库存。当用户取消订单时，订单服务会接收到取消订单事件，并将订单状态更新为取消状态。如果用户评价商品，评价服务会接收到评价事件，并将评价信息保存到数据库中。

通过事件驱动架构，不同的服务可以解耦开来，每个服务只需要关注自己的业务逻辑，并通过事件总线来进行消息的传递和处理。这样可以使系统更加灵活和可扩展，同时提高系统的弹性和容错能力。

8.4.3 事件驱动架构两种拓扑结构

事件驱动架构模式包含了两种主要的拓扑结构：中介(mediator)拓扑结构和代理(broker)拓扑结构。 mediator 拓扑结构通常在你需要在事件内使用一个核心中介分配、协调多个步骤间的关系、执行顺序时使 用；而代理拓扑结构则在你想要不通过一个核心中介将多个事件串联在一起时使用。

中介拓扑结构适合用于拥有多个步骤，并需要在处理事件时能通过某种程度的协调将事件分层的场景

在中介拓扑结构中主要有四种组件：事件队列（event queue），事件中介，事件通道（event channel），和 事件处理器（event processor）。当事件流需要被处理，客户端将一个事件发送到某个事件队列中，由消息队列将其运输给事件中介进行处理和分发。事件中介接收到该消息后，并通过将额外的异步事件发送给事件通道，让事件通道执行该异步事件中的每一个步骤，使得事件中介能够对事件进行分配、协调。同时，又因为事件处理器是事件通道的监听器，所以事件通道对异步事件的处理会触发事件处理器的监听事件，使事件 处理器能够接收来自事件中介的事件，执行事件中具体的业务逻辑，从而完成对传入事件的处理。

代理拓扑结构与中介拓扑结构不同之处在于：代理拓扑结构中没有核心的事件中介；相反，事件流在代理拓 扑结构中通过一个轻量的消息代理（例如：ActiveMQ, HornetQ，等等……）将消息串联成链状，分发至事件处理器组件中进行处理。代理扑结构适用的使用场景大致上具有以下特征：你的事件处理流相对来说比较简单，而且你不想（不需要）使用核心的事件分配、调节机制以提高你处理事件的效率。在代理拓扑结构中主要包括两种组件：代理和事件处理器。代理可被集中或相互关联在一起使用，此外，代理中还可以包含所有事件流中使用的事件通道。

8.4.4 事件驱动架构的分析

1. 整体灵活性高

整体灵活性用于评价架构能否在不断改变的使用场景下快速响应，因为事件处理器组件使用目的单一、高度解耦、与其他事件处理器组件相互独立，不相关联，那么发生的改变对一个或多个事件处理器来说普遍都是独立的，使得对改变的反馈非常迅速，不需要依赖其他事件处理器的响应作出处理。

2. 易于部署

总的来看，事件驱动架构模式由于其高度解耦的事件处理器组件的存在，对事件的部署相对来说比较容易，而使用代理拓扑结构比使用中介拓扑结构进行事件调度会更容易一些，主要是因为在中介拓扑结构中事件处理器与事件中介紧密地耦合在一起：事件处理器中发生改变后，事件中介也随之改变，如果我们需要改变某个被处理的事件，那么我们需要同时调度事件处理器和事件中介。

3. 可测试性低

虽然在事件驱动架构模式中进行单元测试并不困难，但如果我们要进行单元测试，我们就需要某种特定的测试客户端或者是测试工具产生事件，为单元测试提供初始值。此外，由于事件驱动架构模式是异步进行事件分发的，其异步处理的特性也为单元测试带来了一定的困难。

4. Performance 性能高

对消息传递的架构可能会让设计出来的事件驱动架构的表现不如我们的期望，但通常来说，该模式都能通过其异步处理的特性展示优秀的性能表现；换句话来说，高度解耦，异步并行操作大大减少了传递消息 过程中带来的时间开销。

5. 伸缩性高

事件驱动架构中的高度解耦、相互独立的事件处理器组件的存在，使得可拓展性成为该架构与生俱来的优点。架构的这些特定使得事件处理器能够进行细粒度的拓展，使得每一个事件处理器都能单独被拓展，而不影响其他事件处理器。

6.不易于开发

由于使用事件驱动架构进行开发需要考虑其异步处理机制、协议创建流程，并且开发者需要用代码为事件处理器和操作失败的代理提供优秀的错误控制环境，无疑使得用事件驱动架构进行开发会比使用其他架构进行开发要困难一些。

8.5 黄德业的学习笔记

在阅读了 Software Architecture in Practice 并且查阅了新的有关软件体系结构的资料之后，我对微服务架构这一新的软件体系结构进行了学习并做了下面的学习笔记。

8.5.1 什么是微服务架构

微服务架构（Microservices Architecture）是一种软件架构风格，它将应用程序拆分为一组小型、自治的服务。每个服务都专注于完成特定的业务功能，并通过轻量级通信机制进行相互交互，例如使用 HTTP/REST、消息队列等。

在微服务架构中，应用程序被拆分为多个独立部署的服务单元，每个服务单元都有自己的数据库和业务逻辑。这种拆分使得每个服务单元可以独立开发、测试、部署和扩展，而不会对其他服务产生影响。每个服务单元都是相对较小的，易于理解、维护和修改。

微服务架构可以带来许多优势，如更好的可扩展性、灵活性、独立性和团队协作等，但也需要面对一些挑战，例如服务之间的通信复杂性、数据一致性和系统监控等。因此，在采用微服务架构时需要仔细考虑设计和管理方面的问题。

8.5.2 微服务架构的主要特点

1.拆分与自治性：应用程序按业务功能进行拆分成多个服务，每个服务都可以独立进行开发、部署和扩展，并具有高度自治性。

2.独立部署：每个服务都可以独立部署，这意味着可以灵活地对某个服务进行更新、扩展或替换，而无需对整个应用进行重新部署。

3.松耦合：微服务之间通过明确定义的接口进行通信，彼此之间松耦合，这样可以实现更好的独立性和灵活性。

4.技术多样性：不同的微服务可以使用不同的技术栈和编程语言，以满足特定需求和技术偏好。

5.可扩展性：由于每个服务都是独立部署和扩展的，因此可以根据需要对每个服务进行独立的水平扩展，以应对不同的负载需求。

6.容错性：由于微服务是相互独立的，如果某个服务发生故障或出现问题，其他服务仍然可以继续运行，从而提高系统的容错性和可用性。

8.5.3 微服务架构的应用

微服务架构在许多领域和应用中都得到广泛应用。以下是一些微服务架构的典型应用场景：

1.电子商务平台：微服务架构在电子商务领域特别受欢迎。通过将用户管理、商品管理、购物车、订单处理、支付等核心功能拆分为独立的微服务，电子商务平台可以实现高度可扩展性、灵活性和独立性，同时也更容易进行团队协作和快速迭代。

2.在线旅游平台：在线旅游平台（如酒店预订、机票预订等）通常需要处理多个复杂的业务功能，如搜索、预订、支付、配送等。通过微服务架构，可以将这些功能拆分为独立的服务，使得系统更易于扩展、维护和升级。

3.社交媒体应用：社交媒体应用（如社交网络、即时通讯等）需要处理大量的用户交互和实时通信。通过使用微服务架构，可以将用户管理、消息推送、社交关系、内容管理等功能拆分为独立的服务，以实现高性能、可扩展性和实时性。

4.物流和配送平台：物流和配送平台需要管理订单、配送路线、库存等复杂的业务流程。通过微服务架构，可以将这些功能模块拆分为独立的服务，使得系统更具弹性和可扩展性，并支持实时的物流追踪和配送调度。

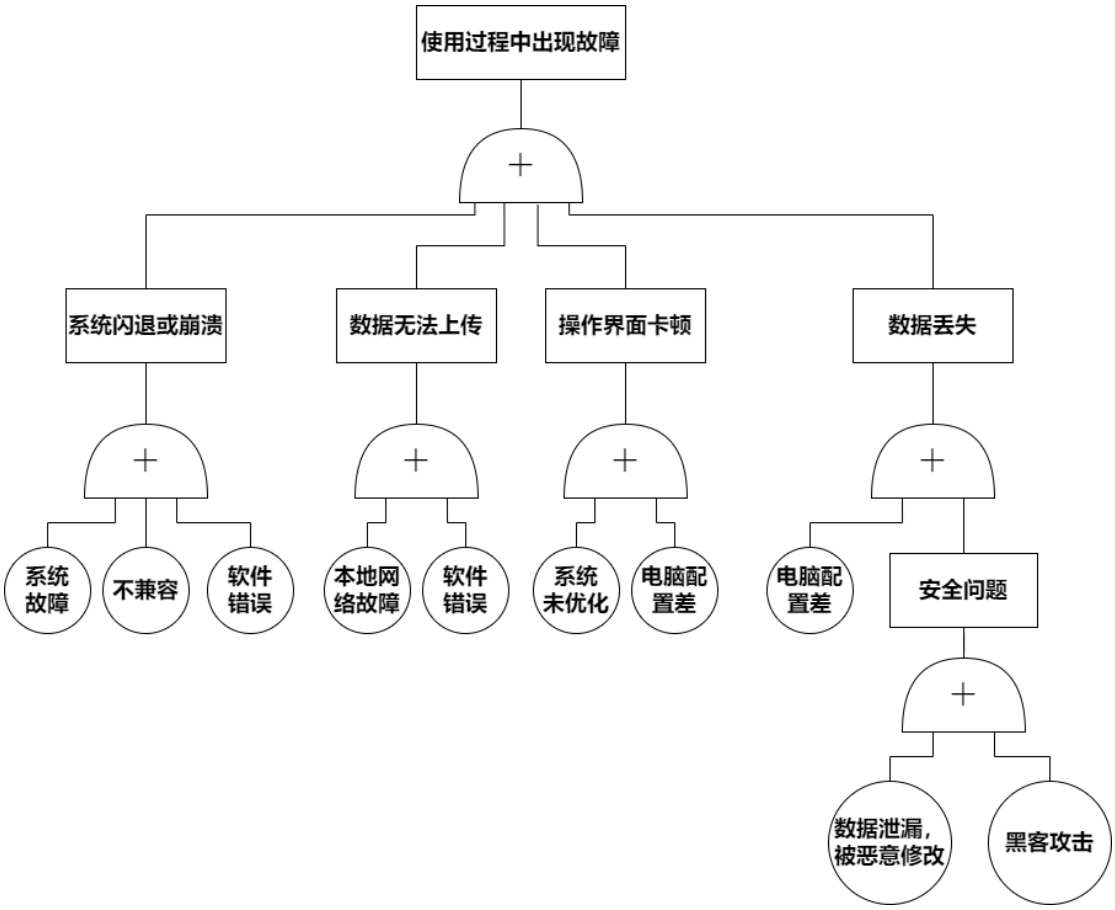
5.金融服务：金融领域的应用（如支付、财务管理、交易处理等）需要高度的可靠性和安全性。通过微服务架构，可以将不同的金融功能拆分为独立的服务，从而实现模块化、可伸缩和容错的系统架构。

6.云计算和 Saas 平台：云计算和软件即服务（Saas）平台需要处理多个租户和大规模的用户请求。通过微服务架构，可以将不同的功能模块拆分为独立的服务，以实现弹性扩展、隔离性和定制化的服务。

9 附录 2 本项目的故障树与割集树

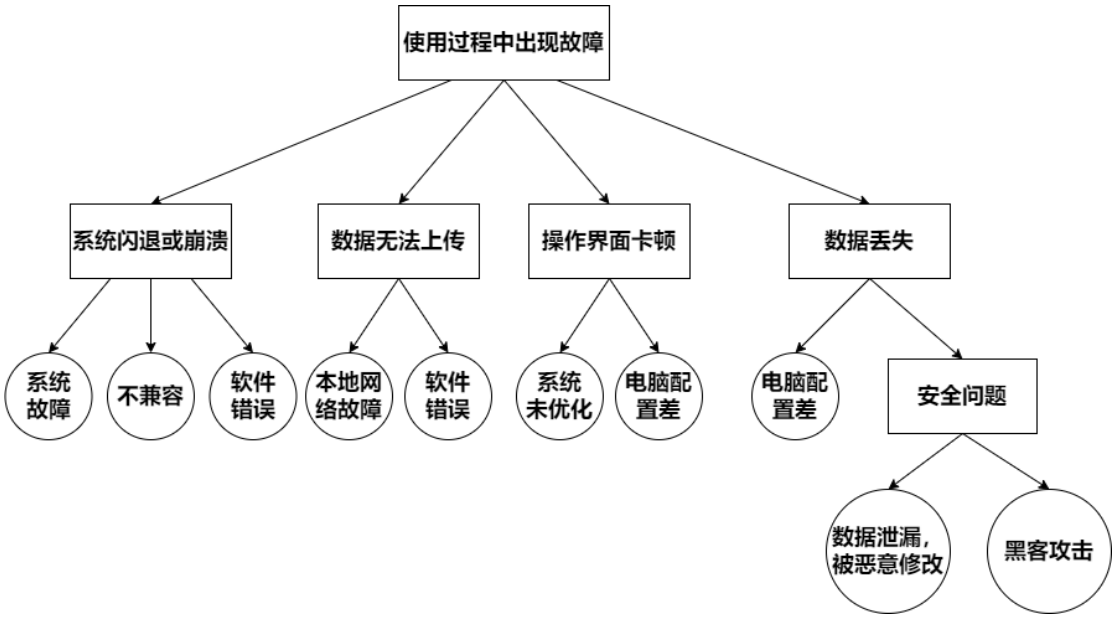
9.1 故障树

本项目的故障树图如下：



9.2 割集树

本项目的割集树图如下：



10 附录 3 课本习题的割集树

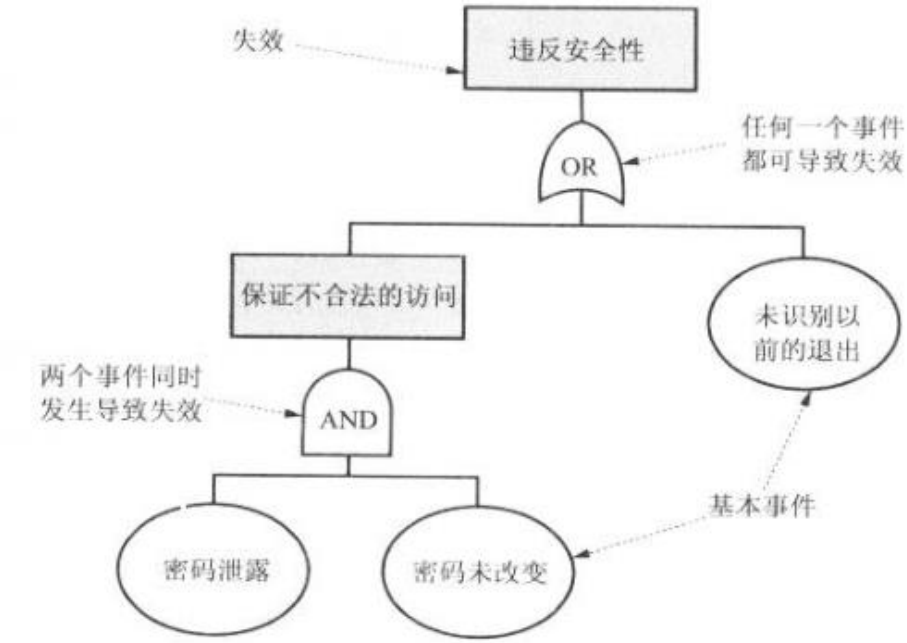


图5-11 违反安全性的故障树

转化为割集树: