软件体系结构设计说明（SAD）

分工：本文档的整体架构以及目录设计由孟小凡(组长)完成，不同部分的编写由组内每位成员根据自己在项目中所负责的内容进行选择，具体分配如下(编写部分对应数字涵盖该章节下的所有内容，例如某位成员编写3.1部分，那么该部分中的3.1.n这样的小章节也由该成员完成)：孟小凡编写本文档的1引言部分、2引用文件部分、3.3体系结构的评估和改进部分；尹国泰编写本文档的3.2体系结构部分、4需求可追踪性下的追踪矩阵(4部分的文字说明沿用SRS中的文档内容)；吴百川编写本文档的3.4全局数据说明部分、3.5体系结构设计评审部分；黄泰达编写本文档的3.1软件体系结构风格部分、5注解部分。实验十内容中的学习报告附在本文档最后，各成员撰写的学习报告前均有对应作者名字。

**目录**

[软件体系结构设计说明（SAD） 1](#_Toc134729923)

[1 引言 3](#_Toc134729924)

[1.1 标识 3](#_Toc134729925)

[1.2 系统概述 3](#_Toc134729926)

[1.3 文档概述 3](#_Toc134729927)

[1.4 基线 3](#_Toc134729928)

[2 引用文件 3](#_Toc134729929)

[3 CSCI体系结构设计 3](#_Toc134729930)

[3.1 软件体系结构风格 3](#_Toc134729931)

[3.1.1 各类风格与项目系统的适配比较 4](#_Toc134729932)

[3.1.2 体系结构风格选取 5](#_Toc134729933)

[3.2 体系结构 6](#_Toc134729934)

[3.2.1 程序单元 6](#_Toc134729935)

[3.2.2 程序层次结构关系 7](#_Toc134729936)

[3.3 体系结构的评估和改进 8](#_Toc134729937)

[3.3.1 故障树分析 8](#_Toc134729938)

[3.3.2 结构改进 10](#_Toc134729939)

[3.4 全局数据说明 11](#_Toc134729940)

[3.4.1 常量 11](#_Toc134729941)

[3.4.2 变量 11](#_Toc134729942)

[3.4.3 数据结构 11](#_Toc134729943)

[3.5 体系结构设计评审 12](#_Toc134729944)

[3.5.1 Validation 12](#_Toc134729945)

[3.5.2 Verification 12](#_Toc134729946)

[4需求可追踪性 12](#_Toc134729947)

[5注解 14](#_Toc134729948)

[学习报告 14](#_Toc134729949)

# 1 引言

本部分简要说明本文档的整体结构划分，同时初步介绍本文档向读者展示有关本组项目系统对应软件体系结构设计的内容。

## 1.1 标识

本文档标识为人才招聘系统的软件体系结构设计说明书（SAD）。

## 1.2 系统概述

本组所开发的“人才招聘系统”具备完善的体系结构，并且通过搜集资料、网络调研以及组内讨论，我们对于系统整体架构相关的各模块内容进行了详细划分，按照软件体系结构设计的标准对项目系统进行了更具结构化地重构与开发。

## 1.3 文档概述

本文档为针对本组开发项目“人才招聘系统”软件体系结构设计说明书，具体讲解了软件在体系结构上采用的架构风格、程序的模块化、系统可能产生的故障以及相应改进与数据说明和系统评估等，基本遵从SAD中涉及到的对应内容，并展示了本组开发项目独特的关键点。

## 1.4 基线

本文档的基线版本为1.0。

# 2 引用文件

本文档的编写引用了以下参考文献：

·《Logic in Computer Science-Modelling and Reasoning about Systems》[英]Michael Huth、Mark Ryan著 – *CAMBRIDGE UNIVERSITY PRESS*

·《Software Architecture: a Roadmap》[美]David Garlan撰 – *School of Computer Science Carnegie Mellon University*

·《Software Architecture in Practice》（Third Edition）Len Bass、Paul Clements、Rick Kazman著 – *Addison-Wesley*

# 3 CSCI体系结构设计

本部分为当前文档介绍软件体系结构设计相关内容的关键模块，涉及内容包括开发系统采用的风格、涉及的体系结构、进行的评估与改进、对关键数据的说明以及对整体架构涉及的评审。

## 3.1 软件体系结构风格

本部分介绍组内对于各类软件体系结构风格的调研并给出这些风格与本组开发项目的适配程度，通过比较决策风格的选取并分析采用原因。

### 3.1.1 各类风格与项目系统的适配比较

对于互联网招聘软件，不同的体系结构风格都有其适用性和优缺点。数据流风格和管道-过滤器风格适合处理大量数据和流式数据，层次结构风格适合构建复杂系统，C/S风格和B/S风格则适合实现跨平台和跨设备的Web应用程序。在选择体系结构风格时，需要根据具体的需求和场景来做出选择，权衡各种因素，以达到最佳的设计效果。

本组成员对于各类风格与开发的“人才招聘系统”适配打分表如下：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 吴百川 | 适配 | 优先级 | 数据流风格 | 管道-过滤器风格 | 层次结构风格 | C/S风格 | B/S风格 |
| 多元功能设计 | 2 | 1 | 5 | 5 | 5 | 5 |
| 算法修改 | 1 | 1 | 4 | 4 | 2 | 5 |
| 系统性能 | 3 | 5 | 4 | 4 | 3 | 3 |
| 安全性 | 4 | 5 | 5 | 4 | 5 | 4 |
| 交互性 | 5 | 1 | 1 | 3 | 4 | 5 |
| 总计 |  | 43 | 51 | 57 | 61 | 65 |
|  |  |  |  |  |  |  |  |
| 孟小凡 | 适配 | 优先级 | 数据流风格 | 管道-过滤器风格 | 层次结构风格 | C/S风格 | B/S风格 |
| 多元功能设计 | 2 | 3 | 5 | 2 | 2 | 2 |
| 算法修改 | 1 | 4 | 4 | 4 | 1 | 2 |
| 系统性能 | 2 | 5 | 4 | 4 | 3 | 3 |
| 安全性 | 3 | 3 | 5 | 3 | 4 | 5 |
| 交互性 | 5 | 1 | 1 | 2 | 5 | 5 |
| 总计 |  | 34 | 42 | 35 | 48 | 52 |
|  |  |  |  |  |  |  |  |
| 尹国泰 | 适配 | 优先级 | 数据流风格 | 管道-过滤器风格 | 层次结构风格 | C/S风格 | B/S风格 |
| 多元功能设计 | 2 | 1 | 2 | 3 | 4 | 5 |
| 算法修改 | 1 | 2 | 5 | 4 | 3 | 5 |
| 系统性能 | 3 | 3 | 4 | 5 | 3 | 3 |
| 安全性 | 4 | 4 | 5 | 4 | 5 | 3 |
| 交互性 | 5 | 2 | 2 | 3 | 5 | 4 |
| 总计 |  | 39 | 51 | 56 | 65 | 56 |
|  |  |  |  |  |  |  |  |
| 黄泰达 | 适配 | 优先级 | 数据流风格 | 管道-过滤器风格 | 层次结构风格 | C/S风格 | B/S风格 |
| 多元功能设计 | 2 | 4 | 3 | 2 | 2 | 4 |
| 算法修改 | 2 | 3 | 4 | 4 | 3 | 2 |
| 系统性能 | 3 | 5 | 5 | 3 | 4 | 3 |
| 安全性 | 5 | 2 | 3 | 3 | 5 | 4 |
| 交互性 | 4 | 2 | 2 | 5 | 4 | 4 |
| 总计 |  | 47 | 52 | 56 | 63 | 57 |

### 3.1.2 体系结构风格选取

基于各类风格的特点与分析，组内对不同风格比较如下：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 数据流风格 | 管道-过滤器风格 | 层次结构风格 | C/S风格 | B/S风格 |
| 吴百川 | 43 | 51 | 57 | 61 | 65 |
| 孟小凡 | 34 | 42 | 35 | 48 | 52 |
| 尹国泰 | 39 | 51 | 56 | 65 | 56 |
| 黄泰达 | 47 | 52 | 56 | 63 | 57 |
| 平均值 | 40.75 | 49 | 51 | 59.25 | 57.5 |

显然，组内一致同意C/S风格较为适配我们组的项目，就C/S风格进行更深入的研究：

优势：

·高效稳定：通过将业务逻辑集中在服务器端处理，可以降低客户端的负担，提高系统的稳定性和效率。在高并发的场景下，可以通过增加服务器数量和优化服务器性能等方式提高系统的响应速度和并发处理能力。

·可扩展性：C/S架构可以更好地支持系统的水平扩展。当用户增加时，可以通过增加服务器数量来扩展系统，这有助于提高系统的并发处理能力和稳定性。

·易于维护：将业务逻辑分离到服务器端，可以降低客户端的复杂度，提高代码的可维护性。在出现问题时，也更容易定位和解决。

劣势：

·服务器成本高：相对于B/S架构，C/S架构需要更多的服务器资源，需要承担更高的服务器成本。

·客户端依赖性强：C/S架构需要客户端软件提供用户界面和交互功能，因此客户端软件需要用户主动安装和更新。对于一些用户来说，可能会遇到客户端软件兼容性和更新等问题。

·安全性较低：由于客户端可以请求服务器提供的各种服务，因此存在一定的安全隐患。需要采取一系列的安全措施来保障用户数据的安全性。

对于互联网招聘系统，C/S架构可以支持用户交互和后台服务的分离，使得系统更加清晰明确，同时也能够更好地支持扩展和维护。但是需要注意客户端依赖性较强，需要对客户端进行维护和升级，以及加强安全性保护，避免敏感信息泄露等问题。

## 3.2 体系结构

本部分采用模块化方法对开发的软件系统各模块涉及的代码程序进行单元功能介绍以及预计目录展示，同时给出程序间的层次结构，给出各分层的依赖关系以及调用过程。

### 3.2.1 程序单元

我们的“人才招聘系统”分为求职者、企业、管理员、后端四个部分，使用java+spring框架编写，各模块所需要编写的程序预计如下：

1.求职者部分：

ApplyController.java：求职者相关的控制器，处理求职者信息的请求。

ApplyInfo.java：求职者信息的实体类，用于存储和传递求职者信息。

ApplyServiceImpl.java：求职者相关的服务实现类，处理求职者信息的业务逻辑。

apply.jsp：前端页面，用于显示求职者信息、收集求职者信息并提交到服务器。

2.企业部分

RecruitController.java：企业相关的控制器，处理企业信息的请求。

RecruitInfo.java：企业信息的实体类，用于存储和传递企业信息。

RecruitServiceImpl.java：企业相关的服务实现类，处理企业信息的业务逻辑。

insert.jsp：前端页面，用于添加企业信息。

show.jsp：前端页面，用于显示企业信息。

update.jsp：前端页面，用于更新企业信息。

3.管理员部分

EmployeeController.java：管理员相关的控制器，处理管理员信息的请求。

Employee.java：管理员信息的实体类，用于存储和传递管理员信息。

EmployeeServiceImpl.java：管理员相关的服务实现类，处理管理员信息的业务逻辑。

appshow.jsp：前端页面，用于显示管理员信息。

info.jsp：前端页面，用于添加或修改管理员信息。

4.后端部分

IApplyDao.java：数据访问层接口，提供访问求职者信息的方法。

ApplyDaoImpl.java：数据访问层的具体实现类，实现IApplyDao接口中的方法。

IRecruitDao.java：数据访问层接口，提供访问企业信息的方法。

RecruitDaoImpl.java：数据访问层的具体实现类，实现IRecruitDao接口中的方法。

IEmployeeDao.java：数据访问层接口，提供访问管理员信息的方法。

EmployeeDaoImpl.java：数据访问层的具体实现类，实现IEmployeeDao接口中的方法。

IApplyService.java：服务层接口，提供处理求职者信息的方法。

IRecruitService.java：服务层接口，提供处理企业信息的方法。

IEmployeeService.java：服务层接口，提供处理管理员信息的方法。

jdbc.properties：数据库配置文件，包含数据库的连接信息。

applicationContext.xml：Spring框架配置文件，配置Spring框架的相关组件。

spring-mvc.xml：Spring MVC框架配置文件，配置Spring MVC框架的相关组件。

web.xml：Web应用程序部署描述文件，用于配置Web应用程序的相关属性。

### 3.2.2 程序层次结构关系

在“人才招聘系统”中，程序之间的层次关系可以简单概括为：web层调用service层，service层调用dao层，dao层访问数据库并返回数据给service层。

具体地，以下是各程序之间的层次关系和调用关系：

1.Web层程序：

·ApplyController.java

·EmployeeController.java

·RecruitController.java

这三个程序是web层的控制器，负责接收用户请求，调用service层的程序处理请求，并将结果返回给用户。

2.Service层程序：

·ApplyServiceImpl.java

·EmployeeServiceImpl.java

·RecruitServiceImpl.java

这三个程序是service层的实现类，包含业务逻辑，通过调用dao层的程序来访问数据库，然后将结果返回给web层控制器。

3.Dao层程序：

·ApplyDaoImpl.java

·EmployeeDaoImpl.java

·RecruitDaoImpl.java

这三个程序是dao层的实现类，用于访问数据库，实现数据的增删改查操作，并将结果返回给service层程序。

程序的调用关系为：

[1].Web层调用Service层的程序：

ApplyController.java、

EmployeeController.java、RecruitController.java分别调用

ApplyServiceImpl.java、

EmployeeServiceImpl.java、RecruitServiceImpl.java

[2].Service层调用Dao层的程序：

ApplyServiceImpl.java调用ApplyDaoImpl.java；

EmployeeServiceImpl.java调用EmployeeDaoImpl.java；

RecruitServiceImpl.java调用RecruitDaoImpl.java。

因此，在“人才招聘系统”中，Web层的控制器负责接收用户请求，并调用Service层的程序，Service层程序包含业务逻辑，调用Dao层的程序来访问数据库。Dao层程序访问数据库，实现数据的增删改查操作，并将结果返回给Service层程序。最终，Service层程序将结果返回给Web层的控制器，控制器再将结果返回给用户。

## 3.3 体系结构的评估和改进

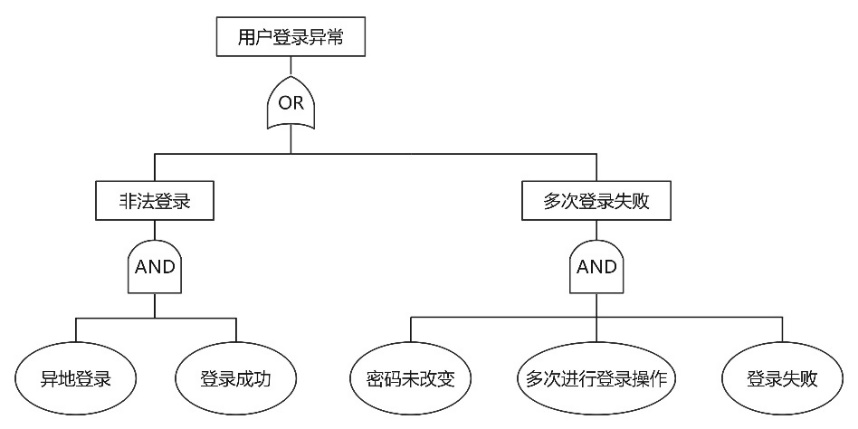
本部分对项目开发完成后软件系统成品可能出现的故障进行了预测，构建故障树并转化为割集树对这些异常进行逻辑分析，从而实现对特定故障进行体系结构上相应的改进。

### 3.3.1 故障树分析

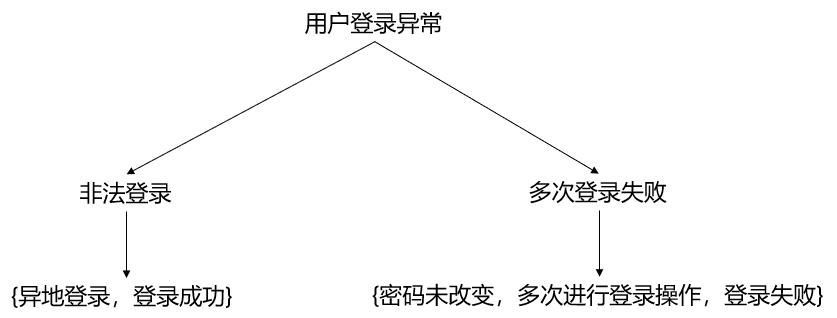
本组开发的“人才招聘系统”是基于网页的信息检索平台，它涉及到账户登录识别身份、网页的访问、数据库的调用等操作，那么必然会面临一些不可抗力带来的故障或异常，前面列举操作对应产生的故障分别为用户登录异常、网页崩溃与数据库调用错乱，下面构建三种故障树来表示，并转化对应割集树来查看对应逻辑：

[1].用户登录异常：

故障树：

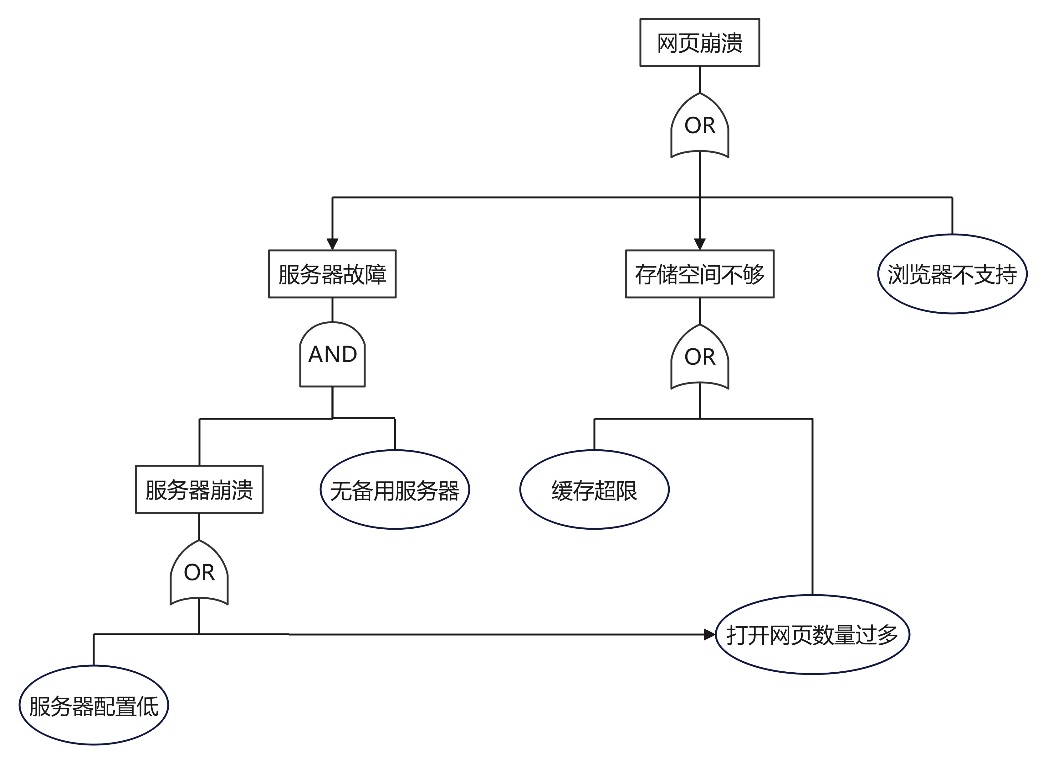


割集树：

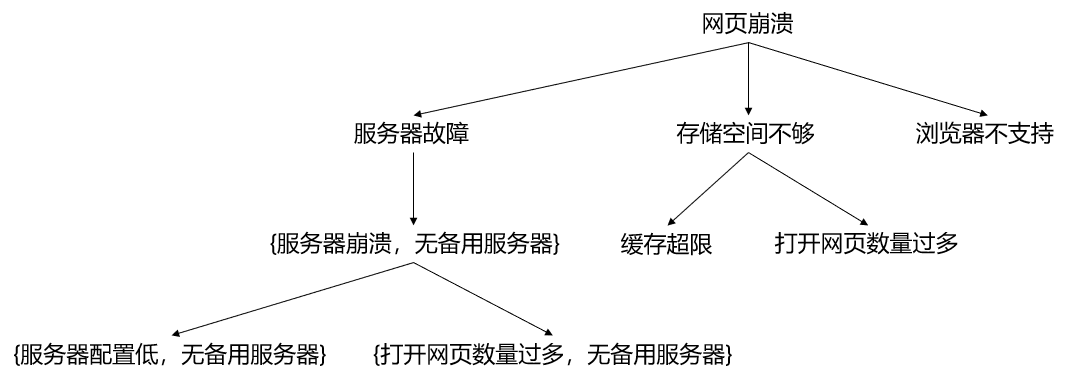


[2].网页崩溃：

故障树：

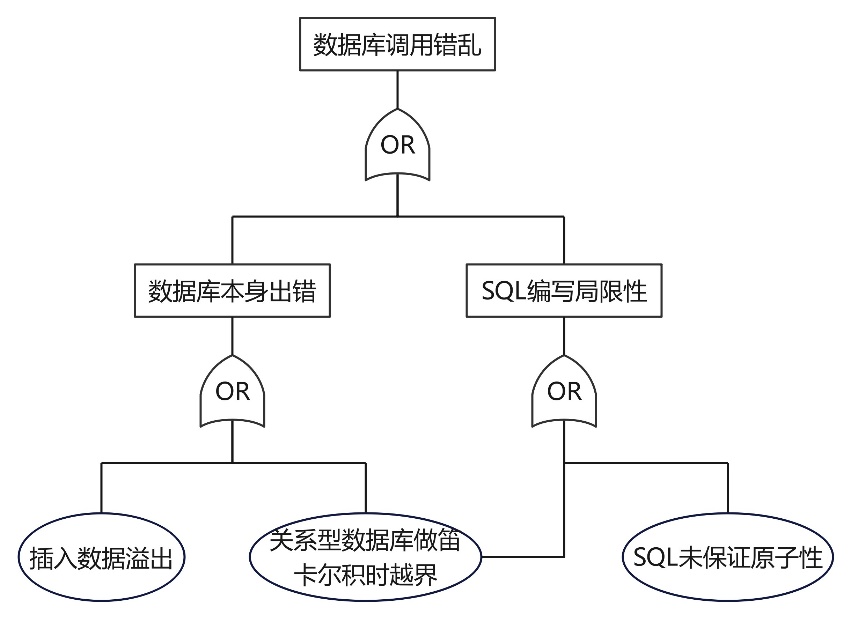


割集树：

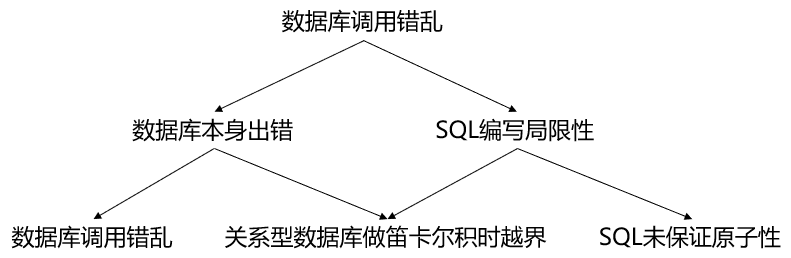


[3].数据库调用错乱：

故障树：



割集树：



### 3.3.2 结构改进

针对上面给出的三种故障异常，可以通过设置一部分全局变量或者定义一些功能函数来对系统进行改进，从而避免故障产生或形成警告：

·系统登录功能的设计上可以进行地区数据的对比，对于异地登录的情况预计采用发送验证码的方式来规避非法登录，需要编写地域对比SQL以及验证码核验程序；对于出现多次登录失败的情况可能是非法分子采用枚举尝试方法来破解密码，那么可以设计一个全局数据定义为登录失败最大次数，大于该次数时系统将锁定密码输入框，然后定义相关功能函数使登录失败次数达到该全局数值后需要通过验证码来解除对密码输入的锁定。

·首先在软件安装说明中明确系统适配的浏览器类型，避免出现浏览器不支持现象；设置一个全局数据定义为打开网页的最大数量，预计后期设计一个函数实现网页超过该数量后系统将不再打开新的网页并给出提示弹窗（网页数量过多，请关闭部分网页）的功能；服务器故障需要后端维护，对于备用服务器的装载后期会纳入项目扩展活动。

·对于数据库的使用采用MySQL关系型数据库，因此在程序编写SQL语句时会根据系统开设功能进行代码的实时更新；在开设数据库时明确好数据库的容量，可以设置限定值来确保数据不会溢出，对于仍要加入已满数据库的数据条目会通知管理员进行结构调节与系统维护。

## 3.4 全局数据说明

本部分说明本程序系统中使用的全局数据常量、变量和数据结构。

### 3.4.1 常量

int Max\_Input；最大输入字符数

int Max\_Try； 最大密码错误次数

### 3.4.2 变量

主要用于输入数据

String name;求职者姓名

String aname;招聘名称

String tele;电话号

String education;教育程度

String position;位置

String request;要求

### 3.4.3 数据结构

招聘要求类

class ApplyInfo {

private int aid;

private String aname;

private String tele;

private String education;

private String position;

private String cvurl;

private String status;

public int getAid() {

return aid;

}

}

求职者类

public class Employee {

private int id;

private String aname;

private String position;

private String tele;

public int getId() {

return id;

}

}

应聘信息类

public class RecruitInfo {

private int id;

private String name;

private String position;

private int number;

private String request;

public int getId() {

}

}

## 3.5 体系结构设计评审

本部分以两种方式评估SAD的质量。在第一部分，确认设计是否符合客户指定的所有需求；在第二部分，确认设计是否遵循了良好的设计原则，是否有助于开发任务的进行。

### 3.5.1 Validation

记录人员由吴百川担任，调解人员由尹国泰担任，系统需求分析人员与系统设计人员由孟小凡担任，系统测试人员由黄泰达担任，同时，全员共同担任系统程序设计人员与系统维护人员。同时，邀请了各自的舍友作为局外人参与讨论。

经过讨论，确认了提议的设计中已经包含了必要的接口、输入输出等，并确认了通过典型的执行路径可以正常工作，也能对于故障和不正常输入等做出相应处理。

讨论结果为，该设计符合客户的需要。

### 3.5.2 Verification

经过验证，该体系结构满足以下设计原则：

1.模块化，结构良好，易于理解。

2.部分方面可复用

3.支持易测试性

4.使用了适当的技术处理故障并防止发生失效。

5.可以适应文档里所有预料中的设计改变和扩展。

# 4需求可追踪性

为了确保系统的需求得到满足并进行有效的测试和验证，需要建立需求追踪矩阵。该矩阵将跟踪每个需求，以确保每个需求都得到了满足并得到了测试。需求追踪矩阵将包含以下信息：

·需求标识符

·需求描述

·需求来源

·需求的优先级

·需求的状态（已实现、未实现、已测试、未测试）

·需求的实现状态（实现、未实现、待实现）

·需求的测试状态（已测试、未测试、待测试）

·需求的关联文档

·需求的关联代码

每个需求都将有一个唯一的标识符，可以是数字、字母或其他任何可以用于标识需求的符号。每个需求都应该有一个明确的描述，以确保所有相关人员都理解该需求的含义和目的。

需求来源应该指明该需求是由客户、用户、法规、标准或其他来源提出的。需求的优先级应该指明该需求的重要程度和实现的紧急程度。

需求的状态将指明该需求当前是否已经得到实现和测试。如果该需求已经得到实现和测试，那么状态将被设置为“已实现、已测试”。如果该需求已经得到实现，但尚未进行测试，则状态将被设置为“已实现、未测试”。如果该需求已经得到测试，但尚未得到实现，则状态将被设置为“未实现、已测试”。如果该需求尚未得到实现和测试，则状态将被设置为“未实现、未测试”。

需求的实现状态将指明该需求当前是否已经得到实现。如果该需求已经得到实现，则状态将被设置为“已实现”。如果该需求尚未得到实现，则状态将被设置为“未实现”。如果该需求尚未得到实现，但将在未来得到实现，则状态将被设置为“待实现”。

需求的测试状态将指明该需求当前是否已经得到测试。如果该需求已经得到测试，则状态将被设置为“已测试”。如果该需求尚未得到测试，则状态将被设置为“未测试”。如果该需求尚未得到测试，但将在未来得到测试，则状态将被设置为“待测试”。

需求的关联文档将列出所有与该需求相关的文档。需求的关联代码将列出所有与该需求相关的代码文件和代码行。这将有助于确保所有相关的文档和代码都被正确地更新和维护。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **需求标识符** | **需求描述** | **需求来源** | **需求优先级** | **需求状态** | **需求实现状态** | **需求测试状态** | **需求关联文档** | **需求关联代码** |
| REQ001 | 显示求职者信息 | 用户需求 | 高 | 已实现 | 实现 | 已测试 | 求职者部分 | IApplyService.java, ApplyServiceImpl.java, apply.jsp |
| REQ002 | 添加求职者信息 | 用户需求 | 高 | 已实现 | 实现 | 已测试 | 求职者部分 | IApplyService.java, ApplyServiceImpl.java, apply.jsp, IApplyDao.java, ApplyDaoImpl.java |
| REQ003 | 更新求职者信息 | 用户需求 | 高 | 已实现 | 实现 | 已测试 | 求职者部分 | IApplyService.java, ApplyServiceImpl.java, apply.jsp, IApplyDao.java, ApplyDaoImpl.java |
| REQ004 | 添加企业信息 | 企业需求 | 高 | 已实现 | 实现 | 已测试 | 企业部分 | IRecruitService.java, RecruitServiceImpl.java, insert.jsp, IRecruitDao.java, RecruitDaoImpl.java |
| REQ005 | 显示企业信息 | 企业需求 | 高 | 已实现 | 实现 | 已测试 | 企业部分 | IRecruitService.java, RecruitServiceImpl.java, show.jsp |
| REQ006 | 更新企业信息 | 企业需求 | 高 | 已实现 | 实现 | 已测试 | 企业部分 | IRecruitService.java, RecruitServiceImpl.java, update.jsp, IRecruitDao.java, RecruitDaoImpl.java |
| REQ007 | 添加管理员信息 | 管理员需求 | 高 | 已实现 | 实现 | 已测试 | 管理员部分 | IEmployeeService.java, EmployeeServiceImpl.java, info.jsp, IEmployeeDao.java, EmployeeDaoImpl.java |
| REQ008 | 显示管理员信息 | 管理员需求 | 高 | 已实现 | 实现 | 已测试 | 管理员部分 | IEmployeeService.java, EmployeeServiceImpl.java, appshow.jsp |
| REQ009 | 更新管理员信息 | 管理员需求 | 高 | 已实现 | 实现 | 已测试 | 管理员部分 | IEmployeeService.java, EmployeeServiceImpl.java, info.jsp, IEmployeeDao.java, EmployeeDaoImpl.java |

# 5注解

学习报告

孟小凡：

**阅读《****Software Architecture-a Roadmap》学习报告**

**概述：**

这篇文章围绕着软件体系结构在软件工程中的演进与发展来展开，介绍了软件体系结构在软件项目开发过程中扮演的角色、起到的作用、过去与现在其地位和关键性的对比以及软件工程学科对其未来的预测，肯定了软件体系结构领域在工程上的应用与实践，整体上根据全局展望绘制出Software Architecture的发展路线图。

**关键点：**

·发展历程：业界对于软件体系结构的态度经历了十年之久的缓慢转变，在过去很长一段时间，设计结构被认为是一种相对临时的事情，例如对于依赖非正式盒线图的结构被构建后通常不会对其进行维护，而且在进行学术交流时，建筑师也很少分析结构描述去推断软件中的重要属性。但是人民逐渐意识到结构描述是系统设计的核心，原因是体系结构设计决定着系统成功，于是业界开始认识到一个更具纪律的方法来作为结构设计的标准格外重要，并形成了两种凸显结构重要性的趋势。第一种趋势为对构建复杂软件系统的方法、技术、模式和习惯用法的共同认可，结构组织允许设计者对于复杂系统的描述使用抽象概念从而使整个系统易于理解，同时它们提供重要的语义内容来更清晰得展现系统；第二种趋势为利用特定领域中的共性为产品族提供可重用的框架，这种趋势可以直观感受到结构的使用为系统开发带来极大方便，通过“实例化”共享设计可以做到相对较低成本构建新系统。现阶段，体系结构在软件开发过程中的重要性越发明显，结构师的出现彰显了体系结构地位的上升，可见结构作为系统开发设计的核心，它的发展历程的终点也必然主导整个软件工程行业。

·桥梁角色：软件体系结构通常扮演需求和实现之间桥梁的关键角色。我的理解是，软件开发的开始源于人们的需求，而其结果就是软件的实现，从开始到结果这中间一定是有过程的，但是由于该过程过于漫长且复杂，全部表述出来会十分困难，那么软件体系结构就作为一个大体框架将软件构建设计的全过程直观展示出来。但是软件体系结构不仅仅是以桥梁角色只给出软件开发骨架，文章中指出它至少可以在软件开发的六个方面发挥重要作用，六个方面分别是：理解、重用、构造、演化、分析以及管理。①对于理解方面，软件体系结构在抽象乘次上呈现系统整体，简化了其复杂的构造而使系统的高层设计很容易被理解；②在重用方面，规范化的体系结构设计使得结构描述支持多层次的重用，不同特定领域的软件结构总能找到相应共性，框架的构建与使用也会相对容易；③针对软件的构造，体系结构可以很清晰地指出组件之间的依赖关系，并以层次结构（分层视图）来进行呈现，同时这便于接口的标识与对接；④系统演化也可以通过体系结构进行预期，维护人员也能根据结构明确部分变更对整个系统带来的变化以及造成的后果，从而更加准确地预估修改成本；⑤分析上，体系结构可以无遮蔽地向调研人员展示软件系统的各部分内容，给出的视图与全局数据方便分析过程的进展；⑥在管理层面，体系结构可以作为软件开发过程的一个关键里程碑，管理系统时可以从软件体系结构设计活动进行敏捷更改，带动开发活动更新的进行。

·前景：现阶段软件体系结构并未被确立为一门在整个软件行业中普遍教授和实践的学科，其传播途径有待扩展，技术基础仍需夯实。但其实用价值决定了其未来的前景会非常宽广，文章中也从改变构建与购买的平衡、以网络为中心的计算以及普适计算三个突出趋势对体系结构的影响讲述了结构设计发展受限的原因、扩展延伸的可能性与可行性以及面临的挑战。但总体上软件体系结构的发展趋势会呈现出螺旋式上升态，所以行业对于体系结构领域的前景充满希望。

**主管感受与结论：**

我对于软件体系结构发展的整体感觉是一个先前被忽视低估的核心在被开发者发觉其巨大价值后逐步成为软件开发过程的关键点并引领行业进入新阶段，而体系结构的规范化以及业界对其的深入探索为当前软件工程的低成本高效率开发活动带来了便利条件，同时对产业未来的扩展延伸奠定了坚实基础。

软件开发对体系结构的重视以及将其作为设计的核心关键，明确了整个产业的大方向，不过现阶段软件体系结构领域仍然存在不成熟的地方，存在大量挑战和未知，因此学界对于体系结构的探索与深入仍需进行。

综合来讲，软件体系结构设计作为系统开发过程中极为重要的活动，其发展的成功标志着软工行业进入产业新时期，相关学者与开发人员应当重视这一领域并为其更深层面价值的挖掘投入精力。

-------------------------------------------------------------------------------

吴百川：

**软件体系结构风格：**

软件体系结构风格是描述某一特定应用领域中系统组织方式的惯用模式。体系结构风格定义了一个系统家族，即一个体系结构定义一个词汇表和一组约束。

一、经典软件体系结构风格：

①管道过滤器风格：在管道与过滤器风格的软件体系结构中，每个构件都有一组输入和输出，构件读输入的数据流，经过内部处理，然后产生输出数据流。这个过程通常通过对输入流的变换及增量计算来完成，所以在输入被完全消费之前，输出便产生了。因此，这里的构件称为过滤器。

优点：

1.使得软件构件具有良好的隐蔽性和高内聚、低耦合的特点；

2.允许设计师将整个系统的输入/输出行为看成是多个过滤器的行为的简单合成；

3.支持软件重用；

4.系统维护和增强系统性能简单；

5.允许对一些吞吐量、死锁等属性的分析；

6.支持并行执行。

缺点：

1.通常导致进程成为批处理的结构；

2.不适合交互的应用；

3.因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作，这样就导致了系统性能下降，并增加了编写过滤器的复杂性。

②数据抽象和面向对象系统：抽象数据类型概念对软件系统有着重要作用，目前软件界已普遍转向使用面向对象系统。这种风格建立在数据抽象和面向对象的基础上，数据的表示方法和他们的响应操作封住在一个抽象数据类型或对象中。这种风格的构件是对象。

优点：

1.因为对象对其他对象隐藏它的表示，所以可以改变一个对象的表示，而不影响其他的对象；

2.设计师可以将一些数据存取操作的问题分解成一些交互的代理程序的集合。

缺点：

1.为了使一个对象和另一个对象通过过程调用等进行交互，必须知道对象的标识。

2.必须修改所有显式调用它的其他对象，并消除由此带来的一些副作用。

③基于事件的系统：基于事件的系统风格的思想是构件不直接调用一个过程，而是触发或广播一个或多个事件。系统中的其他构件中的过程在一个或多个事件中注册，当一个事件被触发，系统自动调用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。因此，这种风格也称隐式调用。这种风格的构件是一些模块。

优点：

1.为软件重用提供了强大的支持；

2.为改进系统带来了方便；

缺点：

1.构件放弃了对系统的控制；

2.数据交换的问题；

3.关于正确性推理的问题；

④分层系统：层次系统组织成一个层次结构，每一层为上层服务，并作为下层客户。这种风格的构件是连接件，连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。

优点：

1.支持基于抽象程度递增的系统设计，使设计师可以把一个复杂系统按递增的步骤进行分解；

2.支持功能增强，因为每一层至多和相邻的上下层交互，因此功能的改变最多影响相邻的上下层；

3.支持重用；

缺点：

1.并不是每个系统都可以很容易地划分为分层的模式；

2.很难找到一个合适的、正确的层次抽象方法；

⑤仓库系统及知识库：在仓库风格中，有两种不同的构件：中央数据结构说明当前状态，独立构件在中央数据存储上执行，仓库与外构件间的相互作用在系统中会有很大的变化。

仓库的组成：知识源、黑板数据结构和控制。

⑥C2风格：通过连接件绑定在一起的按照一组规则运作的并行构件网络。

规则：

1.系统中的构件和连接件都有一个顶部和一个底部；

2.构件的顶部应连接到某连接件的底部，构件的底部则应连接到某连接件的顶部，而构件与构件之间的直接连接是不允许的；

3.一个连接件可以和任意数目的其他构件和连接件连接；

4.当两个连接件进行直接连接时，必须由其中一个的底部到另一个的顶部；

优点：

1.系统的构件可实现应用需求，并能将任意复杂度的功能封住在一起；

2.所有构件之间的通信是通过以连接件为中介的异步消息交换机制来实现的；

3.构件相对独立，构件之间依赖性较少；

二、客户/服务器风格

C/S软件体系结构是基于资源不对等，且为实现共享而提出来的，C/S体系结构定义了工作站如何与服务器相连，以实现数据和应用分布到多个处理机上。C/S体系结构有三个主要组成部分：数据库服务器、客户应用程序和网络。

服务器负责有效地管理系统的资源，其任务集中于：

1.数据库安全性的要求；

2.数据库访问并发性的控制；

3.数据库前端的客户应用程序的全局数据完整性规则；

4.数据库的备份与恢复；

客户应用程序的主要任务是：

1.提供用户与数据库交互的界面；

2.向数据库服务器提交用户请求并接收来自数据库服务器的信息；

3.利用客户应用程序对存在与客户端服务器和客户应用程序之间的数据传输；

网络通信软件的主要作用是：完成数据库服务器和客户应用程序之间的数据传输。

C/S体系结构将应用一分为二，服务器(后台)负责数据管理，客户机(前台)完成与用户的交互任务。

C/S体系结构的优点：

1.将客户应用与服务器构件分别运作在不同的计算机上；

2.易于对系统进行扩充和缩小；

3.将大的应用处理任务分布到许多通过网络连接的低成本计算机上，以节约大量费用。

缺点：

1.开发成本高；

2.客户端程序设计复杂；

3.信息内容形式单一；

4.用户界面风格不一；

5.软件移植困难；

6.软件维护和升级困难；

7.新技术不能轻易应用；

三、三层C/S结构风格

三层C/S体系结构是将应用功能分成表示层、功能层和数据层三个部分。与两层C/S结构相比，增加了应用服务器，可以将应用逻辑驻留在应用服务器上，而只有表示层存在与客户机上。这种结构被称为瘦客户机。

表示层：表示层是应用的用户接口部分，它负担着用户与应用间的对话功能。

功能层：功能层相当于应用的主题，它是将具体的业务处理逻辑编入程序中。

数据层：数据层就是数据库管理系统，负责管理对数据库数据的读写。

在三层C/S体系构件中，中间件是最重要的构件。

三层C/S结构的优点：

1.允许合理地划分三层结构的功能，使之在逻辑上保持相对独立性，从而使整个系统的逻辑结构更为清晰，能提高系统和软件的可维护性和可扩展性；

2.允许更灵活有效地选用相应的平台和各个组成部分可以具有良好的可升级性和开发性。

3.三层C/S结构中，应用的各层可以并行开发，各层也可以选择各自最适合的开发语言。

4.允许充分利用功能层有效地隔开表示层与数据层，未授权的用户难以绕过功能层而利用数据库工具或黑客手段去非法地访问数据层，这就为严格的安全管理奠定了坚实的基础；整个系统的管理层次也更加合理和可控制。

3.4浏览/服务器风格

基于B/S体系结构的软件，系统安装、修改和维护全部在服务器端解决。用户在使用系统时，仅需要一个浏览器就可运行全部的模块，真正达到了“零客户端”的功能，很容易在运行时自动升级。

与C/S体系结构相比，B/S体系结构也有许多不足之处：

1.B/S体系结构缺乏相对动态页面的支持能力，没有集成有效的数据库处理功能；

2.B/S体系结构的系统扩展能力差，安全性难以控制；

3.采用B/S体系结构的应用系统，在数据查询等响应速度上，要远远低于C/S体系结构。

4.B/S体系结构的数据提交一般以页面为单位，数据的动态交互性不强，不利于在线事务处理应用。

四、异构结构风格

异构结构是C/S和B/S体系结构混合而成。

内外有别模型：企业内部用户通过局域网直接访问数据库采用C/S体系结构，外部用户访问数据库采用B/S体系结构。

查改有别模型：需要执行维护和修改数据操作的，使用C/S体系结构；执行一般查询和浏览操作的采用B/S体系结构。

视图模型：

软件架构涉及到抽象、分解和组合、风格和美学。RUP4+1架构方法采用用例驱动，在软件[生命周期](https://so.csdn.net/so/search?q=%E7%94%9F%E5%91%BD%E5%91%A8%E6%9C%9F&spm=1001.2101.3001.7020" \t "_blank)的各个阶段对软件进行建模，从不同视角对系统进行解读，从而形成统一软件过程架构描述。

1.用例视图（Use Cases View），最初称为场景视图，关注最终用户需求，为整个技术架构的上线文环境.通常用UML用例图和活动图描述。

2.逻辑视图（Logical view），主要是整个系统的抽象结构表述，关注系统提供最终用户的功能，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

3.处理视图（Process view）处理视图关注系统动态运行时，主要是进程以及相关的并发、同步、通信等问题。处理视图和开发视图的关系：开发视图一般偏重程序包在编译时期的静态依赖关系，而这些程序运行起来之后会表现为对象、线程、进程，处理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。

4.物理视图（Physical view ）物理视图通常也叫做部署视图(deploymentview)，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。物理视图和处理视图的关系：处理视图特别关注目标程序的动态执行情况，而物理视图重视目标程序的静态位置问题；物理视图是综合考虑软件系统和整个IT系统相互影响的架构视图。

5.开发视图(Development View)，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，也叫做实现视图(implementation view)。开发视图关注程序包，不仅包括要编写的源程序，还包括可以直接使用的第三方SDK和现成框架、类库，以及开发的系统将运行于其上的系统软件或中间件, 在UML中用组件图，包图来表述。开发视图和逻辑视图之间可能存在一定的映射关系：比如逻辑层一般会映射到多个程序包等。

-------------------------------------------------------------------------------

尹国泰：

**《Software Architecture in Practice》学习报告**

本学习报告旨在探讨《Software Architecture in Practice》这本书的内容，该书由Len Bass、Paul Clements和Rick Kazman合著。我通过学习这本经典之作，加深了对软件体系结构的理解，学习了软件体系结构的基本概念、方法和实践，并将其应用于实际软件开发项目中。

《Software Architecture in Practice》是软件体系结构领域的重要参考书籍之一。本书内容涵盖了软件体系结构的基本概念、架构设计方法、架构评估和演化等方面。它提供了全面而系统的知识体系，帮助读者理解和应用软件体系结构的原则和最佳实践。

本书首先介绍了软件体系结构的基本概念，如组件、连接方式、视图和质量属性等。我了解到软件体系结构不仅关注系统的结构和组成部分，还关注系统与其环境之间的交互和依赖关系。此外，书中还强调了软件体系结构的重要性，它对软件系统的可维护性、可扩展性和可重用性起着关键作用。

《Software Architecture in Practice》详细介绍了一系列的架构设计方法，帮助读者在实践中应用软件体系结构。其中包括需求分析、架构设计原则、模式选择和架构评估等内容。我学习了如何通过需求分析来确定系统的功能和非功能需求，以及如何基于这些需求制定适当的架构设计决策。此外，书中还介绍了常用的架构设计模式和架构风格，帮助我选择和应用适当的模式来解决特定的设计问题。

书中强调了架构评估的重要性，以确保所设计的软件体系结构满足预期的质量属性和需求。我学习了不同的评估方法和工具，如质量属性工作表、场景分析和模拟等。这些方法可以帮助我识别潜在的问题和风险，并提供相应的改进措施。此外，书中还介绍了架构演化的概念和方法，帮助我在系统发展过程中对软件体系结构进行适应和演进。

《Software Architecture in Practice》提供了丰富的实际案例和应用场景，帮助读者将所学的软件体系结构知识应用于实际项目中。通过案例分析，我可以了解不同领域和规模的软件系统是如何应用软件体系结构原则和方法的。这些案例展示了成功的实践经验和挑战，让我能够从中汲取经验教训，并在实际项目中更好地设计和实施软件体系结构。

通过学习《Software Architecture in Practice》，我获得了以下几方面的收获：

1. 深入理解软件体系结构的基本概念和原则，包括组件、连接方式、视图和质量属性等。

2. 学习了架构设计方法和最佳实践，掌握了需求分析、架构设计原则、模式选择和架构评估等关键技术。

3. 了解了架构评估的方法和工具，能够识别和解决潜在的问题和风险。

4. 掌握了软件体系结构的演化方法，能够在系统发展过程中进行适应和演进。

5. 通过实际案例和应用场景的学习，加深了对软件体系结构实践的理解和应用能力。

在学习《Software Architecture in Practice》的过程中，我也认识到了一些潜在的改进点和进一步的学习方向：

1.深入学习架构设计模式：《Software Architecture in Practice》简要介绍了一些常见的架构设计模式，但对于每个模式的详细解释和实际应用场景的讨论相对较少。我可以进一步研究各种架构设计模式，理解它们的原理、适用场景和优缺点，以便更好地运用到实际项目中。

2.拓宽领域知识：软件体系结构的实践涵盖了不同领域和行业，而《Software Architecture in Practice》的案例主要集中在一些典型的应用场景上。我可以进一步研究其他领域的实际案例，了解不同行业的软件体系结构挑战和解决方案，从中获得更全面的实践经验。

3.学习新兴技术和趋势：软件体系结构领域一直在不断发展和演进，新的技术和趋势不断涌现。我可以关注当前的热门话题，如微服务架构、容器化和云原生架构等，并学习相关的文献和资源，以跟上行业的最新发展。

4.参与实践项目和团队：除了学习理论知识，我还可以通过参与实际软件开发项目和加入相关的团队，亲身经历和实践软件体系结构的设计和实施过程。这样可以更好地理解实际项目中的挑战和决策，提高我的实践能力和经验。

通过持续学习和实践，我可以不断提升在软件体系结构领域的专业知识和技能，并为构建高质量的软件系统做出积极贡献。《Software Architecture in Practice》为我打下了坚实的基础，而我的学习之路才刚刚开始。

-------------------------------------------------------------------------------

黄泰达：

1．引言

1.1 背景介绍

随着互联网和软件技术的飞速发展，软件工程的安全性和架构设计成为了当今软件行业中最重要的议题之一。在这个数字化时代，每天都要使用各种软件，从个人电脑到智能手机再到云端服务，软件已经成为人们日常生活和商业活动中不可或缺的一部分。但是，随着软件的广泛应用和使用，与之相关的安全问题和架构缺陷也变得越来越突出，如数据泄露、恶意攻击和系统崩溃等。因此，对软件工程安全和架构的关注已成为保障软件质量和用户安全的重要措施。

1.2研究问题的目的和意义

本学习报告的目的是以课程所学为基础，学习会议INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE 的安全与架构方面文章，并查找相关资料，来学习探讨软件工程中的安全和架构方向。

本文包括安全基础知识，包括威胁建模、安全需求、安全测试和安全审计等内容。同时，也会探讨软件架构设计的基础知识，包括架构设计原则、架构风格、架构模式和架构决策等内容。最后在此基础上，将分析安全和架构之间的关系，探讨如何将安全纳入架构决策。

2. 软件工程安全

在软件开发过程中，安全是一个至关重要的方面。任何系统都可能面临各种各样的威胁，包括数据泄露、未经授权的访问、恶意软件等等。软件工程安全包括威胁建模、安全需求、安全测试和安全审计等一系列措施，用于确保软件系统的安全性。

2.1 安全基础知识

首先要熟知软件工程中安全基础特性，包括：

机密性（Confidentiality）：确保系统中的数据只能被授权的用户或程序访问。

完整性（Integrity）：确保系统中的数据在传输和存储过程中不会被篡改或丢失。

可用性（Availability）：确保系统始终可用，即使面临攻击或其他威胁也能继续提供服务。

不可抵赖性（Non-repudiation）：确保用户无法否认他们已经执行过的操作。

2.2. 威胁建模

威胁建模是一种识别和分析系统威胁的方法。通过威胁建模，可以预测潜在的攻击和漏洞，并为系统设计和开发阶段提供指导。常用的威胁建模技术包括：

数据流图（Data flow diagrams，DFD）：DFD可帮助开发人员识别系统中的数据流，以及与数据流相关的威胁和漏洞。

威胁建模语言（Threat modeling language，TML）：TML是一种用于描述威胁建模的语言，可用于描述系统中的组件、数据流和威胁等。

风险评估（Risk assessment）：风险评估用于评估潜在的威胁和漏洞，并确定其对系统的影响。

2.3 安全需求

安全需求是在软件开发过程中为确保系统安全而制定的需求。这些需求应该在系统开发的早期就制定好，并在开发过程中得到跟踪和验证。常见的安全需求包括：

访问控制：确保只有授权用户能够访问系统的敏感数据和功能。

安全通信：确保在系统内部和与外部系统之间传输的数据是加密的和完整的。

安全审计：确保系统可以记录和监视对系统的访问，并能够检测到潜在的攻击和威胁。

2.4安全测试

安全测试是一种用于检测软件系统安全漏洞的测试方法。安全测试可帮助开发人员识别和修复系统中的漏洞和弱点，以确保系统的安全性。常见的安全测试方法包括：

静态分析（Static analysis）：静态分析通过检查源代码或二进制文件中的安全漏洞，以帮助开发人员修复漏洞。

动态测试（Dynamic testing）：动态测试通过模拟真实攻击的方式来测试系统的安全性。

漏洞扫描（Vulnerability scanning）：漏洞扫描器用于自动扫描系统中的漏洞和弱点，并提供修复建议。

2.5安全审计

安全审计是一种用于检查系统中的安全漏洞和弱点的方法。安全审计可用于检查系统的安全性、遵守性和性能，并为系统提供改进建议。常见的安全审计方法包括：

审计日志分析（Log analysis）：审计日志可用于记录系统中的所有活动，并为安全审计提供数据和证据。

漏洞评估（Vulnerability assessment）：漏洞评估用于检查系统中的漏洞和弱点，并提供修复建议。

渗透测试（Penetration testing）：渗透测试模拟真实的攻击方式来测试系统的安全性，并提供修复建议。

在软件工程中，安全是一个至关重要的方面。通过威胁建模、安全需求、安全测试和安全审计等一系列措施，可以确保软件系统的安全性。在开发软件系统时，应该在早期就制定好安全需求，并在开发过程中使用合适的安全测试和审计方法来检测和修复系统中的漏洞和弱点。这样才能确保软件系统在面对各种威胁时具有足够的安全性。

3.软件架构部分

软件架构是指软件系统的高级结构，描述了系统的组成部分、它们之间的交互关系和约束条件。在软件工程中，软件架构是一个关键的方面，它不仅决定了系统的质量和可维护性，还影响了系统的安全性和可靠性。

3.1 软件架构设计

软件架构设计是指在开发软件系统之前，设计软件架构的过程。在软件架构设计过程中，开发人员需要考虑系统的需求、目标、功能、性能、安全性、可维护性等方面，并选择合适的架构模式和技术来满足这些需求。

常见的软件架构模式包括：

客户端-服务器模式（Client-server）：客户端-服务器模式将系统分为客户端和服务器两个部分，客户端向服务器发送请求并接收响应。

分布式系统（Distributed system）：分布式系统将系统分为多个节点，这些节点可以分布在不同的地理位置上，通过网络进行通信和协作。

微服务架构（Microservices）：微服务架构将系统分为多个小型服务，每个服务专注于一个特定的业务功能，并通过API进行通信和协作。

3.2. 软件架构模式

软件架构模式是一种通用的架构设计，它可以帮助开发人员满足软件系统的需求和目标。常见的软件架构模式包括：

分层架构（Layered architecture）：分层架构将系统分为多个层，每个层都有特定的功能和责任。

MVC架构（Model-View-Controller）：MVC架构将系统分为模型、视图和控制器三个部分，模型处理数据、视图呈现数据，控制器负责协调模型和视图之间的交互。

事件驱动架构（Event-driven architecture）：事件驱动架构将系统分为多个组件，这些组件通过事件进行通信和协作。

3.3 软件架构模式的选择

在选择软件架构模式时，开发人员需要考虑系统的需求和目标，选择合适的架构模式和技术。以下是一些选择软件架构模式时应该考虑的因素：

系统的需求和目标

系统的规模和复杂度

系统的性能和可扩展性

系统的安全性和可靠性

开发团队的经验和技能

3.5软件架构评估

软件架构评估是指对软件架构进行分析和评估，以确定其是否满足系统的需求和目标。软件架构评估可以通过多种方式进行，包括：

代码审查：通过对软件代码进行审查，评估软件架构是否符合最佳实践和设计原则。

建模：通过建立软件架构模型，对软件架构进行分析和评估。

静态分析：通过静态分析工具，对软件代码进行分析和评估。

动态分析：通过运行软件系统，并对其进行监视和分析，评估软件架构的性能和可靠性。

3.6软件架构重构

软件架构重构是指在不改变软件系统功能的前提下，对软件架构进行重构和优化，以满足系统的需求和目标。软件架构重构可以通过以下方式实现：

模块化：将系统分解为模块，每个模块专注于一个特定的业务功能。

重构模式：通过使用常见的重构模式，对软件架构进行重构和优化。

面向服务架构（SOA）：将系统分解为多个服务，每个服务专注于一个特定的业务功能。

云架构：将系统部署在云环境中，以提高系统的可扩展性和可靠性。

总之，软件架构是软件工程中非常重要的一个方面。在设计软件架构时，开发人员需要考虑系统的需求、目标、性能、安全性和可维护性等方面，并选择合适的架构模式和技术来实现。同时，软件架构评估和重构也是确保软件系统质量和可维护性的重要手段。

4.安全和架构之间的关系

安全和架构是软件工程中密切相关的两个方面。在软件系统设计和开发过程中，安全和架构之间存在着相互依赖的关系。

4.1. 安全对架构的影响

安全需求对架构的影响：安全需求是指系统在设计和开发过程中需要满足的安全性要求。这些需求可以对软件架构的设计和选择产生影响。例如，在设计系统架构时，需要考虑安全策略和防御机制的集成，以确保系统的安全性。

安全威胁对架构的影响：安全威胁是指可能危及系统安全性的因素。在设计系统架构时，需要考虑安全威胁对系统的影响，并设计相应的防御机制和应急措施来应对。

安全测试对架构的影响：安全测试是评估软件系统安全性的过程。通过安全测试，可以评估系统的安全性并发现可能存在的漏洞和安全问题，从而指导软件架构的优化和改进。

4.2 架构对安全的影响

架构对安全性的保证：软件架构可以对系统的安全性提供保障。例如，通过使用分层架构，可以将安全机制分布在不同层次上，并确保系统的安全性。

架构对安全的防御：软件架构可以通过设计防御机制来保护系统免受攻击。例如，在设计系统架构时，可以使用安全模式和安全策略来确保系统的安全性。

架构对安全问题的解决：软件架构可以通过设计安全措施来解决安全问题。例如，在设计系统架构时，可以选择使用安全技术和加密技术来解决数据安全问题。

总之，软件架构和安全是软件工程中非常重要的两个方面。在软件系统设计和开发过程中，需要考虑这两个方面的相互关系，并选择合适的架构模式和安全措施来确保系统的安全性和可维护性。