软件(体系结构)设计说明(SAD)

组员：吕策、徐伟、李奕辰、方羿阳、杨伟钦

目录

1 引言

1.1 标识

项目名称：国际物流管理系统

版本号：2023.1.1

发行号：2023-5-30

缩略词：ILMS (International logistics management system)

标识号：A0002

1.2 系统概述

本文档适用于开发一款名为“物流管理系统”的软件，用于联合物流有限公司管理物流业务。该软件包括票据管理、接货管理、配车管理、到货管理、中转管理、结算管理、客户服务、监控分析、成本核算、应用管理和系统管理等 11 个模块，以满足用户在物流管理方面的需求。

物流管理系统是一款基于 Web 的应用程序，可以在标准的 Web 浏览器上运行，支持多平台使用。它的主要特性包括界面友好、操作简便、安全性高、功能强大、数据可靠等。

该系统的开发历史可以追溯到 2023 年 3 月，当时联合物流有限公司提出开发物流管理系统的需求，并由山东大学（青岛）计算机科学与技术学院“起个队名这么难”软件工程开发设计团队进行了初步的调研和方案设计。在历经多轮的讨论和开发后，该系统将于 2023 年 5 月底正式发布，目前由联合物流有限公司的员工和管理人员共同使用测试版本。

除需方和开发方外，支持机构为山东大学（青岛）计算机科学与技术学院和联合物流有限公司，提供技术支持和维护服务。

目前，该系统的运行现场为联合物流有限公司的总部和分支机构。未来，随着系统的不断优化和升级，计划将在更多的物流企业中推广使用。

除本文档外，还有以下相关文档:

- 《物流管理系统用户手册》
- 《物流管理系统安装指南》
- 《物流管理系统技术规格说明书》

物流管理系统是为了提高物流配送效率而开发的一套软件系统。它主要用于管理货物的票据、接货、配车、到货、中转、结算、客户服务、监控分析、成本核算、应用管理和系统管理等方面。该系统由自己小队开发，面向物流公司的需方和用户，由开发方负责开发和维护，支持机构提供技术支持。当前运行现场为 SDU，未来计划扩展到全国范围内。

1. 3 文档概述

本文档是物流管理系统的软件需求规格说明书，旨在描述该系统的需求和功能，以便开发方开发出符合用户需求的软件系统。本文档适用于联合物流有限公司全国范围内的物流管理系统开发项目。本文档标识号为 A0002，标题为《物流管理系统软件需求规格说明书》，版本号为 1.1，拟发行号为 2023-05-30。

本文档的内容涵盖了该系统的各个方面，包括票据管理、接货管理、配车管理、到货管理、中转管理、结算管理、客户服务、监控分析、成本核算、应用管理、系统管理等 11 个部分。除了描述系统的功能需求外，本文档还包括对性能、安全、可靠性、易用性和可维护性等方面的需求描述。同时，本文档还包括了系统的用户界面、数据流图、数据字典、用例图、活动图、状态图等详细说明，以便开发方清晰地了解系统的各个方面。

本文档内容仅供内部使用，具有保密性。未经需方授权，开发方和其他未授权人员不得将本文档内容披露给第三方。

1.4 基线

本系统设计说明书的编写基线为《物流管理系统软件需求分析报告》，版本号为 1.0，发布日期为 2023-03-21。

设计基线是本系统设计说明书的依据，它包括了本系统的需求规格说明书、概要设计和详细设计等文档。在编写本系统设计说明书时，我们参考了以下的设计基线：

1.《物流管理系统软件需求规格说明书》

该文档详细描述了物流管理系统的功能需求、非功能需求和性能需求等，是本系统设计的基础。在本系统设计中，我们对该文档中描述的功能和性能需求进行了细化和具体化，并进行了详细的设计。

2.《物流管理系统概要设计》

该文档主要描述了物流管理系统的系统架构、模块划分、接口设计等内容。在本系统设计中，我们对该文档中描述的系统架构和模块划分进行了优化和修改，并进行了详细的模块设计和接口设计。

3.《物流管理系统详细设计》

该文档详细描述了物流管理系统各个模块的具体设计和实现。在本系统设计中，我们对该文档中的模块设计进行了参考和修改，同时进行了代码实现和单元测试。

以上三个文档是本系统设计说明书所依据的设计基线，它们为本系统设计提供了全面的参考和依据，保证了系统设计的完整性和一致性。

2 引用文件

以下是适用于本 SAD 的引用文件的列表:

1. GB/T 985.1-2008，信息技术 项目进展报告 第 1 部分：总则。本标准规定了项目进展报告的概念、应用范围、目的和原则，以及编制、审查、批准和发放的要求。

2. GB/T 985.2-2008，信息技术 项目进展报告 第 2 部分：软件。本标准规定了软件项目进展报告的编制要求，包括软件项目的定义、项目计划、需求分析、设计、编码和测试等阶段的内容和报告格式，以及报告的审查和批准程序。

3. GB/T 19388-2014，信息技术 软件需求规格说明。本标准规定了软件需求规格说明书的编制要求，包括需求规格说明书的定义、结构、内容和格式，以及需求的表示方法、管理、审查和确认的流程。

4. A0002 物流管理系统软件需求规格说明书，版本号 1.1，发行日期 2023-05-30。本文档是本系统的 SRS，旨在描述系统的需求和功能，以便开发方开发出符合用户需求的软件系统。

5. A0003 物流管理系统软件概要设计说明书，版本号 1.1，发行日期 2023-05-31。本文档是本系统的 SDS，旨在描述系统的总体设计和架构，包括系统的组成、模块划分、数据流和控制流的设计等。

需要注意的是，本文档所引用的文件中，除了公开发行的标准文献外，其他文献的来源和获取方式应在文档中进行明确说明。此外，本文档还应遵守所有引用文件中规定的要求和标准。

3 CSCI 级设计决策

3.1 物流计划管理

a.关于 CSCI 应接受的输入和产生的输出的设计决策

- 输入：订单信息、交通状况、路况信息;
- 输出：物流规划生成的货物位置信息、仓储管理所涉及的货物质量、数量、属性

等库存信息；

- 接口：与运输跟踪管理和仓储管理 CSCI 接口进行传递。

b.有关响应每个输入或条件的 CSCI 行为的设计决策

- 根据订单信息、交通状况、路况信息进行物流规划；
- 响应时间需要在数秒之内完成，性能需求主要集中在响应速度和准确性上。
- 对于不允许的输入或条件，应给出错误提示并要求用户重新输入。

c.有关数据库/数据文件如何呈现给用户的设计决策

- 为包含物流计划的多张表建立关系，以便将各物流计划产品、服务请求、货车等信息高效地关联起来；
- 提供基于不同角色和权限的可视化查询界面和报表，以满足客户对物流计划的实时监测需求。

d.为满足安全性、保密性、私密性需求而选择的方法

- 构建多层次数据库系统架构，将不同用户数据存储在相应的逻辑数据库表中，分别授权匹配的权限；

  引入加密技术对重要数据进行加密处理，使用单独的 Key Management System 进行密钥管理；
- 提供访问/操作日志和审计功能，记录和查询所有非正常行为。

e.对应需求所做的其他 CSCI 级设计决策

- 采用数据库集群横向扩展技术，增加服务器数量以应对高并发访问；
- 设计多层次的数据备份策略，定期备份数据，并且保留多个时间点的备份；
- 针对故障恢复方案设计自动化恢复机制，尽可能避免因为人工干预造成的额外风险。

3. 2 运输跟踪管理

a.关于 CSCI 应接受的输入和产生的输出的设计决策

- 输入：包括订单信息、司机和运输工具的可用性和交通状况等；
- 输出：货物实时位置和状态的查询信息；
- 接口：与物流计划管理 CSCI 接口进行传递，允许用户调整货物状态、标记货物损坏情况等操作，并提供显示实时货物位置信息的地图界面和货物状态查询界面的用户接口。

b.有关响应每个输入或条件的 CSCI 行为的设计决策

- 在接收到订单信息后，进行运输路径规划，并持续更新货物实时位置和状态信息以保证物流可视化的效果；
- 响应时间需要在秒级别，性能需求主要集中在响应速度和准确性上；
- 如果输入或条件不满足要求，将对货物状态进行标记并发出警告。

c.有关数据库/数据文件如何呈现给用户的设计决策

- 设计用于 GPS 定位和轨迹跟踪信息的多层数据表架构，精细记录包裹的位置、状态和运行轨迹；
- 提供可视化地图及实时反馈机制，使用户能够及时了解当前订单详细信息以及装卸过程的流畅度。

d.为满足安全性、保密性、私密性需求而选择的方法

- 强化身份验证和访问控制机制以确保只有经过授权的用户才能够对数据进行查看和修改；
- 对涉及个人隐私的信息加上敏感字眼标记，在查看和输出时严格控制访问和可见范围；
- 减小关键词索引，根据业务内容设计智能提取模型进行保密性处理，并增加敏感度阈值。

e.对应需求所做的其他 CSCI 级设计决策

- 设计分布式微服务架构，并对模块进行拆分，实现服务间的松耦合，提高系统可用性和易维护性；
- 使用容器化技术将应用程序与其依赖项打包成一个独立的容器，可以更高效地进行部署、升级和维护操作；
- 引入消息队列中间件提高服务间协作的可靠性和低延迟过程的发送与 如有错误，则停止执行.

3.3 仓储管理

a.关于 CSCI 应接受的输入和产生的输出的设计决策

- 输入：货物入库、出库记录及盘存管理等；
- 输出：货物信息如货物量、属性、序列号等信息给物流计划管理以供运输规划过程参考；

- 接口：与物流计划管理和运输跟踪管理 CSCI 接口进行传递，并提供展示仓库库存的实时状态和库存变化记录相关界面的用户接口。

b.有关响应每个输入或条件的 CSCI 行为的设计决策

- 在接收到货物入库、出库等记录后，实时更新库存信息；
- 响应时间需要在数秒之内，性能需求主要集中在响应速度和准确性上；
- 对于不允许的输入或条件，应给出错误提示并要求用户重新输入。

c.有关数据库/数据文件如何呈现给用户的设计决策

- 实现库存管理等一系列具体业务模块，采用多表联立查询预留资源并定位存货管理等一系列基础信息；
- 配置合适的授权策略，在保障数据访问安全性的同时方便仓库管理员合理拥有数据查看权限。

d.为满足安全性、保密性、私密性需求而选择的方法

- 使用双因素身份认证技术的方式，强化密码策略和口令复杂度规则；
- 针对封闭的数据区域和隔离环境规定黑白名单和访问状态，使得陌生者无法擅自进入；
- 采用访问系统时强制此人处于后台认证并记录日志的方式，按需使用或禁用权限来供识别用户身份。

e.对应需求所做的其他 CSCI 级设计决策

- 管理数据一致性的检查和不一致性的消除等功能，确保数据完整性和可用性；
- 设计鲁棒的错误处理机制，如错误日志记录、故障转移和恢复，保证仓库管理系统的稳定性；
- 加强网络安全，采用标准协议和公共密钥进行计算机之间的身份验证与数据传输加密，以提高安全性。

3. 4 运费结算管理

a.关于 CSCI 应接受的输入和产生的输出的设计决策

- 输入：运输订单和货运收入支出明细等；
- 输出：计算和核算后的费用，可输出最终的运费结算信息给财务管理等其他系统；
- 接口：与订单处理 CSCI 进行接口设计，以传递订单信息，并提供展示对账单、发票等相关信息的用户界面。

b.有关响应每个输入或条件的 CSCI 行为的设计决策

- 在接收到运输订单和货运收入支出明细后，通过相应的算法计算运费，再输出最终的运费结算信息；
- 响应时间需要在数秒之内，性能需求主要集中在响应速度和准确性上；
- 对于不允许的输入或条件，应给出错误提示并要求用户重新输入。

c.有关数据库/数据文件如何呈现给用户的设计决策

- 为资费核算建立一套单独的特定数据表格，包涵所有公司合理的收费项目；
- 实现不同类型货车的费用计算和不同阶段间的结算，将交易记录与发票关联起来实现账务管理功能；
- 支持多方面的人工审核机制以及异常处理等操作。

d.为满足安全性、保密性、私密性需求而选择的方法

- 使用强大的安全服务并为其提供大量紧急 SaaS 组合进行有效地加密、分隔和存储业务数据；
- 对数据的梳理和保护方案设定需要考虑保有数据规中的安全要求，对外系统授权设置特定的生命周期规划；
- 配置内容审计技术，实时检测运营设备的操作记录及其状态变化，以追踪检测业内是否有限定开放风险。

e.对应需求所做的其他 CSCI 级设计决策

- 引入流程引擎技术来解耦业务逻辑和系统实现，降低耦合度，增强可维护性；
- 为分布式系统或多线程应用程序设计容错处理措施，确保系统在访问失败时的友好处理；
- 实现邮件通知功能，当发现交易问题或者积压较严重时可以及时提醒相关人员进行操作。

3.5 下单功能

a.关于 CSCI 应接受的输入和产生的输出的设计决策

- 输入：订单需求、货物属性、运输距离、价格等因素产生可行运输方案；
- 输出：不同的配送方案供用户选择最佳选项；
- 接口：与物流计划管理、运输跟踪管理、订单处理等 CSCI 进行接口设计，向用户展示选择配送方案、输入地址、选择支付方式等相关界面的用户接口。

b.有关响应每个输入或条件的 CSCI 行为的设计决策

- 根据用户输入的订单需求、货物属性、运输距离、价格等进行可行运输方案筛选，并输出不同配送方案供用户选择；
- 响应时间需要在数秒之内，性能需求主要集中在响应速度和准确性上；
- 对于不允许的输入或条件，应给出错误提示并要求用户重新输入。

c.有关数据库/数据文件如何呈现给用户的设计决策

- 设计精心定制的下单者反馈系统，形成新订单报表中集成各个有价值型能力说明，为用户提供订单操作时所需尽量多层且详细的信息披露；
- 使得客户可以在界面上透明地跟踪订单状态，并对物流团队进行评分和回馈。

d.为满足安全性、保密性、私密性需求而选择的方法

- 通过传输层安全协议 TLS 等方式确保 Web 安全通信；
- 建立专用中央身份验证机构以支持身份单点登录，减少重复登录过程及防范无效访问的攻击；
- 加强用户端本地加密和签名技术，使密码不被暴力破解。

e.对应需求所做的其他 CSCI 级设计决策

- 添加灵活的 API，以便其他应用程序调用；
- 设计基于云端互联体系下的可拓展技术架构，实现场景资源动态分配，并根据访问频率判断是否需要进行相应的扩容作业；
- 引入国际标准的 RESTful API 风格，使客户能够轻松访问各个服务端点，并实现状态转换。

当涉及到大规模的软件系统设计时，确保性能和灵活性的同时，也需要注重以下方面，以便为所需的 CSCI 级别需求提供合适的解决方法：

1. 高可用性

- 通过使用负载均衡、故障转移、容错处理以及其他高可用技术来实现系统的高可用性，防止系统单点故障引起的服务中断；
- 设计具备自动恢复机制的监控系统，以迅速发现异常行为并采取适当行动。

2. 可拓展性

- 采用水平扩展的方式实现系统的可拓展性，进行横向扩展，根据业务需求随着数据量、访问率等需求变化动态增加服务器数量；
- 构建弹性架构，在支持快速上下线的前提下，实现更灵活和可定制的服务设计。

3. 安全性

- 采用多层安全策略保护数据安全，包括基于角色访问控制和文档加密；
- 采用常见的安全审计标准，记录所有系统操作和事件，并允许对登录会话和重要操作执行更严格的身份验证流程。

4. 数据集成

- 使用标准化 API，以确保系统与移动设备、物联网设备或其他应用程序的数据互通性；
- 通过引入中间件和集成服务平台，实现不同数据源的无缝链接，并确保数据在传输过程中保持安全和完整。

5. 网络解决方案

- 利用云计算等技术优势，在实现灵活性和可拓展性的前提下，建立一种支持每秒处理数千次请求的网络解决方案；
- 设计具有自我扩展功能的网络体系结构，以便在负载高峰时添加服务器并缩减服务器数可以有效地降低系统的压力。

4 CSCI 体系结构设计

4.1 体系结构

4.1.1 程序(模块)划分

用一系列图表列出本 CSCI 内的每个程序(包括每个模块和子程序)的名称、标识符、功能及其所包含的源标准名。

模块名称	标识符	功能	包含的源标准名
单据处理	com.ansel.controller.BillController	处理单据相关的请求何业务逻辑: 1. 添加单据分发信息 2. 添加货物到货回执信息 3. 分页查询单据信息 4. 查询未分发的运单信息	• Spring Framework: Autowired, ControllerAdvice, CrossOrigin, RequestMapping, RequestMeth

			od • Spring Data JPA: Page, PageRequest, Pageable • Swagger: Api • com.ansel.bean: BillInfoBillRelease、GoodsReceiptInfo
回告处理	com.ansel.controller.CallbackController	处理回告相关的业务请求: 1. 添加回告信息 2. 查询回告信息	• Spring Framework: Autowired, ControllerAdvice, CrossOrigin, RequestMapping, RequestMethod • Spring Data JPA: Page, PageRequest, Pageable • Swagger: Apicom.ansel.bean: CallbackInfo、ICallbackService
货运单回执	com.anse	处理与货运单回执相关的请求和	• Spring

处理	l.controller.CargoReceiptController	业务逻辑: 1. 填写货运回执单主表 2. 查询货运回执单编号 3. 通过货运回执单查询客户信息, 4. 查询所有运单, 5. 查询运单状态 6. 通过 id 查询单个货运单 7. 修改货运回执单 8. 提交货运回执单 9. 删除货运回执单	Framework: Autowired, ControllerAdvice, CrossOrigin, RequestMapping, RequestMethod • Spring Data JPA: Page, PageRequest, Pageable • Swagger: Api • com.ansel.bean: CargoReceipt、CargoReceiptDetail、GoodsBill、ICargoReceiptService、Page、PageRequest、Pageable、Result
财务处理	com.ansel.controller.CheckController	处理与营业外收入、财务费用、管理费用和员工工资相关的请求和业务逻辑: 1. 录入营业外收入 2. 查询所有营业外收入 3. 录入财务费用 4. 根据 id 查询管理费用 5. 录入员工工资 6. 根据员工编号查询员工工资	• Spring Framework: Autowired, ControllerAdvice, CrossOrigin, RequestMapping, RequestMethod • Spring Data

		7. 录入管理费用 8. 查询所有管理费用 9. 查询当前月报等	JPA: Page, PageRequest, Pageable • Swagger: Api • com.ansel.bean: ExtraIncome、FinanceFee、ManageFee、EmployeeWage、ICheckService、Page、PageRequest、PageableIncomeMonthlyTemp • com.ansel.service: IClearService • com.ansel.util: Result
结算处理	com.ansel.controller.ClearController	处理与结算相关的业务逻辑: 1. 查询未结算的司机结算实体及其详细信息; 2. 根据物流订单编号查询已填写的司机结算实体及其详细信息; 3. 向系统中添加新的司机结算实体; 4. 查询未结算的客户结算实体及其详细信息; 5. 根据物流订单编号查询已填写	• Java 基础库: List • Spring Framework: Autowired, ControllerAdvice, CrossOrigin, RequestMapping, RequestMethod • Spring Data

		的客户结算实体及其详细信息； 6. 向系统中添加新的客户结算实体； 7. 查询未结算的代收结算实体及其详细信息； 8. 根据物流订单编号查询已填写的代收结算实体及其详细信息； 9. 向系统中添加新的代收结算实体； 10. 添加新的杂费结算实体； 11. 分页查询所有的杂费结算实体。	JPA： Page, PageRequest, Pageable • Swagger：Api • com.ansel.bean： CustomerBillClear, DriverClear, ExtraClear, ProxyFeeClear • com.ansel.service： IClearService • com.ansel.util：Result
客户信息管理	com.ansel.controller.CustomerController	实现客户信息管理相关业务逻辑： 1. 添加顾客信息 2. 删除顾客信息 3. 查询顾客信息 4. 更新顾客信息	• Spring Data JPA • io.swagger.annotations • org.springframework.web.bind.annotation • com.ansel.bean.CustomerInfo • com.ansel.service.ICustomerService • com.ansel.util.Result
司机信息管理	com.ansel.controller.DriverInfoController	实现司机信息管理相关业务逻辑： 1. 添加司机信息 2. 删除司机信息	• Spring Framework： ControllerAdvice、CrossOrigin、PathVariable

		3. 修改司机信息 4. 查询司机信息	、RequestMapping、RequestMethod、RequestParam • Spring Data JPA • com.ansel.bean: ExtraIncome、FinanceFee、ManageFee、PageRequest、
职工信息管理	com.ansel.controller.EmployeeController	实现职工信息管理相关业务逻辑: 1. 添加职工信息 2. 删除职工信息 3. 修改职工信息 4. 查询职工信息	• Spring Framework: RequestMapping、RequestMethod、RequestParam、RestController • Spring Data JPA • com.ansel.bean: Function、FunctionWithGroup、CustomerInfo
货运单合同管理	com.ansel.controller.GoodsBillControll	实现货运单合同相关业务逻辑: 1. *填写一份货运单合同*	• Spring Framework: RequestMappi

	er	2. *添加货物* 3. *查询所有运单* 4. *查询运单状态* 5. *通过 id 查询单个货运单* 6. *修改货运单* 7. *删除货运单* 8. 获取一个用户的待收货物 9. 获取所有未发过 {提货 | 到货 | 中转 | 已提 | 代收} 回告的运单 10. 获取所有已发过 {提货 | 到货 | 中转 | 已提 | 代收} 回告的运单 11. 获取已提货的运单	ng、RequestMethod、RequestParam、RestController • Spring Data JPA • io.swagger.annotations • com.ansel.service.ICustomerService • com.ansel.util.Result • com.ansel.bean: Function、FunctionWithGroup、CustomerInfo
司机回执信息管理	com.ansel.controller.GoodsReceiptController	实现司机回执信息管理相关业务: 添加回执信息	• Spring Framework: Autowired、ControllerAdvice、RequestMethod、RestController、CrossOrigin、ControllerAdvice、 • com.ansel.bean.GoodsReceiptInfo • com.ansel.ser

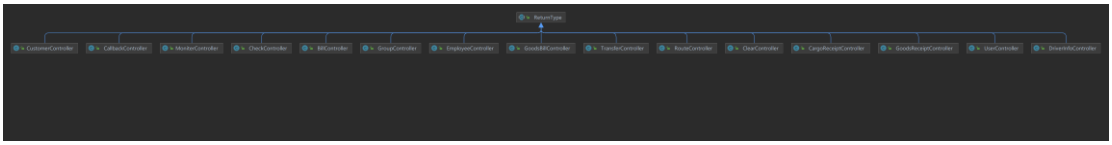
			vice.IGoodsReceiptService • io.swagger.annotations
用户组管理	com.ansel.controller.GroupController	实现管理用户组相关业务: 1. 新增用户组更新 2. 用户组信息 3. 删除用户组 4. 查询用户组及其权限	• Spring Framework: Autowired、ControllerAdvice、RequestMapping、RequestMethod、RestController、CrossOrigin、RequestParam、ControllerAdvice、PathVariable • com.ansel.bean: Function、FunctionWithGroup
监管管理	com.ansel.controller.MoniterController	实现监管相关业务: 1. 查询预期未到运单 2. 滞留未取运单 3. 打印客户用量 4. 打印司机用量 5. 打印往来业务用量 6. 打印专线整体 7. 打印车辆成本及其查询等操作	• Spring Framework: Autowired、ControllerAdvice、RequestMapping、RequestMethod、RestController、CrossOrigin、

			RequestParam、ControllerAdvice、PathVariable • com.ansel.bean: CarCost、ContactsService、DriverAmount、GoodsBill、LineOverall、IMoniterService、Result • io.swagger.annotations: @Api、ApiOperation
返回标识器	com.ansel.controller.ReturnTyper	定义返回结果常量，括成功指示符和失败指示符	NULL
路线管理	com.ansel.controller.RouteController	对城市扩充信息和路线信息的业务处理 1. 新增城市扩充信息 2. 更新一条城市信息 3. 得到所有城市 4. 得到无范围的城市 5. 得到一条的城市范围信息 6. 得到所有线路信息	• Spring Framework: Autowired、ControllerAdvice、RequestMapping、RequestMethod、RestController、CrossOrigin、RequestParam、Page、PageRequest

			、Pageable、ControllerAdvice、PathVariable • com.ansel.bean: CityExpand、Region、RouteInfo • com.ansel.service: ICityExpandService、IRouteService • com.ansel.util.Result • io.swagger.annotations: @Api、ApiOperation
运输管理	com.ansel.controller.TransferController	对中转信息、中转公司信息以及顾客回执信息的增删改查 1. 添加中转公司信息 2. 添加中转信息 3. 添加顾客回执 4. 查询中转公司信息-分页 5. 查询一个司机的所有运单(中转 | 到货) 6. 查询一个司机的所有到货运单 7. 查询运单的中转详情 8. 中转回告所需数据 9. 查询所有的中转信息 10. 查询所有用户的到货回执	• Spring Framework: Autowired、ControllerAdvice、RequestMapping、RequestMethod、RestController、CrossOrigin、RequestParam • com.ansel.bean: CustomerRec

			eiptInfo、GoodsBill、TransferComInfo • com.ansel.service.ITransferService • com.ansel.util.Result • io.swagger.annotations： @Api、ApiOperation
用户登录	com.ansel.controller.UserController	实现用户登录账户验证业务： 1. 用户登录验证 2. 用户修改密码	• Spring Framework： Autowired、ControllerAdvice、RequestMapping、RequestMethod、RestController、CrossOrigin • com.ansel.service.IUserService • io.swagger.annotations： ApiImplicitParam、@Api、ApiOperation

4.1.2 程序(模块)层次结构关系



4. 2 全局数据结构说明

4. 2. 1 常量

a. 数据文件名称及其所在目录

1. 货运合同文件目录：/app/data/consignment/.
2. 回执单文件目录：/app/data/receipt/.

b. 功能说明

1. 快递公司通过系统存储货运合同和回执单等信息。
2. 系统会根据各种操作生成各种单据用于记录、计费等使用。
3. 相关常量将用于定义文件路径，在整个系统中被反复使用。

c. 具体常量说明

1. CONSIGNMENT_DIR：String 类型，表示货运合同文件目录，长度为 50。
2. RECEIPT_DIR：String 类型， 表示回执单文件目录，长度为 50。

4. 2. 2 变量

a. 数据文件名称及其所在目录

1. 数据文件名称：shipment_data.csv
所在目录：C:/data/
2. 数据文件名称: transport_records.xlsx
所在目录：D:/project/records/

b. 功能说明

1. 快递公司通过系统存储货运合同和回执单等信息。
2. 系统会根据各种操作生成各种单据用于记录、计费等使用。

c.具体变量的功能说明:

1. data_file_path - 字符串，存储数据文件所在的完整路径和文件名
2. data_file_directory - 字符串，存储数据文件所在的目录路径
3. shipment_contract - 字典，存储发货客户与快递公司签订的货运合同信息，包括：发货客户、快递公司、货物描述、付款方式等字段
4. payment_method - 字符串，存储货运合同的付款方式

5. payment_amount - 浮点数，存储货运合同的支付金额
6. transport_contract - 字典，存储快递公司与司机签订的运输合同信息，包括：车牌号、司机姓名、运输线路、运费结算方式等字段
7. transport_mode - 字符串，存储货物运输的方式（如陆运、空运等）
8. freight_charge - 字典，存储运输合同的运费结算方式，包括：运费计算方法、单位价格、总价、付款方式等字段
9. driver_receipt - 字典，存储司机对货物进行验收后填写的回执单信息，包括：货物状况、验收时间、司机签名等字段
10. error_record - 字典，存储货物未通过验收时的差错记录信息，包括：原因、处理方式等字段
11. shipment_receipt - 字典，存储客户对发货物进行验收后填写的回执单信息，包括：货物状况、验收时间、客户签名等字段
12. transfer_info - 字典，存储货物中转信息，包括：原发地点、目的地点、中转时间、负责人姓名等字段

4. 2. 3 数据结构

- **`BillInfo`:**

1. 功能说明：单据明细表实体类
2. 具体数据结构说明：
    - id: 主键 id, 取值为 int 类型的整数
    - billType: 单据类型，取值为 String 类型的字符串，长度不超过 50 个字符。
    - billCode: 单据编号，取值为 String 类型的字符串，长度不超过 50 个字符。
    - billState: 单据状态, 取值为 String 类型的字符串，长度不超过 50 个字符。
    - writeDate: 创建时间，取值为 java.util.Date 类型的日期对象。
    - acceptStation: 单据承接站点，取值为 String 类型的字符串，长度不超过 50 个字符。

此外，该类还包括构造方法、设置器和访问器等操作数据的方法，以及重写的`toString()`方法用于返回详细信息

- **`BillRelease`**

1. 功能说明：单据分发表实体类
2. 具体数据结构说明：
    - id：主键 id，取值为 int 类型的整数
    - billType：单据类型，取值为 String 类型的字符串，长度不超过 50 个字符。
    - billCode：单据编号，取值为 String 类型的字符串，长度不超过 50 个字符。
    - receiveBillPerson：接收单据人，取值为 String 类型的字符串，长度不超过 50 个字符。
    - acceptStation：单据承接站点，取值为 String 类型的字符串，长度不超过 50 个字符。
    - receiveBillTime：接收单据时间，取值为 java.sql.Date 类型的日期对象。
    - releasePerson: 发布单据人，取值为 String 类型的字符串，长度不超过 50 个字符。

此外，该类还包括构造方法、设置器和访问器等操作数据的方法，以及重写的`toString()`方法用于返回详细信息。

- **`CallbackInfo`**

1. 功能说明：回告信息表实体类。
2. 具体数据结构说明：
    - id：主键 id，取值为 int 类型的整数。
    - dialNo：拨号器编号，取值为 String 类型的文本，长度不超过 50 个字符。
    - type：回告类型，取值为 String 类型的文本，长度不超过 50 个字符。
    - content：回告内容，取值为 String 类型的文本，长度不限。
    - goodsBillId：货运单 id，取值为 String 类型的文本，长度不超过 50 个字符。
    - writeTime：写入时间，取值为 java.sql.Date 类型的日期对象。

- writer：写入人，取值为 String 类型的文本，长度不超过 50 个字符。
- finallyDialTime：最后一次拨号时间，取值为 java.sql.Date 类型 的日期对象。
- success：是否成功，取值为 boolean 类型的布尔值。
- locked：记录是否被锁定，取值为 boolean 类型的布尔值。
- billId：回执 Id，取值为 String 类型的文本，长度不超过 50 个字符。
- billType：回执类型，取值为 String 类型的文本，长度不超过 50 个字符。

此外，该类还包括默认构造方法和全参构造方法、以及设置器、访问器和重写的`toString()`方法，用于返回详细信息。

- **`CarCost`**

1. 功能说明：表示车辆成本的相关信息，包括司机名称、车号、车型、准载重量、车厢宽度、载物高度、承运费用、加运费用、实际总运费、装货地点、交货地点、回执单编号和结算时间等
2. 具体数据结构说明：

driverCode：String 类型，表示司机名称，长度为 50。

- carNo：String 类型，表示车号，长度为 50。
- carType：String 类型，表示车型，长度为 50。
- allowCarryWeight：double 类型，表示准载重量，取值为正数。
- carWidth：String 类型，表示车厢宽度。
- goodsHeight：String 类型，表示载物高度。
- carryFeeTotal：double 类型，表示承运费用，取值为正数。
- addCarriageTotal：double 类型，表示加运费用，取值为正数。
- factCarriageTotal：double 类型，表示实际总运费，取值为正数。
- loadStation：String 类型，表示装货地点。
- dealGoodsStation：String 类型，表示交货地点。
- backBillCode：String 类型，表示回执单编号。
- balanceTime：Date 类型，表示结算时间，以日期格式存储。

4.3 CSCI 部件

a.标识构成该 CSCI 的所有软件配置项。应赋予每个软件配置项一个项目唯一标识符。

1. BillController 模块: CSCI-SC-001
2. CallbackController 模块: CSCI-SC-002
3. CargoReceiptController 模块: CSCI-SC-003
4. CheckController 模块: CSCI-SC-004
5. ClearController 模块: CSCI-SC-005
6. CustomerController 模块: CSCI-SC-006
7. DriverInfoController 模块: CSCI-SC-007
8. EmployeeController 模块: CSCI-SC-008
9. GoodsBillController 模块: CSCI-SC-009
10. GoodsReceiptController 模块: CSCI-SC-010
11. GroupController 模块: CSCI-SC-011
12. MoniterController 模块: CSCI-SC-012
13. ReturnTyper 模块: CSCI-SC-013
14. RouteController 模块: CSCI-SC-014
15. TransferController 模块: CSCI-SC-015
16. UserController 模块: CSCI-SC-016

b.给出软件配置项的静态关系(如“组成”)。

以下是软件配置项之间的静态关系：

1. CSCI-SC-001 (BillController 模块):
    - 组成：Bill 相关业务逻辑类、Bill 相关数据对象、Bill 相关数据库表
    - 依赖：CallbackController 模块、GoodsBillController 模块
2. CSCI-SC-002 (CallbackController 模块):
    - 组成：Callback 相关业务逻辑类、Callback 相关数据对象、Callback 相关数据库表
    - 依赖：GoodsReceiptController 模块、GoodsBillController 模块

3. CSCI-SC-003 (CargoReceiptController 模块):
    - 组成：CargoReceipt 相关业务逻辑类、CargoReceipt 相关数据对象、CargoReceipt 相关数据库表
    - 依赖：GoodsBillController 模块、CustomerController 模块
4. CSCI-SC-004 (CheckController 模块):
    - 组成：Check 相关业务逻辑类、Check 相关数据对象、Check 相关数据库表
    - 依赖：EmployeeController 模块
5. CSCI-SC-005 (ClearController 模块):
    - 组成：Clear 相关业务逻辑类、Clear 相关数据对象、Clear 相关数据库表
    - 依赖：GoodsBillController 模块、CustomerController 模块
6. CSCI-SC-006 (CustomerController 模块):
    - 组成：Customer 相关业务逻辑类、Customer 相关数据对象、Customer 相关数据库表
7. CSCI-SC-007 (DriverInfoController 模块):
    - 组成：DriverInfo 相关业务逻辑类、DriverInfo 相关数据对象、DriverInfo 相关数据库表
8. CSCI-SC-008 (EmployeeController 模块):
    - 组成：Employee 相关业务逻辑类、Employee 相关数据对象、Employee 相关数据库表
9. CSCI-SC-009 (GoodsBillController 模块):
    - 组成：GoodsBill 相关业务逻辑类、GoodsBill 相关数据对象、GoodsBill 相关数据库表
    - 依赖：RouteController 模块
10. CSCI-SC-010 (GoodsReceiptController 模块):
    - 组成：GoodsReceipt 相关业务逻辑类、GoodsReceipt 相关数据对象、GoodsReceipt 相关数据库表
    - 依赖：DriverInfoController 模块
11. CSCI-SC-011 (GroupController 模块):
    - 组成：Group 相关业务逻辑类、Group 相关数据对象、Group 相关数据库表

12. CSCI-SC-012 (MoniterController 模块):
    - 组成：Moniter 相关业务逻辑类、Moniter 相关数据对象、Moniter 相关数据库表
13. CSCI-SC-013 (ReturnTyper 模块):
    - 组成：ReturnTyper 相关类、ReturnTyper 相关常量
14. CSCI-SC-014 (RouteController 模块):
    - 组成：Route 相关业务逻辑类、Route 相关数据对象、Route 相关数据库表
15. CSCI-SC-015 (TransferController 模块):
    - 组成：Transfer 相关业务逻辑类、Transfer 相关数据对象、Transfer 相关数据库表
16. CSCI-SC-016 (UserController 模块):
    - 组成：User 相关业务逻辑类、User 相关数据对象、User 相关数据库表

c.陈述每个软件配置项的用途，并标识分配给它的 CSCI 需求与 CSCI 级设计决策

模块	用途	分配的 CSCI 需求和 CSCI 级设计决策
BillController	处理单据相关的请求和业务逻辑	单据管理功能的实现、单据分发逻辑、运单查询逻辑等
CallbackController	处理回告相关的业务请求	回告信息的添加和查询功能的实现
CargoReceiptController	处理与货运单回执相关的请求和业务逻辑	货运回执单的填写、查询、修改、删除功能的实现，运单状态查询等
CheckController	处理与营业外收入、财务费用、管理费用和员工工资相关的请求和业务逻辑	营业外收入、财务费用、管理费用和员工工资的录入、查询和管理功能的实现
ClearController	分配的 CSCI 需求和 CSCI 级设计决策：司机结算实体、客户结算实体、代收结算实体和杂费结算实体的查	司机结算实体、客户结算实体、代收结算实体和杂费结算实体的查询、添加和管理功能的实现

	询、添加和管理功能的实现	
CustomerController	实现客户信息管理相关业务逻辑	顾客信息的添加、删除、查询和更新功能的实现
DriverInfoController	实现司机信息管理相关业务逻辑	司机信息的添加、删除、修改和查询功能的实现
EmployeeController	实现职工信息管理相关业务逻辑	职工信息的添加、删除、修改和查询功能的实现
GoodsBillController	实现货运单合同相关业务逻辑	货运单合同的填写、货物的添加、运单状态查询等功能的实现
GoodsReceiptController	实现司机回执信息管理相关业务	司机回执信息的添加功能的实现
GroupController	实现管理用户组相关业务	用户组的新增、更新、删除等功能的实现，用户组信息的查询，以及用户组权限的管理
MoniterController	实现监管相关业务	预期未到运单的查询，滞留未取运单的查询，打印客户用量、司机用量、往来业务用量、专线整体的功能实现
ReturnTyper	定义返回结果常量，包括成功指示符和失败指示符	定义返回结果常量的方式和结构
RouteController	处理城市扩充信息和路线信息的业务处理	城市扩充信息的新增、更新，城市信息和路线信息的查询
TransferController	处理中转信息、中转公司信息以及顾客回执信息的增删改查	中转公司信息的添加，中转信息的添加和查询，顾客回执的添加和查询等功能的实现
UserController	实现用户登录账户验证业务	用户登录验证、用户密码修改等功能的实现

d.标识每个软件配置项的开发状态/类型

模块	开发状态/类型
BillController	新开发的软件配置项
CallbackController	新开发的软件配置项
CargoReceiptController	新开发的软件配置项
CheckController	新开发的软件配置项
ClearController	新开发的软件配置项
CustomerController	新开发的软件配置项
DriverInfoController	新开发的软件配置项
EmployeeController	新开发的软件配置项
GoodsBillController	新开发的软件配置项
GoodsReceiptController	新开发的软件配置项
GroupController	新开发的软件配置项
MoniterController	新开发的软件配置项
ReturnTyper	新开发的软件配置项
RouteController	新开发的软件配置项
TransferController	新开发的软件配置项
UserController	新开发的软件配置项

e.描述 CSCI(若适用，每个软件配置项)计划使用的计算机硬件资源

1. ClearController 模块：

- 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问网络设备以进行与报关、税费计算和通关手续相关的通信。
- 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理清关相关的业务逻辑。假设网络设备提供了适当的通信能力和连接性。
- 影响使用的特殊考虑：在处理大量清关数据时，可能需要考虑使用虚存、多处理器或其他优化策略来提高性能和响应时间。
- 度量单位：处理器能力百分比、内存字节数、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

2. CustomerController 模块：
   - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与客户信息和管理相关的操作。
   - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理客户信息和管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
   - 影响使用的特殊考虑：在处理大量客户信息和管理数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
   - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
   - 进行评估或度量的级别：CSCI 或软件配置项。
3. DriverInfoController 模块：
   - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与司机信息和调度相关的操作。
   - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理司机信息和调度的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
   - 影响使用的特殊考虑：在处理大量司机信息和调度数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
   - 度量单位：处理器能力百分度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。

- 进行评估或度量的级别：CSCI 或软件配置项。

4. EmployeeController 模块：

   - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与员工信息和薪资管理相关的操作。

   - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理员工信息和薪资管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。

   - 影响使用的特殊考虑：在处理大量员工信息和薪资管理数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。

   - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。

   - 进行评估或度量的级别：CSCI 或软件配置项。

5. GoodsBillController 模块：

   - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与货运单据相关的操作。

   - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理货运单据的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。

   - 影响使用的特殊考虑：在处理大量货运单据数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。

   - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。

   - 进行评估或度量的级别：CSCI 或软件配置项。

6. CustomerController 模块：

   - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与客户信息和管理相关的操作。

   - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理客户信息和管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。

   - 影响使用的特殊考虑：在处理大量客户信息和管理数据时，可能需要考虑使

用数据库优化、网络优化或其他技术来提高性能和可靠性。

- 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

7. DriverInfoController 模块：

- 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与司机信息和调度相关的操作。
- 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理司机信息和调度的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
- 影响使用的特殊考虑：在处理大量司机信息和调度数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
- 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

8. EmployeeController 模块：

- 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与员工信息和薪资管理相关的操作。
- 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理员工信息和薪资管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
- 影响使用的特殊考虑：在处理大量员工信息和薪资管理数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
- 度量单位：处理器能力百分比、内存字节数、数据库存。
- 进行评估或度量的级别：CSCI 或软件配置项。

9. GoodsBillController 模块：

- 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与货运单据相关的操作。
- 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理货运单据的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能

力。

- 影响使用的特殊考虑：在处理大量货运单据数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
- 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

10. GoodsReceiptController 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与货物收货相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理货物收货的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
    - 影响使用的特殊考虑：在处理大量货物收货数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
    - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
    - 进行评估或度量的级别：CSCI 或软件配置项。
11. GroupController 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与组管理相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理组管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
    - 影响使用的特殊考虑：在处理大量组管理数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
    - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
    - 进行评估或度量的级别：CSCI 或软件配置项。
12. MoniterController 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问监控设备和网络设备以进行监控相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量

来处理监控功能的业务逻辑。假设监控设备和网络设备提供了适当的存储和通信能力。

- 影响使用的特殊考虑：在处理大量监控数据和实时监控时，可能需要考虑使用适当的硬件加速或优化技术来提高性能和实时性。
- 度量单位：处理器能力百分比、内存字节数、监控设备数量、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

13. ReturnTyper 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和其他系统以进行退货类型相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理退货类型的业务逻辑。假设数据库和其他系统提供了适当的存储和通信能力。
    - 影响使用的特殊考虑：在处理大量退货类型数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
    - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
    - 进行评估或度量的级别：CSCI 或软件配置项。
14. RouteController 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问地图数据和网络设备以进行路径规划和导航相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理路径规划和导航的业务逻辑。假设地图数据和网络设备提供了适当的存储和通信能力。
    - 影响使用的特殊考虑：在处理复杂的路径规划和导航任务时，可能需要考虑使用适当的算法和技术来提高性能和实时性。另外，还需要考虑地图数据的更新和网络连接的稳定性，以确保路线规划和导航的准确性和可靠性。
    - 度量单位：处理器能力百分比、内存字节数、地图数据大小、网络带宽、导航准确性等。
    - 进行评估或度量的级别：CSCI 或软件配置项。
15. TransferController 模块：

- 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与转运操作相关的功能。
- 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理转运操作的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
- 影响使用的特殊考虑：在处理大量转运数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
- 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
- 进行评估或度量的级别：CSCI 或软件配置项。

16. UserController 模块：
    - 计算机硬件资源：该模块需要使用一个具有适当处理能力和内存容量的计算机系统来支持其运行。它可能需要访问数据库和网络设备以进行与用户管理相关的操作。
    - 使用数据基于的假设和条件：假设系统提供了足够的处理器能力和内存容量来处理用户管理的业务逻辑。假设数据库和网络设备提供了适当的存储和通信能力。
    - 影响使用的特殊考虑：在处理大量用户管理数据时，可能需要考虑使用数据库优化、网络优化或其他技术来提高性能和可靠性。
    - 度量单位：处理器能力百分比、内存字节数、数据库存储容量、网络带宽等。
    - 进行评估或度量的级别：CSCI 或软件配置项。

f. 指出实现每个软件配置项的软件放置在哪个程序库中。

1. BillController 模块：放置在控制器库中。
2. CallbackController 模块：放置在控制器库中。
3. CargoReceiptController 模块：放置在控制器库中。
4. CheckController 模块：放置在控制器库中。
5. ClearController 模块：放置在控制器库中。
6. CustomerController 模块：放置在控制器库中。
7. DriverInfoController 模块：放置在控制器库中。
8. EmployeeController 模块：放置在控制器库中。

9. GoodsBillController 模块：放置在控制器库中。
10. GoodsReceiptController 模块：放置在控制器库中。
11. GroupController 模块：放置在控制器库中。
12. MoniterController 模块：放置在控制器库中。
13. ReturnTyper 模块：放置在工具库中。
14. RouteController 模块：放置在控制器库中。
15. TransferController 模块：放置在控制器库中。
16. UserController 模块：放置在控制器库中。

4. 4 执行概念

执行概念是指软件配置项在运行期间之间的动态交互和行为。下面是各个软件配置项之间的执行概念描述：

1. GroupController 模块的执行概念：
    - 执行控制流：GroupController 模块负责协调和管理其他模块的执行顺序和流程。
    - 数据流：GroupController 模块接收和发送数据给其他模块，以实现组管理和任务调度等功能。
    - 动态控制序列：GroupController 模块根据系统状态和输入信息，动态决定执行顺序和调度策略。
    - 优先关系：GroupController 模块可能根据任务的优先级和紧急程度，对不同模块的执行进行优先级排序和调度。
    - 并发执行：GroupController 模块可以与其他模块并发执行，以提高系统的响应性和并行处理能力。
2. GoodsReceiptController 模块的执行概念：
    - 执行控制流：GoodsReceiptController 模块负责货物收货的控制流程，包括货物装卸、入库处理和质量检查等步骤。
    - 数据流：GoodsReceiptController 模块接收货物信息和仓库状态等数据，并将处理结果反馈给其他模块或系统。
    - 状态转换图：GoodsReceiptController 模块可能维护一些状态信息，用于跟踪货物收货过程中的状态转换。

- 异常处理：GoodsReceiptController 模块可能处理异常情况，如货物损坏、缺货等，并采取相应的措施进行处理。

3. ClearController 模块的执行概念：

   - 执行控制流：ClearController 模块负责清关流程的控制和协调，包括报关、税费计算和通关手续等步骤。

   - 数据流：ClearController 模块接收相关的报关和税费数据，执行相应的清关操作，并将处理结果反馈给相关模块或系统。

   - 中断处理：ClearController 模块可能会处理中断事件，如报关系统故障、通关手续被拒绝等，并进行相应的中断处理流程。

4. CustomerController 模块的执行概念：

   - 执行控制流：CustomerController 模块负责处理客户信息和管理相关的业务逻辑，包括客户注册、查询和管理等功能的控制流程。

   - 数据流：CustomerController 模块接收客户信息的输入，进行相关操作并提供查询结果，同时与其他模块进行数据交互，如与 OrderController 模块进行订单信息的关联。

   - 状态转换图：CustomerController 模块可能维护客户状态的变化，如客户状态的更新、活跃状态的切换等。

   - 异常处理：CustomerController 模块可能处理客户相关的异常情况，如重复注册、无效的客户信息等，并进行相应的异常处理逻辑。

5. DriverInfoController 模块的执行概念：

   - 执行控制流：DriverInfoController 模块负责处理司机信息和管理相关的业务逻辑，包括司机信息录入、查询和调度等功能的控制流程。

   - 数据流：DriverInfoController 模块接收司机信息的输入，进行相关操作并提供查询结果，同时与其他模块进行数据交互，如与 RouteController 模块进行路线调度信息的关联。

   - 中断处理：DriverInfoController 模块可能会处理中断事件，如司机调度变更、司机临时不可用等，并进行相应的中断处理流程。

6. GoodsBillController 模块的执行概念：

   - 执行控制流：GoodsBillController 模块负责处理货运单据相关的业务逻辑，包括货物装运、运输跟踪和签收确认等功能的控制流程。

   - 数据流：GoodsBillController 模块接收货运单据的输入，进行相关操作并提供

运输跟踪和签收确认的结果，同时与其他模块进行数据交互，如与 TransferController 模块进行货物转运信息的关联。

- 状态转换图：GoodsBillController 模块可能维护货运单据的状态转换，如装运状态、运输状态和签收状态的更新。

- 异常处理：GoodsBillController 模块可能处理货物运输过程中的异常情况，如货物丢失、运输延误等，并进行相应的异常处理逻辑。

7. GoodsReceiptController 模块的执行概念：

   - 执行控制流：GoodsReceiptController 模块负责处理货物收货相关的业务逻辑，包括接收货物、验收、入库等功能的控制流程。

   - 数据流：GoodsReceiptController 模块接收货物信息的输入，进行相关操作并更新库存信息，同时与其他模块进行数据交互，如与 InventoryController 模块进行库存管理的关联。

   - 状态转换图：GoodsReceiptController 模块可能维护货物收货状态的变化，如货物收货状态的更新、入库状态的转换等。

8. ClearController 模块的执行概念：

   - 执行控制流：ClearController 模块负责处理清关相关的业务逻辑，包括报关、税费计算和通关手续等功能的控制流程。

   - 数据流：ClearController 模块接收报关信息和税费计算的输入，进行相关操作并更新清关状态，同时与其他模块进行数据交互，如与 FinanceController 模块进行费用结算的关联。

   - 异常处理：ClearController 模块可能处理清关过程中的异常情况，如报关失败、税费计算错误等，并进行相应的异常处理逻辑。

9. MoniterController 模块的执行概念：

   - 执行控制流：MoniterController 模块负责监控系统运行状态和业务数据，包括监控系统日志、性能指标和异常事件等功能的控制流程。

   - 数据流：MoniterController 模块获取系统日志、性能指标和异常事件的数据流，并对其进行分析和处理，同时与其他模块进行数据交互，如向 AdminController 模块报告系统异常事件。

   - 中断处理：MoniterController 模块可能会处理系统中断事件，如系统崩溃、硬件故障等，并进行相应的中断处理流程。

10. ReturnTyper 模块的执行概念：

    - 执行控制流：ReturnTyper 模块负责处理退货管理相关的业务逻辑，包括退货

申请、审批和退款等功能的控制流程。

- 数据流：ReturnTyper 模块接收退货申请和退款请求的输入，进行相关操作并更新退货状态和退款记录，同时与其他模块进行数据交互，如与 FinanceController 模块进行退款流程的关联。

11. RouteController 模块的执行概念：

    - 执行控制流：RouteController 模块负责处理路径规划和导航相关的业务逻辑，包括根据起始点和目的地计算最优路径、生成导航指示等功能的控制流程。

    - 数据流：RouteController 模块接收起始点和目的地的输入，进行路径规划和导航操作，并输出最优路径和导航指示，同时与其他模块进行数据交互，如与 MapController 模块获取地图数据进行路线计算。

    - 异常处理：RouteController 模块可能处理路径规划和导航过程中的异常情况，如无法找到合适路径、导航失败等，并进行相应的异常处理逻辑。

12. TransferController 模块的执行概念：

    - 执行控制流：TransferController 模块负责处理货物转运相关的业务逻辑，包括货物转运计划、调度和跟踪等功能的控制流程。

    - 数据流：TransferController 模块接收货物转运计划和调度的输入，进行相关操作并更新转运状态和位置信息，同时与其他模块进行数据交互，如与 DriverInfoController 模块进行司机调度的关联。

    - 并发执行：TransferController 模块可能需要支持并发处理多个货物转运任务，确保高效的调度和跟踪。

13. UserController 模块的执行概念：

    - 执行控制流：UserController 模块负责处理用户信息和权限管理相关的业务逻辑，包括用户注册、登录和权限控制等功能的控制流程。

    - 数据流：UserController 模块接收用户注册和登录的输入，进行相关操作并验证用户身份和权限，同时与其他模块进行数据交互，如与 AuthController 模块进行权限验证的关联。

    - 对象/进程/任务的动态创建与删除：UserController 模块可能涉及用户对象的动态创建和删除，包括注册新用户和注销用户的操作。

14. RouteController 模块的执行概念：

    - 执行控制流：RouteController 模块负责处理路径规划和导航相关的业务逻辑，包括路线规划、导航指引和实时路况更新等功能的控制流程。

    - 数据流：RouteController 模块接收起点和终点位置的输入，进行路径规划并

生成导航指引，同时与其他模块进行数据交互，如与 MapController 模块获取地图数据和与 GPSController 模块接收实时位置信息。

- 时序图：RouteController 模块与 MapController 模块和 GPSController 模块之间可能存在时序关系，例如在接收到位置更新后触发路径重新规划。

- 异常处理：RouteController 模块可能会处理路径规划和导航过程中的异常情况，如无法找到合适路径、导航错误等，并进行相应的异常处理逻辑。

15. TransferController 模块的执行概念：

    - 执行控制流：TransferController 模块负责处理货物转运相关的业务逻辑，包括货物配载、转运跟踪和交接确认等功能的控制流程。

    - 数据流：TransferController 模块接收货物配载和转运信息的输入，进行相关操作并更新转运状态，同时与其他模块进行数据交互，如与 LogisticsController 模块进行货物跟踪和与 DriverInfoController 模块进行司机调度的关联。

    - 状态转换图：TransferController 模块可能维护货物转运状态的变化，如货物配载状态的更新、转运完成状态的转换等。

    - 并发执行：TransferController 模块可能需要支持多个货物转运任务的并发执行，确保转运过程的并行处理。

16. UserController 模块的执行概念：

    - 执行控制流：UserController 模块负责处理用户管理相关的业务逻辑，包括用户注册、登录和权限管理等功能的控制流程。

    - 数据流：UserController 模块接收用户注册和登录信息的输入，进行身份验证和权限控制，并与其他模块进行数据交互，如与 EmployeeController 模块获取员工信息和与 GroupController 模块进行用户权限分配的关联。

    - 中断处理：UserController 模块可能会处理用户操作中的异常情况，如无效的登录凭证、权限不足等，并进行相应的中断处理流程。

    - 动态创建与删除：UserController 模块可能会根据用户注册和注销的需求，动态创建和删除用户账户，同时更新相关的用户权限和信息。

4. 5 接口设计

4. 5. 1 接口标识与接口图

a.由接口实体分配给接口的优先级

1. 用户界面接口 (UserInterfaceInterface)
2. 数据库接口 (DatabaseInterface)
3. 货物收货接口 (GoodsReceiptInterface)
4. 清关接口 (ClearInterface)
5. 监控接口 (MonitoringInterface)
6. 退货管理接口 (ReturnManagementInterface)
7. 路径规划接口 (RoutePlanningInterface)
8. 运输管理接口 (TransportationManagementInterface)
9. 车辆调度接口 (VehicleSchedulingInterface)
10. 员工管理接口 (EmployeeManagementInterface)
11. 薪资管理接口 (SalaryManagementInterface)
12. 财务管理接口 (FinanceManagementInterface)
13. 客户信息管理接口 (CustomerInfoManagementInterface)
14. 司机信息管理接口 (DriverInfoManagementInterface)
15. 库存管理接口 (InventoryManagementInterface)
16. 安全管理接口 (SecurityManagementInterface)

b.要实现的接口的类型

1. 实时数据传输接口：用于实时传输数据，通常在系统之间进行实时通信和数据交换。
2. 存储与检索接口：用于数据的存储和检索操作，包括对数据库、文件系统或其他数据存储设备的读写操作。
3. 控制接口：用于控制系统的各种功能和操作，例如启动、停止、暂停、重置等。
4. 外部服务接口：用于与外部服务或第三方系统进行集成和通信，包括调用外部 API、访问 Web 服务等。
5. 用户界面接口：用于与用户进行交互，包括用户输入和输出、显示界面、用户操作反馈等。
6. 安全接口：用于实现系统的安全性和权限控制，包括用户认证、访问控制、数据加密等。
7. 配置管理接口：用于管理系统的配置参数和设置，包括读取和修改配置项、配置文件的加载和保存等。
8. 事件通知接口：用于发送和接收系统事件通知，包括错误、警告、状态变更等系

统事件的通知和处理。

9. 异步消息接口：用于异步消息的发送和接收，支持系统之间的松耦合通信和消息传递。

10. 日志记录接口：用于系统日志的记录和管理，包括日志的生成、存储、查询和分析等功能。

c.接口实体将提供、存储、发送、访问、接收的单个数据元素的特性

1. CustomerController 模块接口：
    - 唯一标识符：CSCI-IF-CUSTOMER
    - 接口实体：CustomerController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与客户信息交互和管理的功能，包括客户注册、查询和管理等操作。
        - 数据库接口：与数据库进行交互，查询和更新客户信息。
2. DriverInfoController 模块接口：
    - 唯一标识符：CSCI-IF-DRIVERINFO
    - 接口实体：DriverInfoController 模块与 UserInterface 模块、DatabaseController 模块、TransferController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与司机信息交互和管理的功能，包括司机信息录入、查询和调度等操作。
        - 数据库接口：与数据库进行交互，查询和更新司机信息。
        - 货物转运接口：与货物转运模块进行接口，调度司机进行货物的运输。
3. EmployeeController 模块接口：
    - 唯一标识符：CSCI-IF-EMPLOYEE
    - 接口实体：EmployeeController 模块与 UserInterface 模块、DatabaseController 模块、PayrollController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与员工信息交互和管理的功能，包括员工信息录入、查询和薪资管理等操作。

        - 数据库接口：与数据库进行交互，查询和更新员工信息。
        - 薪资管理接口：与薪资管理模块进行接口，处理员工薪资计算和发放。
4. GoodsBillController 模块接口：
    - 唯一标识符：CSCI-IF-GOODSBILL
    - 接口实体：GoodsBillController 模块与 UserInterface 模块、DatabaseController 模块、TransferController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与货运单据交互和管理的功能，包括货物装运、运输跟踪和签收确认等操作。
        - 数据库接口：与数据库进行交互，查询和更新货运单据信息。
        - 货物转运接口：与货物转运模块进行接口，进行货物的装运和运输跟踪。
5. GoodsReceiptController 模块接口：
    - 唯一标识符：CSCI-IF-GOODSRECEIPT
    - 接口实体：GoodsReceiptController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与货物收货交互和管理的功能，包括接收货物、验收、入库等操作。
        - 数据库接口：与数据库
6. GroupController 模块接口：
    - 唯一标识符：CSCI-IF-GROUP
    - 接口实体：GroupController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
    - 特性：
        - 用户界面接口：用户界面提供与用户组交互和管理的功能，包括用户组创建、权限管理和成员管理等操作。
        - 数据库接口：与数据库进行交互，查询和更新用户组信息。
7. MoniterController 模块接口：
    - 唯一标识符：CSCI-IF-MONITER

- 接口实体：MoniterController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供显示系统监控信息和处理异常事件的功能。
  - 数据库接口：与数据库进行交互，查询系统日志和性能指标。

8. ReturnTyper 模块接口：
   - 唯一标识符：CSCI-IF-RETURN
   - 接口实体：ReturnTyper 模块与 UserInterface 模块、DatabaseController 模块、FinanceController 模块之间的接口
   - 特性：
     - 用户界面接口：用户界面提供与退货管理交互和管理的功能，包括退货申请、审批和退款等操作。
     - 数据库接口：与数据库进行交互，查询和更新退货信息。
     - 退款流程接口：与财务管理模块进行接口，处理退款流程和记录退款信息。
9. RouteController 模块接口：
   - 唯一标识符：CSCI-IF-ROUTE
   - 接口实体：RouteController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
   - 特性：
     - 用户界面接口：用户界面提供路径规划和导航功能的交互和管理，包括输入起始点和目的地、获取最佳路径、显示导航指引等操作。
     - 数据库接口：与数据库进行交互，查询地图数据和存储路径信息。
10. TransferController 模块接口：
    - 唯一标识符：CSCI-IF-TRANSFER
    - 接口实体：TransferController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
    - 特性：
      - 用户界面接口：用户界面提供货物转运管理的交互和操作，包括货物调度、运输跟踪和状态更新等功能。
      - 数据库接口：与数据库进行交互，查询和更新货物转运信息。

11. UserController 模块接口：

- 唯一标识符：CSCI-IF-USER
- 接口实体：UserController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供用户管理和权限控制的交互和操作，包括用户注册、登录、密码重置和权限设置等功能。
  - 数据库接口：与数据库进行交互，查询和更新用户信息。

12. GoodsReceiptController 模块接口：

- 唯一标识符：CSCI-IF-GOODSRECEIPT
- 接口实体：GoodsReceiptController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供货物收货管理的交互和操作，包括接收货物、验收、入库等功能的界面展示和用户输入。
  - 数据库接口：与数据库进行交互，查询和更新货物收货信息、更新库存状态。

13. GroupController 模块接口：

- 唯一标识符：CSCI-IF-GROUP
- 接口实体：GroupController 模块与 UserInterface 模块、DatabaseController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供组管理功能的交互和操作，包括创建组、添加成员、管理权限等功能的界面展示和用户输入。
  - 数据库接口：与数据库进行交互，查询和更新组信息、更新成员列表。

14. MoniterController 模块接口：

- 唯一标识符：CSCI-IF-MONITER
- 接口实体：MoniterController 模块与 UserInterface 模块、LogController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供系统监控和日志查看的交互和操作，包括展示

系统日志、性能指标、异常事件等功能。

- 日志接口：与日志控制器模块进行交互，获取系统日志和异常事件的信息。

15. ReturnTyper 模块接口：

- 唯一标识符：CSCI-IF-RETURNTYPER
- 接口实体：ReturnTyper 模块与 UserInterface 模块、DatabaseController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供退货管理功能的交互和操作，包括退货申请、审批和退款等功能的界面展示和用户输入。
  - 数据库接口：与数据库进行交互，查询和更新退货信息、更新退款记录。

16. RouteController 模块接口：

- 唯一标识符：CSCI-IF-ROUTE
- 接口实体：RouteController 模块与 UserInterface 模块、MapController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供路径规划和导航功能的交互和操作，包括输入起始点和目的地、显示路径规划结果、提供导航指引等功能。
  - 地图接口：与地图控制器模块进行交互，获取地图数据、进行路径规划和导航算法的调用。

17. TransferController 模块接口：

- 唯一标识符：CSCI-IF-TRANSFER
- 接口实体：TransferController 模块与 UserInterface 模块、InventoryController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供货物转运管理的交互和操作，包括选择转运货物、调度运输资源、更新转运状态等功能的界面展示和用户输入。
  - 库存接口：与库存控制器模块进行交互，查询和更新货物转运的库存状态、更新库存记录。

18. UserController 模块接口：

- 唯一标识符：CSCI-IF-USER

- 接口实体：UserController 模块与 UserInterface 模块、AuthenticationController 模块之间的接口
- 特性：
  - 用户界面接口：用户界面提供用户管理功能的交互和操作，包括用户注册、登录、权限管理等功能的界面展示和用户输入。
  - 认证接口：与认证控制器模块进行交互，验证用户身份、管理用户权限等。

d.接口实体将提供、存储、发送、访问、接收的数据元素集合体(记录、消息、文件、数组、显示、报表等)的特性

见上一点的内容

e.接口实体为该接口使用通信方法的特性

接口实体使用通信方法的特性是描述接口在通信层面上的特性，包括通信链路、消息格式化、流控制、数据传输率、路由寻址、传输服务和安全性等方面。以下是对这些特性的描述：

1. 项目唯一标识符：每个接口实体应具有唯一的标识符，以便在系统中进行唯一标识和识别。
2. 通信链路/带宽/频率/媒体及其特性：描述接口实体所使用的通信链路类型、带宽、频率以及通信媒体的特性，如有线、无线、光纤等。
3. 消息格式化：定义接口实体之间交换的消息格式，包括数据结构、字段、编码方式等，以确保通信的正确解析和处理。
4. 流控制：描述接口实体之间的流量控制机制，包括序列编号、缓冲区分配等，以确保数据的可靠传输和避免数据丢失或溢出。
5. 数据传输率、周期或非周期和传送间隔：定义数据传输的速率、周期性或非周期性，以及传送数据之间的时间间隔。
6. 路由、寻址及命名约定：描述接口实体之间的路由选择、寻址机制以及命名约定，确保数据能够准确地传递到目标接口实体。
7. 传输服务：定义传输过程中的服务质量要求，包括优先级、等级、可靠性、时延等，以满足不同类型数据传输的需求。
8. 安全性/保密性/私密性考虑：描述接口实体在通信过程中的安全性、保密性和私密性考虑，如加密算法、用户鉴别、数据隔离、审计等，以确保通信的安全性和机密性。

f.接口实体为该接口使用协议的特性

1. UserInterface 模块接口实体：

- 项目唯一标识符：CSCI-IF-UI
- 协议的优先级/层：UI 模块作为用户与系统交互的接口，其优先级/层较高，负责接收用户输入、显示界面和信息反馈。
- 分组：UI 模块负责对用户输入进行分段与重组，将用户操作解析为系统可理解的指令，同时将系统的信息展示给用户。
- 合法性检查、错误控制、恢复过程：UI 模块负责对用户输入进行合法性检查，处理错误情况并提供适当的错误控制和恢复过程，确保系统与用户之间的交互正常进行。
- 同步：UI 模块通过连接的建立、保持和终止来实现与用户的同步，确保用户输入与系统的响应一致性。
- 状态、标识和其他报告特性：UI 模块负责报告系统状态、标识用户身份和权限，并提供其他相关信息的报告特性。

2. MapController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-MAP
   - 协议的优先级/层：MapController 模块作为地图控制器，其优先级/层较低，负责地图数据的处理和提供路径规划与导航的功能。
   - 分组：MapController 模块负责将地图数据进行分段与重组，以支持路径规划和导航算法的运行。
   - 合法性检查、错误控制、恢复过程：MapController 模块负责对地图数据进行合法性检查，并处理路径规划和导航过程中的错误情况，并提供适当的错误控制和恢复过程。
   - 同步：MapController 模块通过与 UserInterface 模块和 RouteController 模块的同步，确保路径规划和导航的准确性和一致性。
   - 状态、标识和其他报告特性：MapController 模块负责报告地图状态、标识路径规划和导航结果，并提供其他相关信息的报告特性。
3. InventoryController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-INVENTORY
   - 协议的优先级/层：InventoryController 模块作为库存管理控制器，其优先级/层较高，负责管理货物的入库、出库和库存信息的更新。
   - 分组：InventoryController 模块负责对货物信息进行分段与重组，以支持库存管理的操作和数据传输。

- 合法性检查、错误控制、恢复过程：InventoryController 模块负责对入库、出库操作进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，确保库存数据的准确性和一致性。
- 同步：InventoryController 模块与 GoodsReceiptController 模块、GoodsBillController 模块和 ReturnTyper 模块等相互同步，以确保库存信息与货物收发过程的同步更新。
- 状态、标识和其他报告特性：InventoryController 模块负责报告库存状态、标识货物位置和库存变动情况，并提供其他相关信息的报告特性。

4. FinanceController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-FINANCE
   - 协议的优先级/层：FinanceController 模块作为财务管理控制器，其优先级/层较高，负责处理费用结算、退款和报表生成等功能。
   - 分组：FinanceController 模块负责对财务数据进行分段与重组，以支持财务管理的操作和数据传输。
   - 合法性检查、错误控制、恢复过程：FinanceController 模块负责对费用结算、退款过程进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，确保财务数据的准确性和一致性。
   - 同步：FinanceController 模块与 GoodsReceiptController 模块、ReturnTyper 模块和 ClearController 模块等相互同步，以确保财务信息与货物收发、退货和清关过程的同步更新。
   - 状态、标识和其他报告特性：FinanceController 模块负责报告财务状态、标识费用结算和退款记录，并提供其他相关信息的报告特性。
5. MoniterController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-MONITER
   - 协议的优先级/层：MoniterController 模块作为系统监控控制器，其优先级/层较高，负责监控系统运行状态和异常事件的处理。
   - 分组：MoniterController 模块负责对监控数据进行分段与重组，以支持系统监控的操作和数据传输。
   - 合法性检查、错误控制、恢复过程：MoniterController 模块负责对监控数据进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，以确保系统监控的准确性和可靠性。
   - 同步：MoniterController 模块与各个模块和外部系统进行同步，以获取系统日

志、性能指标和异常事件的数据，并进行相应的处理和报告。

- 状态、标识和其他报告特性：MoniterController 模块负责报告系统运行状态、标识异常事件，并提供其他相关信息的报告特性。

6. UserController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-USER
   - 协议的优先级/层：UserController 模块作为用户管理控制器，其优先级/层较高，负责用户身份认证、权限管理和用户信息的处理。
   - 分组：UserController 模块负责对用户数据进行分段与重组，以支持用户管理的操作和数据传输。
   - 合法性检查、错误控制、恢复过程：UserController 模块负责对用户数据进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，以确保用户管理的准确性和安全性。
   - 同步：UserController 模块与其他模块进行同步，以获取用户信息并进行相应的权限控制和用户身份认证。
   - 状态、标识和其他报告特性：UserController 模块负责报告用户状态、标识用户身份和权限，并提供其他相关信息的报告特性。
7. GoodsReceiptController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-GOODSRECEIPT
   - 协议的优先级/层：GoodsReceiptController 模块作为货物收货控制器，其优先级/层较高，负责货物收货相关的业务逻辑的处理。
   - 分组：GoodsReceiptController 模块负责对货物数据进行分段与重组，以支持货物收货的操作和数据传输。
   - 合法性检查、错误控制、恢复过程：GoodsReceiptController 模块负责对货物数据进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，以确保货物收货的准确性和可靠性。
   - 同步：GoodsReceiptController 模块与其他模块进行同步，如与 InventoryController 模块进行库存更新的同步。
   - 状态、标识和其他报告特性：GoodsReceiptController 模块负责报告货物收货状态、标识货物信息，并提供其他相关信息的报告特性。
8. RouteController 模块接口实体：
   - 项目唯一标识符：CSCI-IF-ROUTE

- 协议的优先级/层：RouteController 模块作为路径规划控制器，其优先级/层较高，负责路径规划和导航相关的业务逻辑的处理。
- 分组：RouteController 模块负责对路径规划数据进行分段与重组，以支持路径规划和导航的操作和数据传输。
- 合法性检查、错误控制、恢复过程：RouteController 模块负责对路径规划数据进行合法性检查，处理错误情况，并提供适当的错误控制和恢复过程，以确保路径规划和导航的准确性和可靠性。
- 同步：RouteController 模块与其他模块进行同步，如与 MapController 模块进行地图数据的同步。
- 状态、标识和其他报告特性：RouteController 模块负责报告路径规划和导航状态、标识路径信息，并提供其他相关信息的报告特性。

g.其他特性，如接口实体的物理兼容性(尺寸、容限、负荷、电压、接插件的兼容性等)

接口实体的物理兼容性是指接口在物理层面的兼容性，包括尺寸、容限、负荷、电压、接插件的兼容性等方面。具体的特性可以根据系统需求和接口标准进行定义和规定。以下是一些目前涉及的物理兼容性特性：

1. 尺寸：接口实体的尺寸应符合规定的标准，以确保与其他设备或连接器的物理接口兼容。
2. 容限：接口实体应具备适当的容限，能够容忍一定范围内的物理变化，如温度变化、机械振动等，而不影响正常的数据传输和功能运作。
3. 负荷：接口实体应能够承受预期的负荷，包括电流、数据传输速率等方面的负荷要求，以确保稳定的信号传输和操作性能。
4. 电压：接口实体应支持规定的电压标准，以确保与其他设备或系统的电源兼容，同时能够提供足够的电压稳定性。
5. 接插件的兼容性：接口实体的接插件应符合规定的标准，以确保与其他设备或连接器的插拔兼容性，保证可靠的物理连接和接触。

在接口设计过程中，应参考相关的物理接口标准和规范，如电气标准、机械标准等，以确保接口实体在物理层面上的兼容性。此外，也要考虑系统的特殊需求和环境条件，例如工作温度范围、防水防尘要求等，来确定适当的物理兼容性特性。

5 CSCI 详细设计

5.1 物流计划管理 LPM（Logistic Plan Management）

a. 配置项设计决策和算法选择：在开发物流管理计划时，需要考虑物流运输的方式、物流路线以及货物分配等问题。可以根据具体业务场景选择不同的物流算法，如遗传算法、模拟退火算法等。

b. 软件配置项约束、限制或非常规特征：在物流管理计划中，可能会存在一些具体实现限制和需求，例如货物运输类型（如海运、陆运等）、地理位置限制、车辆限制等。

c. 编程语言选择及其理由：物流管理计划需要处理大量数据并且涉及复杂算法，通常使用的编程语言有 Python、Java、C++等，可根据开发人员技术特点和项目需求进行选择。

d. 过程式命令列表和解释的用户手册或其他文档的引用：物流管理计划一般需要调用许多外部数据源和接口，例如地图接口、车辆状态监控接口等。在开发过程中需要对这些过程进行统一规范化，并提供相应的命令列表以及 API 文档等参考文档。

e. 输入、输出和其他数据元素的说明，以及在局部数据和数据库中的描述：物流管理计划涉及到大量的数据处理，如车辆、司机、路线和货物等信息。需要在局部数据和数据库中提供详细的数据格式和描述，并且应严格保障数据隐私和安全性。

f. 逻辑描述：物流管理计划需要一个高效有序的逻辑结构来实现其功能，包括以下方面:

1. 路线规划逻辑：根据订单或者货物清单，生成最优运输方案，如综合考虑货物种类、体积、重量以及配送时间等因素，确保物品能够按时到达目的地。
2. 流程控制逻辑：针对一些特殊情况进行流程控制，例如交通堵塞、车辆故障、恶劣天气、异常品问题等，对物流监控系统进行整合，及时调度并进行协同决策。
3. 数据透明逻辑：对于已经完成的订单和货物，在物流管理平台上应该具有完整记录，以便用户进行查询，同时也可为公司内部监管和财务核算提供数据。

5.2 运输跟踪管理 TFUM（Transportation Follow-Up Management）

a. 配置项设计决策和算法选择：在开发运输跟踪管理模块时，需要考虑传感器技术、物联网技术等因素，选择合适的算法，如基于卡尔曼滤波、神经网络等实现位置信息

的精确计算和预测。

b. 软件配置项约束、限制或非常规特征：在运输过程中，由于许多因素的影响，可能存在一些非正常情况，如车辆被拦截、交通堵塞、货物丢失等。需要在软件中充分考虑这些限制和约束，并提供相应的异常处理机制。

c. 编程语言选择及其理由：运输跟踪管理模块需要获取分布于不同位置的大量数据，并且需要通过呈现客户端的方式向用户提供数据。因此，通常使用的编程语言有 Python、Java、C++、JavaScript 等可根据业务需求和技术选型等因素进行选择。

d. 过程式命令列表和解释的用户手册或其他文档的引用：在运输过程中，涉及到外部环境、订单变更以及货运损坏等复杂情况。因此，需要为运输跟踪管理模块提供完整的指令列表，方便用户快速进行相关操作，并正确地处理各种特殊情况。

e. 输入、输出和其他数据元素的说明，以及在局部数据和数据库中的描述：在运输跟踪管理模块中，包含设备位置信息、物流路线信息、货物数量、交通状况等多个数据元素，其中一些重要的数据需要进行加密保存，以满足信息安全的需求。

f. 逻辑描述：对于运输跟踪管理模块，需要一个高效有序的逻辑结构来实现其功能，包括以下方面：

1. 数据抓取逻辑：从传感器获取不断变化的车辆位置、温度、湿度等数据，构建多层次、多源头数据采集系统。

2. 轨迹分析与预测逻辑：根据数据统计和相关算法，在图表上展示运输跟踪情况，并针对路线异常进行预警提示。

3. 智能推荐机制逻辑：基于历史记录和物流路线数据，结合用户偏好和市场需求，开发智能推荐机制，实现货物寄送的快速匹配。

5.3 仓储管理 WMS (Warehouse Management System)

a. 配置项设计决策和算法选择：在设计仓储管理模块时，需要考虑多种因素，如库存情况、物品属性、仓库环境等，选择合适的仓储算法，如先进先出算法、最佳适配算法等。

b. 软件配置项约束、限制或非常规特征：在仓储过程中，仓库可能会受到温度、湿度等外部环境的影响，还可能存在人为操作失误等情况。因此，需要在软件中充分考虑这些限制和约束，并提供相应的异常处理机制。

c. 编程语言选择及其理由：仓储管理模块需要对大量数据进行分析和处理，并通过客户端界面展示给用户。因此，通常使用的编程语言有 Python、Java、C++、JavaScript 等，可根据业务需求和技术选型等因素进行选择。

d. 过程式命令列表和解释的用户手册或其他文档的引用：在仓储过程中，难免会出现商品移位、货物堆积、损坏等情况，因此需要为仓储管理模块提供完整的指令列表，方便用户快速进行相关操作，并正确地处理各种特殊情况。

e. 输入、输出和其他数据元素的说明，以及在局部数据和数据库中的描述：在仓储管理模块中，包含货物信息、库存情况、库位分配等多个数据元素，其中一些重要的数据需要进行加密保存，以满足信息安全的需求。

f. 逻辑描述：对于仓储管理模块，需要一个高效有序的逻辑结构来实现其功能，包括以下方面：

1. 库存管理逻辑：根据质量状况、过期时间等属性将货物合理分配到相应的库区，在库存达到临界值时，系统能够自动向供应商发送补货通知，确保仓库货物充足。

2. 订单物流配送逻辑：当用户下单后，自动将订单和库存相匹配，并通过物流信息查询，向用户提供准确到达时间和货物状态。

3. 数据统计和分析逻辑：根据历史出/入库记录，对货物库存变化情况进行数据挖掘和分析，为企业提供优化仓储管理和商品库存管理的有效决策。

5.4 运费结算管理 TFM (Transportation Financial Management)

a. 算法选择和配置项设计决策：在开发运费结算管理模块时，需要考虑多种因素如货物类型、运输距离、运输方式等，选择合适的算法如成本加成法、体积重量法、客户优惠政策等实现对运费的准确测算。

b. 数据安全和合规性保障：运费结算涉及大量的敏感数据如财务账目、客户信息以及税务信息等，因此需要采取严格的措施保证数据的隐私性和安全性，并遵守有关隐私保护和税务规定的法律法规。

c. 软件架构和接口设计：在运费结算管理中，需要涉及多个系统之间的数据交换和接口连接，因此需要建立完善的软件架构和接口机制，用于实现数据共享和传输，保证运费结算的准确和高效。

d. 用户手册和说明文档：运费结算过程相对较为复杂，需要提供详尽的用户手册和说明文档，使用户可以快速了解系统操作流程和各项功能特点，并掌握使用技巧和避免问题。

e. 输入、输出与数据存储：对于运费结算，需要进行多个数据字段的输入输出，如货物数量、重量、价格、送达时间等，各种数据之间的一致性与正确性是保证计费准确的重要依据。同时也需要考虑数据的存储和备份，以防意外损失或数据丢失造成不必

要的影响。

f. 系统逻辑描述：对于运费结算管理模块，需要一个高效有序的逻辑结构来实现其功能，包括以下方面：

1. 运费计算规则逻辑：负责自动处理全部客户订单，适时执行多变的价格接口及政策更改。

2. 应付账款管理逻辑：实现对应收账款、欠款情况的预警与管理，提高资金回笼率和现金流动情况。

3. 会计核算逻辑：把手工发票认证处理转变为电子化管理体系中，敏捷地完成财务报表生成等需求。

5.5 下单功能 OMS (Order Management System)

a. 配置项设计决策和算法选择： 在开发下单功能时，需要考虑许多配置项，如订单处理模式、支付方式等。针对不同的业务场景，可以选择不同的算法来实现下单功能，如商品组合优化算法用于折扣计算等。

b. 软件配置项约束、限制或非常规特征：在下单功能中，可能会涉及到一些具体实现限制和需求，例如仅可购买一定数量的商品、限制使用某些支付方式、支持跨境电商等。

c. 编程语言选择及其理由：考虑到下单功能一般与前后端框架集成，因此需要根据技术栈选定合适的编程语言，通常使用的编程语言有 Java、Python、JavaScript 等。

d. 过程式命令列表和解释的用户手册或其他文档的引用：下单功能需要调用数据库、支付接口等过程式命令，在实现此类功能时，需提供命令列表、相应文档以及相关示例代码供参考。

e. 输入、输出和其他数据元素的说明，以及在局部数据和数据库中的描述：下单功能涉及到大量的数据处理，包括订单信息、用户信息、商品信息等，需要在局部数据和数据库中提供详细公开详细的数据格式和描述，并统一规范化。

f. 逻辑描述：下单功能的实现需要一个高效有序的逻辑结构，包括以下方面：

1. 订单生成逻辑：在用户提交订单后，系统应该能够根据用户所选择的商品、数量和价格等信息自动生成一个唯一的、带有订单编号的订单。

2. 价格计算逻辑：根据商品价格、促销活动、配送费用和税费等因素计算出订单总价，并及时更新显示给用户。

3. 支付确认逻辑：当用户完成支付后，系统应该能够将支付金额及相关信息与订单信息进行比对，并在确认无误后将订单状态由“待支付”更新为“已支付”。

4. 异常处理逻辑：在下单功能的实现过程中，应当考虑各种异常情况的处理，如网络连接异常、支付失败、商品库存不足等，根据预先制定的处理方法进行相应的异常处理操作。

5. 订单管理逻辑：对于已经生成的订单，系统应该能够支持订单查询、修改、取消等功能，并且应保留相关订单信息，以便之后的业务需求查询。

6 需求的可追踪性

需求 ID	需求	描述	输入	处理	输出
R1	物流计划管理	提高物流运输的效率和准确性，降低物流成本。	物流需求、司机和运输工具的可用性、交通状况和路况	根据物流需求和实际情况生成最优化的运输计划，考虑货物的类型、大小和数量，运输距离，运输时间，以及运输成本等因素	运输计划：包括运输路线、物流成本、运输时间、车辆使用时间和司机安排等
R2	运输跟踪管理	实现物流系统对运输的管理，确保货物配送的准确、高效和便捷处理，以满足公司业务需求。	货物信息、运输计划	监控货物的实时位置和状态，实时更新货物运输的信息和进度，处理运输过程中的异常情况，例如货物损坏、交通堵塞、车辆故障等	实时的货物位置和状态，运输进度的更新，异常情况的处理结果
R3	仓储管	实现物流系统对	仓库信	管理货物的	入库、出库和库

	理	库存的管理，确保货物存储、出入库和盘点的准确、高效和便捷处理，以满足公司业务需求。	息、入库信息、出库信息	入库、出库和库存信息，安排货物的存储位置，考虑货物的类型、质量和运输方式等因素，实时更新货物的库存信息	存信息的管理和更新，库存状况的查询和统计
R4	运费结算管理	计算货物的运输费用，并管理运输费用的结算，确保物流运输的费用合理、准确、规范化，并提供结算明细，以支持企业的财务管理和运营决策。	运输信息、运输费用	计算货物的运输费用，考虑车辆使用费、司机工资、燃料费等因素，管理运输费用的结算，根据合同条款和发票信息进行支付或收款，并提供结算明细。	运输费用的计算和管理，运输费用的结算单据和明细
R5	下单功能	主要用于客户提交订单，包括货物信息、发货信息和客户信息等，以便系统能够根据这些信息计算出物流成本和预计送货时间，并将订单信息存储在系统中以备查询和跟	客户信息、货物信息、发货信息	检查客户信息和货物信息的完整性和正确性，根据货物信息和发货信息计算物流成本和预计送货时间，并将订单信息存储在系	确认订单信息，包括订单号、订单状态、物流成本、预计送货时间等

		踪。		统中以备查询和跟踪。	

7 注解

无特殊注解，相应解释都在对应处给出。

附录 A

附录可用来提供那些为便于文档维护而单独出版的信息(例如图表、分类数据)。为便于处理，附录可单独装订成册。附录应按字母顺序(A, B 等)编排。

学习报告 --- 吕策

学习任务 0：概述

本笔记系统学习了软件体系结构设计。

如文献 [1] 所述，“A critical issue in the design and construction of any complex software system is its architecture: that is, its gross organization as a collection of interacting components. ”，可见，软件体系结构研究的是软件系统的整体结构和各组件之间的联系，而不是具体的组件实现细节。

什么是软件体系结构设计的目标呢？文献 [1] 告诉我们，"A good architecture can help ensure that a system will satisfy key requirements in such areas as performance, reliability, portability, scalability, and interoperability. A bad architecture can be disastrous."。文献 [2] 告诉我们，软件体系结构设计的目的是为了实现软件系统的可修改性、性能、安全性、可靠性、健壮性、易使用性以及商业目标等质量属性。我们将对这些质量属性的需求作简要分析。

软件体系结构设计的方法有很多，文献 [2] 给出了一些常用的软件体系结构设计风格: 管道和过滤器、客户端-服务器、对等网络、发布-订阅、信息库、分层以及组合体系结构等。我们将关注于各种软件体系结构风格所适用的范围。

最后，我们还将简要介绍体系结构的评估与改进方法，如文献 [2] 中介绍的一些方法，包括测量设计质量、故障树分析、安全性分析、权衡分析、成本效益分析、原型化等方法。

本笔记在尽可能全面地介绍软件体系结构设计的同时，立足于软件体系结构设计的发展历程，将更为关注于其未来的发展方向。

学习内容 1：软件体系结构的定义与作用

如文献 [3] 所述，软件体系结构是与组件具体实现（包括算法和数据结构）完全不同的。

"As the size of software systems increases, the algorithms and data structures of computation no longer constitute major design problems. When systems are constructed from many components, the organization of the overall system—the software architecture—presents a new set of design problems. "

对软件体系结构的准确定义，能够同时指明其重要性和目标。如文献 [1] 中定义的：

"While there are numerous definitions of software architecture, at the core of all of them is the notion that the architecture of a system describes its gross structure. This structure illuminates the top level design decisions, including things such as how the system is composed of interacting parts, where are the main pathways of interaction, and what are the key properties of the parts. Additionally, an architectural description includes sufficient information to allow high-level analysis and critical appraisal. "

我们必须要指明组成系统的部件、部件间交互的方式，还需要给出分析和评估的方法。可见，软件体系结构充当着需求和具体代码的桥梁，起到了一种封装的作用。

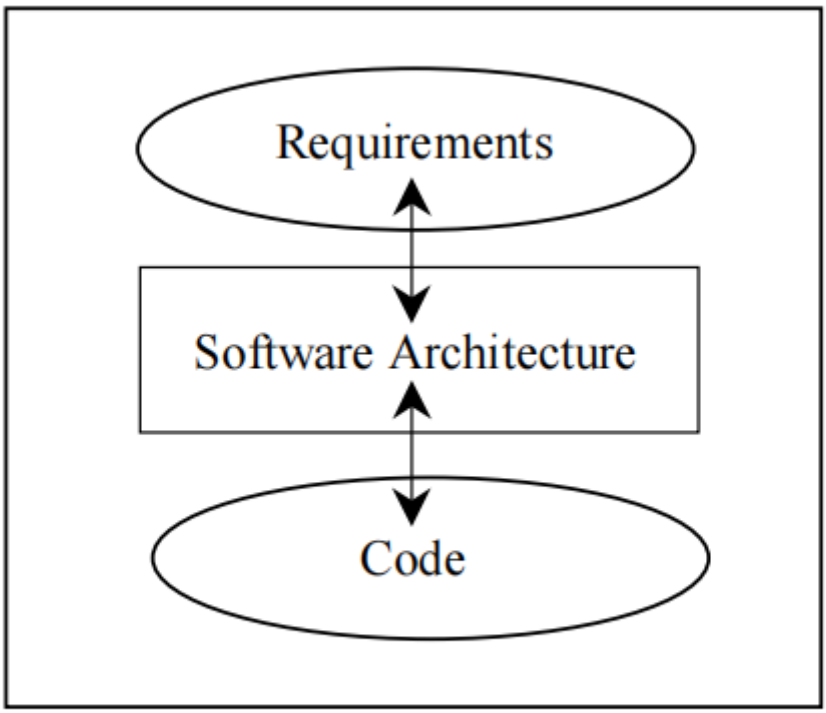


Figure 1: Software Architecture as a Bridge

为什么软件体系结构设计那么重要？我们知道，一件事物是重要的，大概率是因为它发挥了某些关键的作用。文献 [1] 指出，软件体系结构有以下重要的作用，包括软件系统理解与设计、组件重用和集成、组件组成与交互、改进计划与成本评估、更先进的软件分析和管理：

- *Understanding*: Software architecture simplifies our ability to comprehend large systems by presenting them at a level of abstraction at which a system's high-level design can be easily understood. Moreover, at its best, architectural description exposes the high-level constraints on system design, as well as the rationale for making specific architectural choices.
- *Reuse*: Architectural descriptions support reuse at multiple levels. Current work on reuse generally focuses on component libraries. Architectural design supports, in addition, both reuse of large components and also frameworks into which components can be integrated. Existing work on domain-specific software architectures, reference frameworks, and architectural design patterns has already begun to provide evidence for this.
- *Construction*: An architectural description provides a partial blueprint for development by indicating the major components and dependencies between them. For example, a layered view of an architecture typically documents abstraction boundaries between parts of a system's implementation, clearly identifying the major internal system interfaces, and constraining what parts of a system may rely on services provided by other parts.
- *Evolution*: Software architecture can expose the dimensions along which a system is expected to evolve. By making explicit the "load-bearing walls" of a system, system maintainers can better understand the ramifications of changes, and thereby more accurately estimate costs of modifications. More over, architectural descriptions separate concerns about the functionality of a component from the ways in which that component is connected to (interacts with) other components, by clearly distinguishing between components and mechanisms that allow them to interact. This separation permits one to more easily change connection mechanisms to handle evolving concerns about performance interoperability, prototyping, and reuse.
- *Analysis*: Architectural descriptions provide new opportunities for analysis, including system consistency checking, conformance to constraints imposed by an architectural style, conformance to quality attributes, dependence analysis, and domain-specific analyses for architectures built in specific styles.
- *Management*: Experience has shown that successful projects view achievement of viable software architecture as a key milestone in an industrial software development process. Critical evaluation of architecture typically leads to a much clearer understanding of requirements, implementation strategies, and potential risks.

文献 [3] 也指出软件体系结构有如下重要作用：

- Inhibiting or Enabling a System's Quality Attributes
- Reasoning About and Managing Change
- Predicting System Qualities
- Enhancing Communication among Stakeholders
- Carrying Early Design Decisions
- Defining Constraints on an Implementation

- Influencing the Organizational Structure
- Enabling Evolutionary Prototyping
- Improving Cost and Schedule Estimates
- Supplying a Transferable, Reusable Model
- Allowing Incorporation of Independently Developed Components
- Restricting the Vocabulary of Design Alternatives
- Providing a Basis for Training

学习内容 2：软件体系结构设计的目标 ---- 质量属性

在概述中，我们提到：软件体系结构设计的目标，就是提高（指定的）质量属性。

首先，我们要搞懂什么是质量属性（Quality Attributes）。如文献 [4] 中定义的：

"A quality attribute (QA) is a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders. You can think of a quality attribute as measuring the "goodness" of a product along some dimension of interest to a stakeholder. "。

这一定义表明了质量属性是系统“好”的一种度量。

那么，有哪些比较典型的质量属性呢？我们依据文献 [2] 来抛砖引玉地列举几个：

- 可修改性：在对系统做出一个特定改变的情况下，软件单元可以被分为受直接影响和受间接影响的软件单元。受直接影响的软件单元需要为了适应系统改变而改变自身职责；受间接影响的软件单元只需要修改其实现以适应受直接影响的软件单元的影响，而不需要修改自身职责。对于上述两种受到影响的软件单元，我们都要减少其数量以保证可修改性，做法包括公开接口、降低耦合性等。

- 性能：性能描述了系统的速度和容量，包括响应时间、吞吐量、负载等。提高系统性能要求提高资源利用率、有效管理资源分配、降低对资源的需求等。

- 安全性：大多数的安全性需求阐述了什么是需要被保护的，以及谁不可以访问它。体系结构的安全性有两个方面：高免疫力，指系统能够阻挡攻击企图；高弹性，指系统能够快速容易地从成功的攻击中恢复。体系结构可以通过功能分段、减少安全性弱点等方法提高免疫力。

- 可靠性与健壮性：如果一个软件系统可以在假设的环境下正确地实现所要求的功能，那么我们就说这个软件系统是可靠的；如果在不正确或意外的环境下仍然能够正确地工作，那么它就是健壮的，也即——可靠性与软件本身内部是否有错误有关，健壮性与软件容忍错误或外部环境异常时的表现有关。为了提高可靠性，我们需要实现主动故障检测、异常处理以及故障恢复；为了提高健壮性，我们可以采用“互相怀疑”的设计策略。

- 易使用性：易使用性反应了用户能够操作系统的容易程度。
- 商业目标：商业目标包括开发成本最小化、维护成本最小化、产品上市时间最小化等。

在物流管理系统项目中，我主要负责前端的软件体系结构设计和实现，这是一个至关重要的角色，负责定义前端应用程序的整体结构、组件划分和交互逻辑。通过参与这个部分的开发，我获得了许多宝贵的学习和成长机会。

首先，我学到了软件体系结构的基本概念和原则。在前端开发中，良好的软件体系结构可以提高代码的可维护性、可扩展性和可重用性。我学习了各种常见的前端体系结构模式，如 MVC（Model-View-Controller）、MVVM（Model-View-ViewModel）和 Flux 等，以及它们的特点和适用场景。通过合理选择和应用这些模式，我能够更好地组织和管理前端代码。

其次，我学到了组件化开发的重要性。在复杂的前端应用中，组件化开发可以将界面和功能划分为独立的模块，提高代码的可复用性和可测试性。我学习了如何设计和构建可复用的组件，以及如何通过组件之间的通信和状态管理实现复杂的交互逻辑。通过使用前端框架（如 React、Angular 或 Vue.js），我能够更高效地创建和管理组件，并实现良好的组件复用和维护性。

此外，我学到了前端性能优化的技巧和策略。在开发过程中，优化前端应用的性能是至关重要的，可以提高用户体验并减少加载时间。我学习了如何进行代码压缩和合并、资源缓存和懒加载、图片优化以及网络请求的优化等技术手段。通过应用这些优化策略，我能够提升前端应用的性能和响应速度。

最后，通过参与前端软件体系结构的设计和实现，我培养了团队合作和沟通的能力。在项目中，我与后端开发人员、设计师和产品经理密切合作，理解他们的需求并将其转化为具体的前端实现。我学会了与团队成员协调工作，共同解决问题，并及时进行沟通和反馈。

以下是我在前端开发中所学到的一些关键点：

- 通过设计和实现前端软件体系结构，我深入理解了前端开发的整体流程和架构思维。我学会了将复杂的需求拆解为独立的组件和模块，设计清晰的数据流和交互逻辑，以及构建可维护和可扩展的前端应用。这使我能够更加高效地开发和维护复杂的前端项目，并在团队中分享我的经验和最佳实践。
- 参与前端软件体系结构的设计让我更加注重代码的可重用性和可测试性。通过将前端应用划分为独立的组件，我能够更好地复用已有的代码，减少冗余和重复的工作。同时，我学会了编写可测试的前端代码，通过单元测试和集成测试确保代码的质量和稳定性。这使我在开发过程中更加注重代码的可维护性，以及减少 bug 的产生和修复的工作量。
- 通过与其他团队成员的密切合作，我学会了良好的沟通和协作技巧。在前端软件

体系结构的设计中，我需要与后端开发人员、UI/UX 设计师和产品经理进行频繁的沟通和协商。我学会了倾听和理解他们的需求，解决潜在的冲突和问题，并提供可行的解决方案。这使我能够更好地与团队合作，推动项目的顺利进行，并达到预期的目标。

- 参与前端软件体系结构的学习和实践，不仅提升了我的技术能力，还让我对前端开发有了更深入的理解。我能够更加全面地思考和评估不同的前端技术选型和框架，选择最适合项目需求的解决方案。这让我在面对新的前端挑战时更加自信和有条理，能够快速学习和适应新的技术和工具。

综上所述，参与前端软件体系结构的设计和实现让我受益匪浅。我不仅掌握了前端开发的核心原理和技术，还培养了团队合作、沟通和解决问题的能力。这些学习和成长的经验将对我的职业发展产生积极的影响，并使我能够在前端开发领域不断进步和追求卓越。

学习报告 —— 李奕辰

学习内容 1：软件体系结构的风格

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。词汇表中包含一些构件和连接件组合起来的。体系结构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个模块和子系统有效地组织成完整的系统。按这种方式理解，软件体系结构风格定义了用于描述系统的术语表和一组指导构件系统的规则。

构件的定义：构件是具有某种功能的可重用的软件模板单元，表示了系统中主要的计算元素和数据存储。构件有两种：复合构件和原子构件，复合构件由其他复合构件和原子构件通过连接而成；原子构件是不可再分的构件，底层由实现该构件的类组成，这种构件的划分提供了体系结构的分层表示能力，有助于简化体系结构的设计。

连接件的定义：连接件表示了构件之间的交互，简单的连接件如管道、过程调用、事件广播等，更为复杂的交互如客户-服务器通信协议，数据库和应用之间的 SQL 连接等。

软件体系结构风格的四要素：（1）提供一个词汇表；

（2）定义一套配置规则；

（3）定义一套语义解释规则；

（4）定义对基于这种风格的系统所进行的分析。

软件体系结构风格的目的:软件体系结构风格为大粒度的软件重用提供了可能。

几种软件体系结构风格的分类：

（1）管道与过滤器：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，所以在输入被完成消费之前，输出便产生了。因此，这里的构件被称为过滤器，这种风格的连接件就像是数据流传输的管道，将一个过滤器的输出传到另一个过滤器的输入。此风格特别重要的过滤器必须是独立的实体，它不能与其他的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。一个管道与过滤器网格输出的正确性并不依赖于过滤器进行增量计算过程的顺序。

管道与过滤器风格的特点：

优点：使得构件具有良好的隐蔽性和高内聚、低耦合的特点；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；支持软件重用；系统维护和增强系统性能简单；允许对一些如吞吐量、死锁等属性的分析；支持并行执行。

缺点：通常导致进程成为批处理结构；不适合处理交互的应用；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）客户/服务器：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，是 20 世纪 90 年代成熟起来的技术，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，其任务主要集中于：数据库安全性的要求；数据库访问并发性的控制；数据库前端的客户应用程序的全局数据完整性规则；数据库的备份与恢复。

客户应用程序的主要任务是：提供用户与数据库交互的界面；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；利用客户应用程序对存在于客户端的数据执行逻辑要求。

客户/服务器体系风格的特点：

优点：C/S 体系结构的优点主要在于系统的客户应用程序和服务器构件分别运行在不同的计算机上，系统中每台服务器都可以适合各构件的要求，这对于硬件和软件的变化显示出极大的适应性和灵活性，而且易于对系统进行扩充和缩小。在 C/S 体系结构中，系统中的功能构件充分分离，客户应用程序的开发集中于数据的显示和分析，而数据库服务器的开发则集中于数据的管理，不必在每一个新的应用程序中都要对一个 DBMS 进行编码。将大的应用处理任务分布到许多通过网络连接低成本计算机上，以节约大量费用；C/S 体系结构具有强大的数据操作和事务处理能力，模型思想简单，易于人们理解和接受。

缺点：开发成本较高；客户端程序设计复杂；信息内容和形式单一，因为传统应用一般为事务处理，界面基本遵循数据库的字段解释，开发之初就已确定，而且不能随时截取办公信息和档案等外部信息，用户获得的只是单纯的字符和数字，既枯燥又死

板；用户界面风格不一，使用复杂，不利于推广使用；软件移植困难；软件维护和升级困难；新技术不能轻易应用。

（3）浏览/服务器风格：B/S 体系结构主要是利用不成熟的 WWW 浏览器技术。结合浏览器的多种脚本语言，用通过浏览器就实现了原来需要复杂的专用软件才能实现的强大功能，并节约了开发成本。从某种程度上来说，B/S 结构是一种全新的软件体系结构。

C/S 体系结构的风格特点：

优点：B/S 结构的”零客户端“方式，使组织的供应商和客户的计算机方便地成为管理信息系统的客户端，进而在限定的功能范围内查询组织相关信息，完成与组织的各种业务往来的数据交换和处理工作，扩大了组织计算机应用系统的功能覆盖范围，可以更加充分利用网络上的各种资源，同时应用程序维护的工作量也大大减少。

缺点：B/S 体系结构缺乏对动态页面的支持能力，没有集成有效的数据库处理功能；B/S 体系结构的系统扩展能力差，安全性难以控制；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；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。

学习内容 2：软件体系结构设计的方法

在介绍具体的软件体系结构设计方法之前，我们首先给出文献 [1] 中的一段描述：

"Within industry, two trends highlighted the importance of architecture.

The first was the recognition of a shared repertoire of methods, techniques, patterns and idioms for structuring complex software systems. For example, the box-and-line-diagrams and explanatory prose that typically accompany a high-level system description often refer to such organizations as a "pipeline," a "blackboard-oriented design," or a "client-server system." Although these terms were rarely assigned precise definitions, they permitted designers to describe complex systems using abstractions that make the overall system intelligible. Moreover, they provided significant semantic content about the kinds of properties of concern, the expected paths of evolution, the overall computational paradigm, and the relationship be tween this system and other similar systems.

The second trend was the concern with exploiting com monalities in specific domains to provide reusable frame works for product families. Such exploitation is based on the idea that common aspects of a collection of related systems can be extracted so that each new system can be built at relatively low cost by "instantiating" the shared design. Familiar examples include the standard decomposition of a compiler (which permits undergraduates to construct a new compiler in a semester), standardized communication protocols (which allow vendors to interoperate by providing services at different layers of abstraction), fourth generation languages (which exploit the common patterns of business information processing), and user interface toolkits and

frameworks (which provide both a reusable framework for developing interfaces and sets of reusable components, such as menus and dialogue boxes)."

可见，软件体系结构设计方法需要着重于两点：一是对构建复杂软件系统的共同方法、技术、模式和范式的认识；二是注开发特定领域的共性，为产品族提供可重用的框架。因此，我们给出的软件体系结构设计要有足够高的抽象性和复用性，又要有足够的针对性。

我们将基于文献 [2] 和 [3] 简略列举几种常用的软件体系结构：

Pipes and Filters

In a pipe and filter style each component has a set of inputs and a set of outputs. A component reads streams of data on its inputs and produces streams of data on its outputs, delivering a complete instance of the result in a standard order. This is usually accomplished by applying a local transformation to the input streams and computing incrementally so output begins before input is consumed. Hence components are termed "filters". The connectors of this style serve as conduits for the streams, transmitting outputs of one filter to inputs of another. Hence the connectors are termed "pipes".

使用管道和过滤器的典型软件系统是编译器，各个过滤器（如词法分析器、语法分析器等）会接收上一个过滤器的输出，输出直接传送到到下一个过滤器，过滤器间彼此独立，并具有严格的输入输出格式。

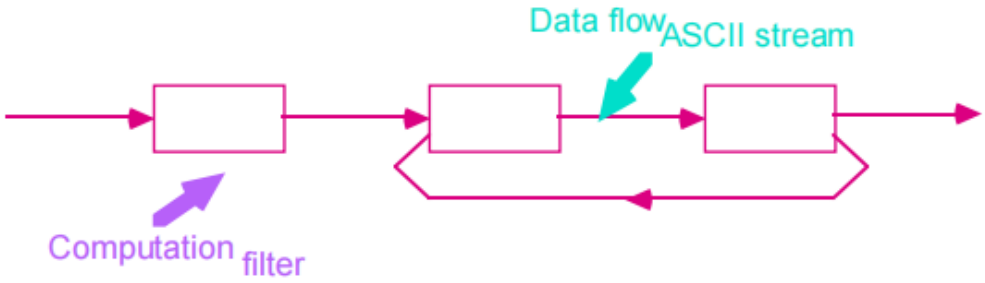


Figure 1: Pipes and Filters

- **Data Abstraction and Object-Oriented Organization**

In this style data representations and their associated primitive operations are encapsulated in an abstract data type or object. The components of this style are the objects—or, if you will, instances of the abstract data types. Objects are examples of a sort of component we call a manager because it is responsible for preserving the integrity of a resource (here the representation). Objects interact through function and procedure invocations. Two important aspects of this style are (a) that an object is responsible for preserving the integrity of its representation (usually by maintaining some invariant over it), and (b) that the representation is hidden from other objects.

数据抽象和面向对象是当前软件设计最主流的范式，其对于对象细节的隐藏、对于数据的保护等特性对软件体系结构设计的重要性不言而喻。

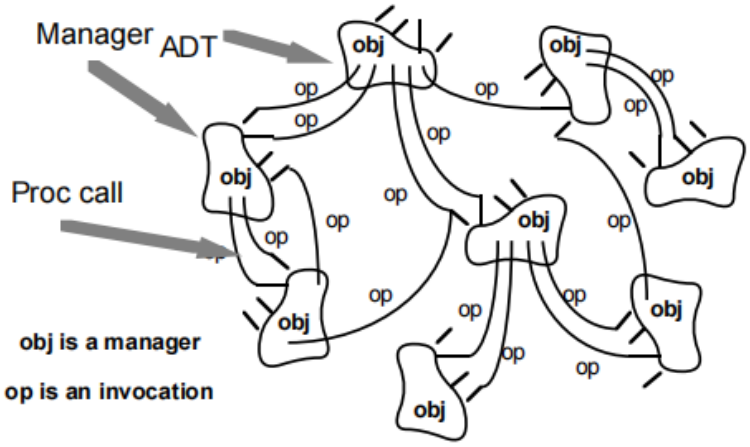


Figure 2: Abstract Data Types and Objects

- **Event-based, Implicit Invocation**

The idea behind implicit invocation is that instead of invoking a procedure directly, a component can announce (or broadcast) one or more events. Other components in the system can register an interest in an event by associating a procedure with the event. When the event is announced the system itself invokes all of the procedures that have been registered for the event. Thus an event announcement ``implicitly'' causes the invocation of procedures in other modules.

很多系统都使用了隐式调用机制，例如在数据库管理系统中用于确保一致性约束，通过语法制导编辑器来支持增量语义检查等。隐式调用的一个重要好处是它为重用提供了强大的支持。只要将任何组件注册为该系统的事件，就可以将其引入到系统中。第二个好处是隐式调用简化了系统进化，在不影响系统其他部件接口的情况下，部件可以被其他部件替换。

- **Layered Systems**

A layered system is organized hierarchically, each layer providing service to the layer above it and serving as a client to the layer below. In some layered systems inner layers are hidden from all except the adjacent outer layer, except for certain functions carefully selected for export. Thus in these systems the components implement a virtual machine at some layer in the hierarchy. (In other layered systems the layers may be only partially opaque.) The connectors are defined by the protocols that determine how the layers will interact. Topological constraints include limiting interactions to adjacent layers.

典型的分层系统包括：计算机网络协议、早年的层次化操作系统等。

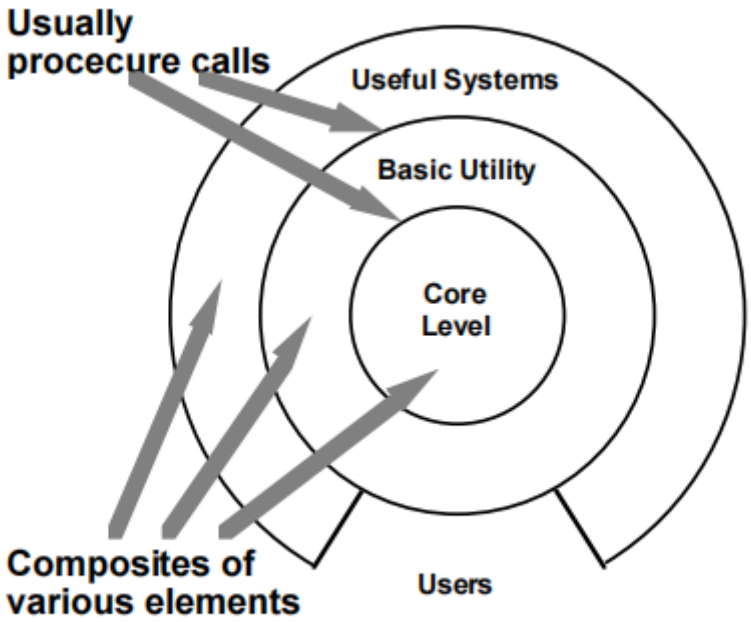


Figure 3: Layered Systems

- **Repositories**

In a repository style there are two quite distinct kinds of components: a central data structure represents the current state, and a collection of independent components operate on the central data store.

The knowledge sources: separate, independent parcels of application-dependent knowledge. Interaction among knowledge sources takes place solely through the blackboard.

The blackboard data structure: problem-solving state data, organized into an application-dependent hierarchy. Knowledge sources make changes to the blackboard that lead incrementally to a solution to the problem.

Control: driven entirely by state of blackboard. Knowledge sources respond opportunistically when changes in the blackboard make them applicable.

信息库系统基于共享的数据，例如编译器之于符号表等即是典例。

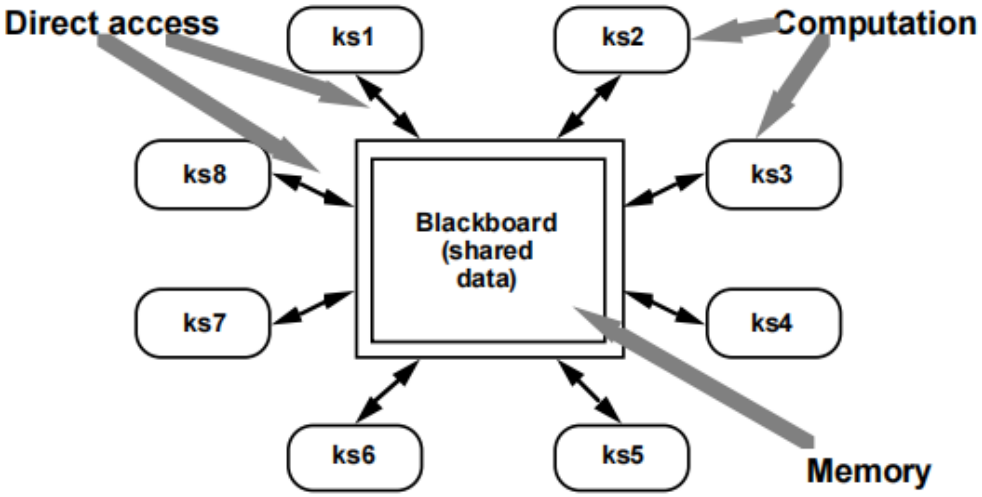


Figure 4: The Blackboard

- **Table Driven Interpreters**

In an interpreter organization a virtual machine is produced in software. An interpreter includes the pseudo-program being interpreted and the interpretation engine itself. The pseudo-program includes the program itself and the interpreter's analog of its execution state (activation record). The interpretation engine includes both the definition of the interpreter and the current state of its execution. Thus an interpreter generally has four components: an interpretation engine to do the work, a memory that contains the pseudo-code to be interpreted, a representation of the control state of the interpretation engine, and a representation of the current state of the program being simulated.

显然，解释器架构很适合虚拟机的构建。

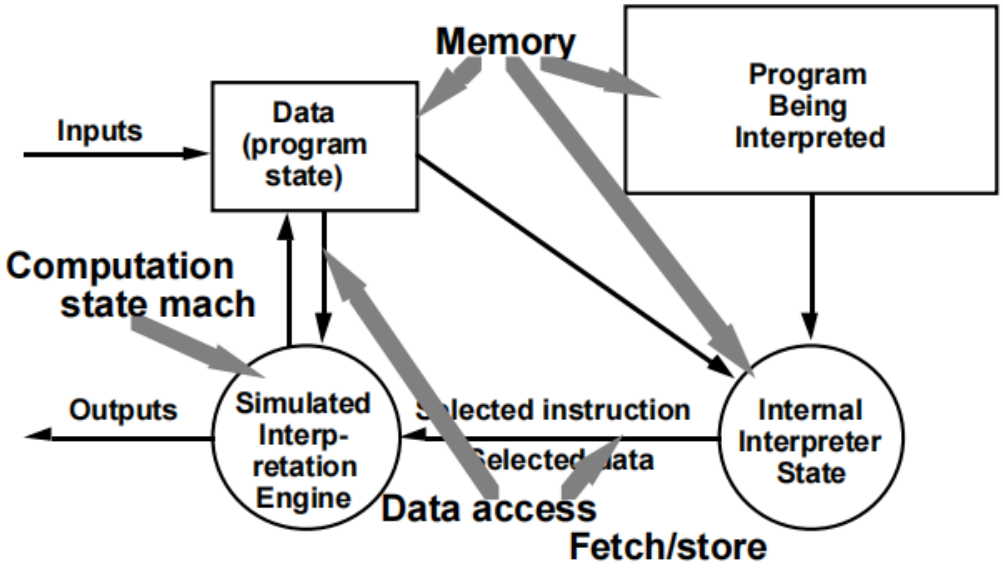


Figure 5: Interpreter

- **C/S 架构和 P2P 架构**

文献 [3] 中缺少了一些十分重要的体系结构——C/S 架构和 P2P 架构。我们知道，C/S 架构（客户-服务器）是基于请求/应答协议的，是最主要的服务器架构；而 P2P 架构则文件共享网络中广泛应用。两者在计算机网络中有非常广泛且重要的应用。

实践与收获：后端开发

在物流管理系统项目中，我承担了后端开发的角色，参与了整个软件体系结构的设计和实现。通过这个过程，我获得了宝贵的学习经验，并提升了在后端开发和软件体系结构方面的能力。

- 学到了软件体系结构的设计原则和方法。在项目开始之前，我进行了详细的需求分析，并与团队成员讨论确定了系统的功能和架构要求。我学习了如何将系统划分为不同的模块和层次，以及如何定义模块之间的接口和交互方式。我还学习了常见的软件体系结构模式，如 MVC（模型-视图-控制器）和微服务架构，以选择适合项目需求的架构风格。
- 掌握了后端开发的核心技术和工具。在项目中，我使用了一种后端开发语言（如 Java、Python 或 Node.js）来编写服务器端代码，实现系统的业务逻辑和数据处理。我学习了如何使用框架和库来简化开发过程，并提高代码的可维护性和扩展性。我还学习了如何设计和管理数据库，以及如何与前端和数据库团队进行协作，确保数据的准确传输和存储。
- 了解了系统的性能优化和安全保护。在开发过程中，我学习了如何优化后端代码和数据库查询，以提高系统的响应速度和并发处理能力。我还学习了如何实施安全策略和防护措施，保护系统免受恶意攻击和数据泄露的风险。通过学习这些知识，我能够为系统提供高效、安全和可靠的后端服务。
- 通过参与软件体系结构的设计和实现，我培养了团队合作和沟通的能力。在项目中，我与前端开发人员和数据库团队紧密合作，理解他们的需求并提供有效的解决方案。我学会了与团队成员协调工作，共同解决问题，并及时进行沟通和反馈。

综上所述，通过参与后端开发和软件体系结构的设计，我不仅掌握了后端开发的核心原理和技术，还培养了团队合作和沟通的能力。这些学习和成长的经验将对我的职业发展产生积极的影响，并使我能够在后端开发和软件体系结构领域不断提升自己。

学习报告 —— 方羿阳

学习背景和目的

在这次学习中，我学习了软件架构设计和管理的基本概念和原则，旨在更好地理解软件开发过程中的架构设计和管理的重要性，并能够将这些知识应用于实际开发项目中。

学习内容

1. 在学习中，我了解了以下内容：

软件架构的定义和作用：软件架构是指软件系统中各个组件之间的关系和交互，是整个系统的骨架和基础。良好的软件架构能够提高系统的可维护性、可扩展性和可重用性。

软件架构设计的原则和方法：软件架构设计需要遵循一些基本原则，如分层、模块化、解耦等。同时，也有一些方法可以用来辅助架构设计，如面向对象设计、面向服务设计、微服务架构等。

软件架构管理的重要性：软件架构不是一次性的工作，而是一个持续演化的过程。在软件开发过程中，架构设计和管理需要与其他活动（如需求分析、测试等）紧密结合，以保证软件系统的质量和可靠性。

软件架构设计和管理工具：为了更好地支持软件架构设计和管理，有一些工具可以用来辅助，如 UML 建模工具、架构描述语言、架构评估工具等。

软件架构的类型：软件架构可以分为多种类型，如客户端-服务器架构、分层架构、发布-订阅架构、事件驱动架构等。不同类型的架构适用于不同的应用场景，需要根据实际需求进行选择和设计。

软件架构的风格：软件架构还可以按照不同的风格进行分类，如面向对象风格、面向服务风格、面向数据流风格等。不同的架构风格具有不同的特点和优缺点，需要根据实际需求进行选择和设计。

软件架构的演化：软件架构不是一成不变的，它会随着软件系统的演化和需求变化而变化。在架构演化过程中，需要考虑兼容性、可维护性、可扩展性等方面的问题，并且需要在演化过程中保持系统的稳定性和一致性。

软件架构评估：软件架构设计的好坏直接关系到软件系统的质量和可靠性，因此需要进行评估和验证。常见的软件架构评估方法包括场景模拟、模型检测、性能测试等。

软件架构的管理：软件架构管理是指对软件架构进行规划、设计、实施和监督的过程。软件架构管理需要考虑到团队协作、项目管理、风险管理等方面的问题，以确保软件架构的有效性和持续性。

下面是关于软件系统结构设计的评估与改进的详细内容：

故障树

软件系统结构的评估与改进，首要的对故障的分析和处理。一种有利的工具是故障树。故障树的根节点是我们想分析的失效，而其他节点表示事件或导致根节点失效发生的故障，边是节点间的逻辑关系（表示为逻辑门），表示父节点的事件发生是由子节点的

事件的某一发生情况导致的。

一旦建立了故障树，我们就可以查找设计中的弱点。特别地，我们可以将故障树转化为割集树，它解释了哪些事件的组合可以引起失效，以简化对故障树的分析。在割集树中，每个节点表示一个割集，即一组事件的组合。割集树可以从故障树的根节点逻辑门开始，根据逻辑门不断分裂或合并，直到所有子节点都是基本事件为止。

通过故障树/割集树，我们可以找到导致失效的事件组合，然后对这些事件组合进行分析，找到导致失效的原因，从而改进软件系统结构。为此，我们可以改正故障、预防故障、增加检测故障或故障恢复。

安全性分析

安全性分析可以被描述为六个步骤：

1. 软件特征化
2. 威胁分析
3. 漏洞评估
4. 风险可能性决策
5. 风险影响决策
6. 风险缓解计划

除此以外，软件系统结构设计的评估与改进还包括：

- 权衡分析：根据质量属性比较多个设计候选项，利用比较表等工具。
- 成本效益分析：根据经济效益和经济成本来综合分析系统的成本效益，一般包括：
- 计算效益，即改进某个质量属性所带来的增值；计算投资回报，即 ROI = 效益/成本；计算投资回收期等。

学习收获

通过这次学习，我更深入地了解了软件架构设计和管理的基本概念和原则，能够更好地理解软件开发过程中的架构设计和管理的重要性，并能够将这些知识应用于实际开发项目中。同时，我也了解到一些支持软件架构设计和管理的工具，可以在实践中更好地支持软件架构的演化和管理。

另外，这次学习还让我意识到了以下几点：

1. 软件架构设计和管理需要跨越整个软件开发过程，从需求分析到发布维护都需要考虑架构设计和管理，而不是只关注开发阶段。

2. 软件架构的演化是一个渐进的过程，需要在不断实践和反思中进行调整和优化，而不是一次性地完成。

3. 软件架构设计和管理不仅仅是技术问题，还涉及到组织管理、团队合作等方面的问题，需要全局思考和协调。

4. 软件架构设计和管理需要不断地学习和更新知识，以跟上技术的发展和变化，避免陷入过时的设计和管理模式中。

综上所述，这次学习让我获得了丰富的软件架构设计和管理的知识，提高了我的技能水平，也让我意识到了软件开发中架构设计和管理的重要性，我将会在实践中不断地运用这些知识和技能，不断完善和优化软件系统的架构设计和管理。

学习报告 —— 徐伟

学习内容 1 ： 软件体系结构风格专题

软件体系结构是将系统分解为多个部分，并确定它们之间如何交互和通信的过程。在软件开发中，选择正确的体系结构风格对于保证项目质量、按时交付以及面向未来具有重要意义。本文分享了笔者在学习经典软件体系结构风格方面所获得的知识。

一、什么是软件体系结构

在讨论软件体系结构风格时，了解软件体系结构的概念和重要性是必要的。软件体系结构可以看作是软件中不变的主干或骨架，承载和支撑着软件的功能和特性。良好设计的软件体系结构可以增加可维护性、扩展性和性能。同时，体系结构决定了软件的演进路线，因此选择正确的体系结构风格非常关键。

二、经典的软件体系结构风格

1、层次体系结构模式

层次模式是最传统也是最常见的体系结构之一。该模式包括一个层次结构，每个层级都提供一组相关的服务，这些服务通过接口暴露给顶层。底层实现低级别操作，提供上层所需的基础服务。该模式易于理解和维护，并且在更改或升级系统时也很方便。

2、客户端-服务器体系结构

客户端-服务器体系结构将应用程序分解为前端和后端两个主要部分。客户端负责呈现用户界面并与用户交互，而服务器负责完成应用程序逻辑、数据处理和存储任务。该体系结构可以提高系统响应时间和应用程序的总成本，并且部署环境非常灵活。

3、管道和过滤器（P/F）模式

管道和过滤器（P/F）模式是流处理和实时数据处理中优选的模型。它包含数个过

滤器和连接它们的管道。输入数据通过管道向过滤器流动，在经过过滤器处理后由管道输出结果。该模式具有可扩展性、保持协议中自描述性等特点。

4、预订阅模式

预订阅模式是一种异步通信模式，通过从事件总线预订感兴趣的主题，当该主题上出现新消息时，系统会自动触发警告机制。该模式可以提高分布式系统的可伸缩性和灵活性，对于支持高吞吐量的应用程序尤其有帮助。

三、总结与展望

随着业务需求和技术的发展，软件体系结构的重要性日益凸显。当我们在进行软件开发时，选择一个正确的体系结构风格对于保证项目的质量和按时交付是至关重要的。而了解和掌握多种经典体系结构风格，则可以让我们更好地作出决策和实践。

在学习软件开发过程中，我们会接触到很多新的概念和理论，比如软件体系结构设计。如果不了解软件体系结构是什么以及如何进行选择和设计，那么可能会陷入困境。但是，通过学习一些常见的体系结构风格，我们将更好地理解这个知识点。总结一下，层次体系结构模式是一个非常流行且易于理解的体系结构风格，它将系统分解成几个独立的部分，便于维护、扩展和升级系统；客户端-服务器模式则将应用程序分为前端和后端两个主要部分，这种模式适合处理对实时性和负载均衡有较高要求的场景；对于一些需要流处理或实时数据处理的场景，我们可以考虑采用管道和过滤器（P/F）模式，该模式具有高度可扩展性和保持协议自描述的特点；而预订阅模式适用于异步通信场景，用户可以通过从事件总线中预订感兴趣的主题来接收消息，可以提高分布式系统的可伸缩性和灵活性。

当我们选择正确的体系结构风格时，需要考虑到项目需求、技术能力和团队规模等多方面问题，并且在实践过程中不断优化和改进体系结构设计。了解并选择适合自己的体系结构风格可以使软件开发更加高效、合理，具有更好的可维护性和可扩展性。对于初学者来说，前期可以先了解一些基本的体系结构风格，并在实际应用中逐步学习和掌握更多有用的知识点，这样才能更好地运用体系结构设计，提高软件开发效率。了解并选择正确的体系结构风格可以使我们的软件开发更加合理、高效，具有更好的可维护性和可扩展性。但是，在实际应用过程中，我们需要不断地根据实际项目需求和技术进展不断优化和改进体系结构设计。同时，学习和掌握多种软件体系结构风格也是提高编程水平的重要一步。

学习内容 2 ：软件系统结构设计的今天

文献 [1] 中准确地描述了现代软件系统结构设计的发展表现在三个方面：架构描述语言和工具、产品线工程和架构标准的出现，以及架构设计专业知识的编码和传播。

In addition, the technological basis for architectural design has improved dramatically. Three of the important advancements have been the development of architecture description languages and tools, the emergence of product line engineering and

architectural standards, and the codification and dissemination of architectural design expertise.

1.the development of architecture description languages and tools

大多数架构设计的盒式图示通常是 informal 的，这会带来很多问题。例如，设计意义不明确、无法进行正式的一致性、完整性或正确性分析、初始设计中假定的架构约束无法随着系统演变而得到执行，以及缺乏支持架构设计人员完成任务的工具等。因此，业界和学术界的许多研究人员建议使用形式化符号表示和分析架构设计，使用 Architecture Description Languages（ADL）可以解决上述问题。ADL 不仅提供软件架构的概念框架，还提供具体的语法描述，并提供用于解析、显示、编译、分析或模拟架构描述的工具。

体系结构描述语言（ADLs）为描述软件体系结构提供了概念框架和具体语法，例如 Adage, Aesop, C2, Darwin, Rapide, SADL, UniCon, Meta-H, and Wright.

Languages explicitly designed with software architecture in mind are not the only approach. There has been considerable interest in using general-purpose object design notations for architectural modeling [8, 24]. Moreover, recently there have been a number of proposals that attempt to show how the concepts found in ADLs can be mapped directly into an object-oriented notation like UML.

现在也有很多工作，将 ADL 中的软件结构直接映射到 UML 这样的更一般的面向对象建模语言中。

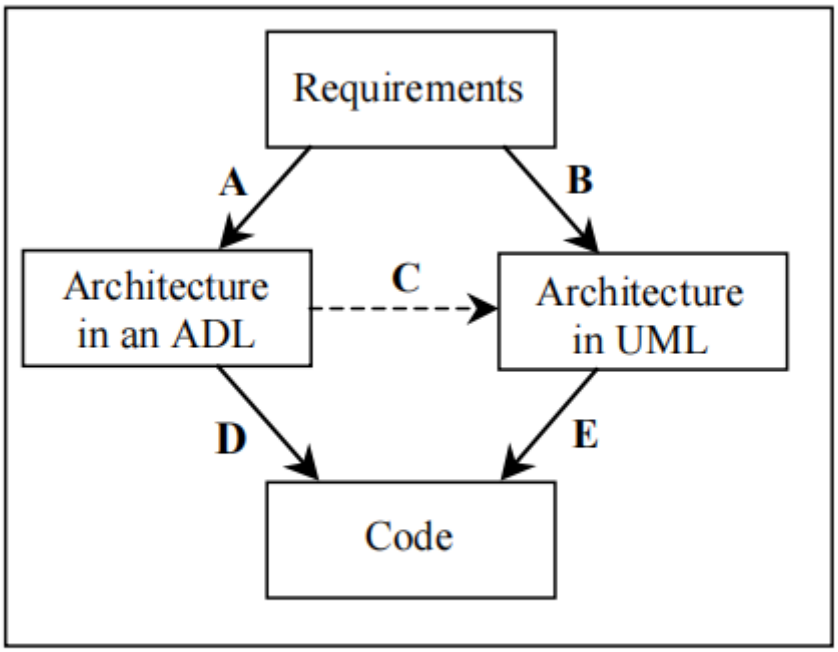


Figure 2: Software Architecture as a Bridge

其中，Adage 支持航空电子导航和制导架构框架的描述；Aesop 支持架构风格的使用；C2 支持使用基于事件的风格描述用户界面系统；Darwin 支持对分布式消息传递系统进行分析；Meta-H 为实时航空电控软件设计人员提供指导；Rapide 允许模拟架构设计，并具有分析这些仿真结果的工具；SADL 为架构精炼提供了一个形式化基础；UniCon 具有高级编译器，用于支持异构组件和连接器类型混合的架构设计；Wright 支持架构组件之间交互的形式化说明和分析。

2.the emergence of product line engineering

As noted earlier, one of the important trends has been the desire to exploit commonality across multiple products. Two specific manifestations of that trend are improvements in our ability to create product lines within an organization and the emergence of cross-vendor integration standards.

正如先前提到的一样，利用多个产品之间的共性已经成为了重要趋势之一。这种趋势的两个具体表现是我们改进了在组织内创建产品系列的能力，并出现了跨厂商集成标准。就产品系列而言，一个关键挑战是产品系列方法需要不同的开发方式。在单一产品方法中，架构必须根据特定产品需求进行评估。此外，单个产品可以独立建立，每个产品都有不同的架构。

然而，在产品系列方法中，还必须考虑系统家族的需求以及这些需求与每个特定实例相关联的需求之间的关系。图 3 说明了这种关系。尤其需要投入前期（以及持续）的时间来开发可重用的架构，该架构可以应用于每个产品。其他可重用的资产，例如组件、测试套件、工具等，通常也随之而来。

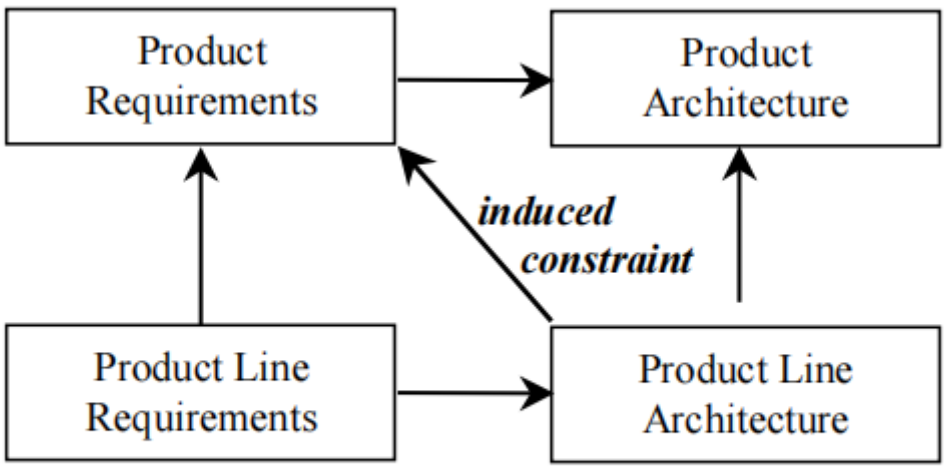


Figure 3: Product Line Architectures

尽管产品系列工程并未得到广泛应用，但我们正在开始更好地理解实现产品系列方法所需的过程、经济学和文物，在这些领域已经发布了许多产品系列成功案例研究。

与产品系列方法类似，跨供应商集成标准需要架构框架，以允许系统开发人员通过实例化该框架来配置多种特定系统。集成标准通常提供系统粘合剂（从概念上和通过运行时基础设施），以支持由多个供应商提供的零部件的集成。这样的标准可以是正式的国际标准（例如由 IEEE 或 ISO 赞助的标准）或由工业领袖推广的临时和实际标准。

产品线方法必须要考虑到系统族的需求，还需要考虑到这些需求和与每个特定实例相关的需求之间的联系，开发出可重用的架构。一些典型的例子包括 High Level Architecture (HLA) for Distributed Simulation, Sun’s Enterprise Java Beans (EJB) architecture。

3.the codification and dissemination of architectural design expertise

体系结构设计的发展一直面临的一个难点是缺乏关于如何开发好的架构的共享知识。针对这个问题，使用标准化的体系结构风格可以帮助解决。每种风格都适合于特定目的，但也有不适用的情况。因此，在识别和记录这些风格（以及它们更特定领域的变体）的同时，需要选择相应的分析方法进行配合。软件构建过程中，往往需要将没有按照相同方式设计的部分组合在一起，这会导致连接不匹配的桥接架构策略。虽然我们还没完全解决这一问题，并且尚未完全理解如何检测并修复不匹配，但已经开发出了一些技术来缓解这种情况。

学习报告 —— 杨伟钦

学习内容 1：软件系统结构设计概述

如文献 [1] 所述，“A critical issue in the design and construction of any complex software system is its architecture: that is, its gross organization as a collection of interacting components. ”，可见，软件体系结构研究的是软件系统的整体结构和各组件之间的联系，而不是具体的组件实现细节。

什么是软件体系结构设计的目标呢？文献 [1] 告诉我们，"A good architecture can help ensure that a system will satisfy key requirements in such areas as performance, reliability, portability, scalability, and interoperability. A bad architecture can be disastrous."。文献 [2] 告诉我们，软件体系结构设计的目的是为了实现软件系统的可修改性、性能、安全性、可靠性、健壮性、易使用性以及商业目标等质量属性。

软件体系结构设计的方法有很多，文献 [2] 给出了一些常用的软件体系结构设计风格：管道和过滤器、客户端-服务器、对等网络、发布-订阅、信息库、分层以及组合体系结构等。

文献 [2] 还介绍了一些体系结构的评估与改进方法，包括测量设计质量、故障树分析、安全性分析、权衡分析、成本效益分析、原型化等方法。

在学习了上述基础知识后，我将主要基于文献 [1] 介绍软件系统结构设计的未来方向。

学习内容 2：软件系统结构设计的未来

文献 [1] 中介绍了软件系统结构设计的三个未来方向：

- Changing Build-Versus Buy Balance

Throughout the history of software engineering a critical issue in the development of systems has been the decision about what parts of the system to obtain elsewhere and what parts to build in-house.

While the issue itself has not changed, economic pressures to reduce time-to-market are drastically changing the balance. For an increasing number of products, introducing a product a month early may be the difference between success and failure. In such situations building systems using software that others have written becomes the only feasible choice.

First, this trend heightens the need for industry-wide standards. Today's component technologies work at a fairly low level of architectural abstraction – essentially at the level of procedure call between objects. To obtain more significant integration will require higher-level architectural standards. This is likely to lead us from "component-based" engineering to "architecture-based" engineering, a shift that will emphasize the role of domain-specific integration architectures (such as EJB or HLA, mentioned earlier) in promoting composability. Thus, the trend toward architectural standards, noted earlier, is likely to become even more pronounced.

Second, this trend is leading to new software subcontracting processes. When integration is the critical issue, we can expect that externally contracted software will be held to much higher standards of architectural conformance. In many cases the standards will be commercial or governmental standards. In others, it is likely that sub-contractors will need to guarantee compatibility with product line architectures.

Third, this trend is leading toward standardization of notations and tools across vendors. When down-stream integrators are recombining systems, it is no longer acceptable to have in-house, idiosyncratic architecture descriptions and tools. This has led many to adopt languages like UML and XML for architectural modeling.

可见，软件之间的互相集成已经成为了未来软件开发的发展趋势，这一趋势强调了对软件体系结构的行业标准的需求，包括软件的体系结构一致性标准、供应商之间的符号和工具的标准化等等。

- Network-Centric Computing

Historically, software systems have been developed as closed systems, developed and operated under control of individual institutions. The constituent components may be acquired from external sources, but when incorporated in a system they come under control of the system designer. Architectures for such systems are either completely static – allowing no run time restructuring – or permit only a limited form

of run time variation.

However, within the world of pervasive services available over networks, systems may not have such centralized control. The Internet is an example of such an open system. For such systems a new set of software architecture challenges emerges. First, is the need for architectures that scale up to the size and variability of the Internet. While many of the same architectural paradigms will likely apply, the details of their implementation and specification will need to change.

For example, one attractive form of composition is implicit invocation – sometimes termed "publish-subscribe." Within this architectural style components are largely autonomous, interacting with other components by broadcasting messages that may be "listened to" by other components. Most systems that use this style, however, make many assumptions about properties of its use. For example, one typically assumes that event delivery is reliable, that centralized routing of messages will be sufficient, and that it makes sense to define a common vocabulary of events that are understood by all of the components. In an Internet-based setting all of these assumptions are questionable.

Second, is the need to support computing with dynamically-formed, task-specific, coalitions of distributed autonomous resources. The Internet hosts a wide variety of resources: primary information, communication mechanisms, applications that can be invoked, control that coordinates the use of resources, and services such as secondary (processed) information, simulation, editorial selection, or evaluation. These resources are independently developed and independently supported; they may even be transient. They can be composed to carry out specific tasks set by a user; in many cases the resources need not be specifically aware of the way they are being used, or even whether they are being used. Such coalitions lack direct control over the incorporated resources. Selection and composition of resources is likely to be done afresh for each task, as resources appear, change, and disappear. Unfortunately, it is hard to automate the selection and composition activity because of poor information about the character of services and hence with establishing correctness. Composition of components in this setting is difficult because it is hard to determine what assumptions each component makes about its operating context, let alone whether a set of components will interoperate well (or at all) and whether their combined functionality is what you need. Moreover, many useful resources exist but cannot be smoothly integrated because they make incompatible assumptions about component interaction. For example, it is hard to integrate a component packaged to interact via remote procedure calls with a component packaged to interact via shared data in a proprietary representation. These problems will provide new challenges for architecture description and analysis. In particular, the need to handle dynamically-evolving collections of components obtained from a variety of sources will require new techniques for managing architectural models at run time, and for evaluating the properties of those ensembles.

Third, is the related need to find architectures that flexibly accommodate commercial application service providers. In the future, applications will be composed out of a mix of local and remote computing capabilities on each desktop, requiring architectural

support that supports billing, security, etc.

Fourth, is the need to develop architectures that permit end users to do their own system composition. With the rapid growth of the Internet, an increasing number of users are in a position to assemble and tailor services. Such users may have minimal technical expertise, and yet will still want strong guarantees that the parts will work together in the ways they expect.

可见，在以网络为中心的软件系统中，软件系统并没有集中控制的方式，许多软件对资源或事件的假设都需要进行修改，对于体系结构的分析与评估需要重新进行。未来，需要构建能够灵活适应商业应用服务提供商的架构，包括远程通信、安全支持、分布式和云端计算等。此外，还可能需要开发允许用户定制自己的系统组合的体系结构。

- Pervasive Computing

A third related trend is toward pervasive computing in which the computing universe is populated by a rich variety of heterogeneous computing devices: toasters, home heating systems, entertainment systems, smart cars, etc. This trend is likely to lead to an explosion in the number of devices in our local environments – from dozens of devices to hundreds or thousands of devices. Moreover these devices are likely to be quite heterogeneous, requiring special considerations in terms of their physical resources and computing power. There are a number of consequent challenges for software architectures.

First, we will need architectures that are suited to systems in which resource usage is a critical issue. For example, it may be desirable to have an architecture that allows us to modify the fidelity of computation based on the power reserves at its disposal.

Second, architectures for these systems will have to be more flexible than they are today. In particular, devices are likely to come and go in an unpredictable fashion. Handling reconfiguration dynamically, while guaranteeing uninterrupted processing, is a hard problem.

Third, is the need for architectures that will better handle user mobility. Currently, our computing environments are configured manually and managed explicitly by the user. While this may be appropriate for environments in which we have only a few, relatively static, computing devices (such as a couple of PCs and laptops), it does not scale to environments with hundreds of devices. We must therefore find architectures that provide much more automated control over the management of computational services, and that permit users to move easily from one environment to another.

在普适计算阶段，我们需要让系统适配多种不同的计算设备，这需要合适的资源评估、分配和使用。因此，未来需要一种更加灵活的系统架构，允许不可预测的设备出现、移动或消失，实现对计算服务的自动化控制与迁移。

总的来说，软件体系结构的未来，必然要适应软件设计的复杂化、抽象化、集成化趋势，必然要适应数据网络化、分布式化的趋势，必然要适应设备多样化、计算边缘化的趋势。这要求我们构建出在规范上更加严格、在应用上更加灵活的软件体系结构

标准。

实践与收获：数据库管理

在物流管理系统项目中，我承担了数据库的设计和实现工作，这对整个软件体系结构起着至关重要的作用。通过参与数据库的开发，我获得了宝贵的学习经验，并提升了自己在数据库管理和优化方面的能力。

首先，我学习了数据库设计的基本原理和方法。在项目中，我通过分析系统需求和业务流程，设计了符合物流管理系统要求的数据库结构。我学习了如何根据实体和关系模型设计数据库表，并定义表之间的关联和约束。我还学习了数据库范式化和反范式化的概念，以及在设计过程中如何平衡数据一致性和性能方面的考虑。

其次，我熟悉了数据库管理系统（DBMS）的使用和优化。在项目中，我选择了适合物流管理系统的 DBMS，并学会了使用其提供的功能和工具。我学习了如何创建和管理数据库、定义和执行查询语言（如 SQL）来检索和操作数据。我还学习了数据库索引的概念和优化技巧，以提高查询性能和数据访问效率。通过优化数据库的结构和查询，我能够更好地满足系统的性能和可扩展性需求。

此外，我学习了数据库安全和备份的重要性。在物流管理系统中，数据的安全性和可靠性至关重要。我学习了如何设置合适的权限和访问控制策略，以确保只有授权用户可以访问和修改数据库。我还学习了数据库备份和恢复的方法，以防止数据丢失和系统故障。通过实施安全策略和备份机制，我能够保护和维护系统的数据完整性和可用性。

最后，通过参与数据库的设计和实现，我培养了团队合作和沟通的能力。在项目中，我与前端和后端开发人员密切合作，理解他们的需求，并提供有效的数据库解决方案。我学会了与团队成员协调工作，共同解决问题，并及时进行沟通和反馈。

综上所述，通过参与数据库的设计和实现，我不仅掌握了数据库设计和管理的核心知识和技术，还培养了团队合作和沟通的能力。这些学习和成长的经验将对我的职业发展产生积极的影响，并使我能够在数据库管理和优化领域不断提升自己，为企业的软件体系结构做出更大的贡献。

参考文献

本小组学习所参考的文献如下：

[1] Garlan, David. "Software architecture: a roadmap." Proceedings of the Conference on the Future of Software Engineering. 2000.

[2] Pfleeger, Shari Lawrence, and Joanne M. Atlee. Software engineering: theory and

practice. Pearson Education India, 2010.

[3] Garlan, David, and Mary Shaw. "An introduction to software architecture." Advances in software engineering and knowledge engineering. 1993. 1-39.

[4] Bass, Len, Paul Clements, and Rick Kazman. Software architecture in practice. Addison-Wesley Professional, 2003.

附录 B

故障树

根节点: 项目故障

中间节点: 订单管理故障、运输管理故障、仓库管理故障、资源管理故障、统计分析故障

订单管理故障:

叶节点 1: 订单创建失败

叶节点 2: 订单查询失败

叶节点 3: 订单修改失败

叶节点 4: 订单删除失败

运输管理故障:

叶节点 5: 货物调度失败

叶节点 6: 运输路线规划失败

叶节点 7: 运输跟踪失败

仓库管理故障:

叶节点 8: 仓库管理失败

叶节点 9: 库存管理失败

叶节点 10: 入库管理失败

叶节点 11: 出库管理失败

资源管理故障:

叶节点 12: 车辆管理失败

叶节点 13: 司机管理失败

叶节点 14: 设备管理失败

叶节点 15: 人员管理失败

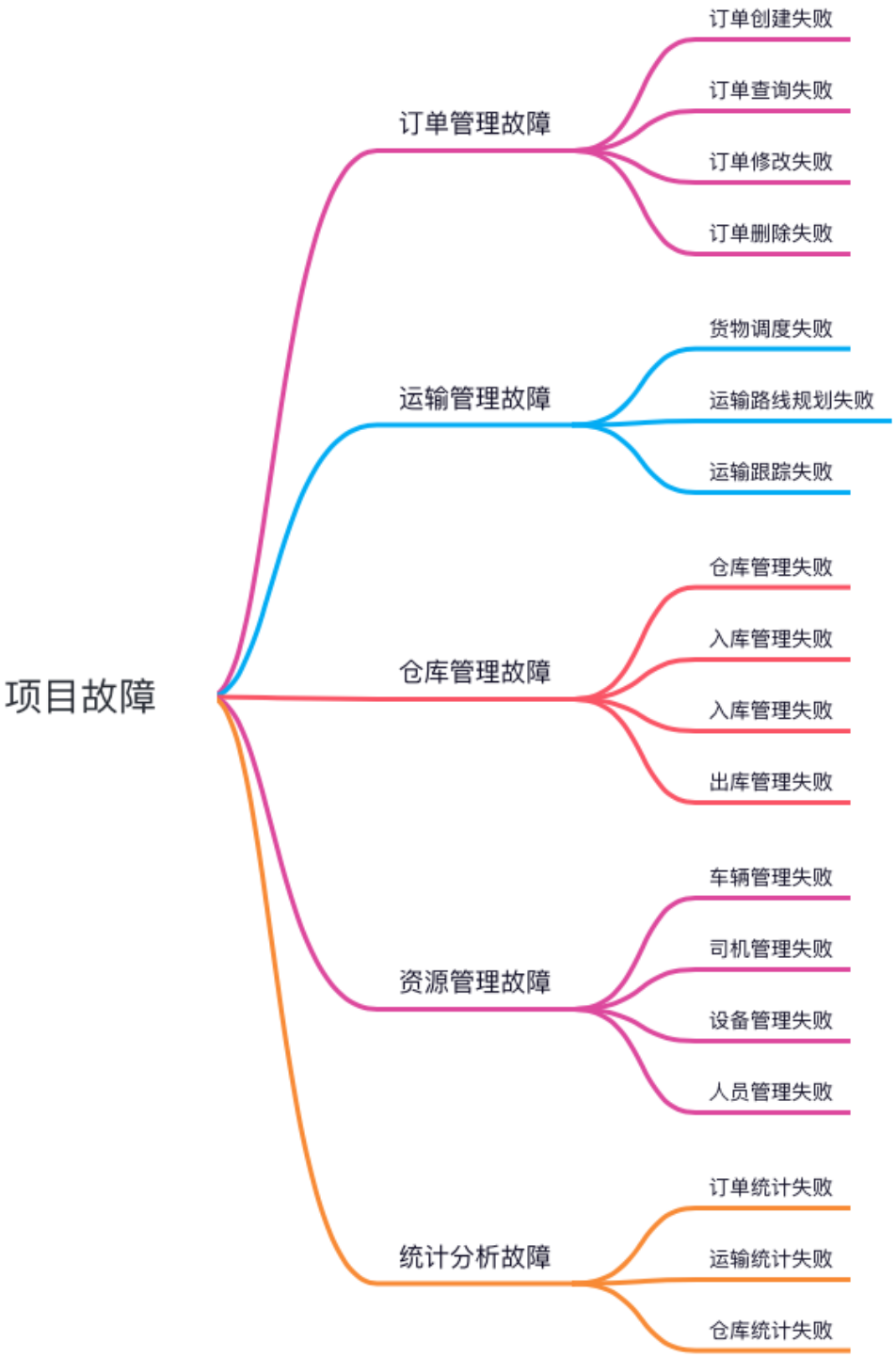
统计分析故障:

叶节点 16: 订单统计失败

叶节点 17: 运输统计失败

叶节点 18: 仓库统计失败

割集树


项目故障
订单管理故障
订单创建失败
订单查询失败
订单修改失败
订单删除失败
运输管理故障
货物调度失败
运输路线规划失败
运输跟踪失败
仓库管理故障
仓库管理失败
入库管理失败
入库管理失败
出库管理失败
资源管理故障
车辆管理失败
司机管理失败
设备管理失败
人员管理失败
统计分析故障
订单统计失败
运输统计失败
仓库统计失败