|  |  |  |
| --- | --- | --- |
| 文档状态： | 文档编号： |  |
| [ ] Draft  [√] Released  [ ]Modifying | 编 撰： |  |
| 编撰日期： |  |
| 保密级别： |  |
| 文档版本： | 1.0.0 |

仓库管理系统

（Warehouse management system）

**软件设计规格说明书**

**苏泳芝、林雪丹、陈佩君项目小组**

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **时间** | **修订内容** | **修订人** |
| 1 | 2025年04月10日 | 创建文档 | **苏泳芝** |
|  |  |  |  |
|  |  |  |  |

**目录**

[1 系统概述 4](#_Toc356139525)

[1.1文档概览 4](#_Toc356139526)

[1.2术语定义 4](#_Toc356139527)

[1.3软件系统简述 4](#_Toc356139528)

[1.4软件设计目标 4](#_Toc356139529)

[1.5 设计和实现约束 4](#_Toc356139530)

[1.6 参考文献 5](#_Toc356139531)

[2 界面设计模型 5](#_Toc356139532)

[2.1 用户界面设计 5](#_Toc356139533)

[2.1.1界面1的设计模型 5](#_Toc356139534)

[2.1.2界面2的设计模型 6](#_Toc356139535)

[2.2 用户界面流设计 6](#_Toc356139536)

[3体系结构模型 7](#_Toc356139537)

[4 接口设计 7](#_Toc356139538)

[5用例设计模型 7](#_Toc356139539)

[5.1传感器监测 7](#_Toc356139540)

[5.2系统配置 8](#_Toc356139541)

[6子系统设计模型 8](#_Toc356139542)

[6.1传感器监测子系统的设计模型 8](#_Toc356139543)

[6.1.1设计类图 8](#_Toc356139544)

[6.1.2交互图 9](#_Toc356139545)

[6.1.3类设计模型 10](#_Toc356139546)

[6.1.4状态图（可选） 10](#_Toc356139547)

[6.1.5活动图（可选） 10](#_Toc356139548)

[6.2子系统2的设计模型 10](#_Toc356139549)

[7.构件设计模型 10](#_Toc356139550)

[8.数据设计模型 10](#_Toc356139551)

[8.1 持久数据的组织结构 10](#_Toc356139552)

[8.2 持久数据操作（可选） 11](#_Toc356139553)

**仓库管理系统**

**软件设计规格说明书**

# 1 系统概述

## 1.1文档概览

本文档旨在为仓库管理系统的开发和实施提供指导。它涵盖了系统的设计、实现、测试和部署等方面，以确保项目的成功完成。

1.系统概述：描述项目的背景、目标、设计和实现的约束条件，以及参考文献。

2.界面设计模型：介绍用户界面的设计，包括界面布局、用户交互元素和操作流程。

3.体系结构模型：阐述系统的总体架构，包括模块划分、组件交互和数据流。

4.接口设计：详细说明系统内部和外部的接口，包括API规范和数据交换格式。

5.用例设计模型：描述系统的主要功能用例，包括传感器监测和系统配置等。

6.子系统设计模型：深入分析各个子系统的设计，包括类图、交互图、状态图和活动图。

7.构件设计模型：描述系统中各个构件的设计和实现细节。

8.数据设计模型：说明系统中数据的组织结构和持久化操作。

本文档主要面向以下读者：项目开发人员、项目管理人员、测试人员和仓库管理人员。

## 1.2术语定义

*本文档中使用的所有术语、缩写、标识符的完整定义，特别包括在设计模型中出现的每个UML构造型的定义。*

## 1.3软件系统简述

本仓库管理系统是一款旨在提高仓储操作效率、降低运营成本、增强数据准确性的软件产品。它通过集成先进的技术，如数据库管理、用户界面设计和网络通信，提供了一个全面、自动化的解决方案，以应对现代供应链管理的挑战。

* 整体目标

1. 提高效率：通过自动化流程减少人工操作，提高出入库操作的速度和准确性。
2. 实时监控：提供实时库存监控功能，帮助管理者及时响应库存变化。
3. 数据准确性：确保库存数据的准确性，减少因人为错误导致的库存不准确问题。
4. 决策支持：通过数据分析和报表生成，为企业管理层提供决策支持。

* 功能

1. 库存管理：包括入库、出库、库存盘点和库存调整等功能。
2. 订单处理：快速处理订单，优化发货流程。
3. 采购管理：支持采购计划的制定和订单跟踪。
4. 销售支持：提供实时库存数据，支持销售计划的制定。
5. 报表和分析：生成各种报表，进行数据分析，支持决策制定。

* 目标用户

1. 仓库管理人员：负责日常的仓库运营管理。
2. 采购人员：负责原材料和商品的采购计划和订单跟踪。
3. 销售团队：需要实时库存数据来支持销售活动。
4. 企业管理层：需要通过系统报表和数据分析来做出决策。

* 运行环境

1. 硬件环境：系统支持部署在物理服务器或云服务器上，建议配置为8核CPU、16GB内存、500GB硬盘。用户可通过配备主流操作系统的PC端访问系统。
2. 软件环境：服务器端推荐使用Windows操作系统，PC端支持Windows系列操作系统。数据库采用MySQL，系统基于Java Spring Boot和Layui开发框架。

## 1.4软件设计目标

一、功能性需求实现

（一）客户管理

本文档中的设计模型实现了客户信息的添加、编辑、删除和查询操作。用户可以通过系统界面方便地对客户资源进行维护和利用，满足了需求规格说明书中“客户管理”用例的功能性需求。系统会验证输入信息的有效性，确保必填项不为空，联系电话和银行账号为有效数字，并在删除客户信息前确认该客户无关联业务数据。

（二）供应商管理

设计模型实现了对供应商信息的全面管理，包括添加、编辑、删除和查询功能。这与需求规格说明书中的“供应商管理”用例完全对应。系统会检查必填项是否为空，联系电话和银行账号是否为有效数字，并在删除供应商信息前确认该供应商无未完成的采购订单。

（三）商品管理

设计模型实现了商品信息的添加、编辑、删除和查询功能，满足了需求规格说明书中“商品管理”用例的要求。系统会验证价格、数量、预警值是否为有效数字，并确保供应商通过下拉表选择，且删除商品信息前确认该商品无库存记录。

（四）商品进货

设计模型实现了商品进货信息的记录和库存数量的更新，满足了“商品进货”用例的功能性需求。系统会验证价格、数量是否为有效数字，并确保供应商和商品通过下拉表选择，同时筛选查询时开始时间不能晚于结束时间。

（五）商品退货查询

设计模型实现了商品退货信息的查询功能，满足了“商品退货查询”用例的需求。系统会根据用户选择的查询条件（如供应商名称、商品名称、时间等）筛选并显示退货信息列表，并在选择的开始时间晚于结束时间时默认无数据。

（六）商品销售

设计模型实现了商品销售信息的记录和库存数量的更新，满足了“商品销售”用例的功能性需求。系统会验证用户输入的查询条件是否有效，并确保筛选时选择的客户和商品通过下拉表选择。

（七）销售退货查询

设计模型实现了销售退货信息的查询功能，满足了“销售退货查询”用例的需求。系统会根据用户选择的查询条件筛选并显示退货信息列表，并在选择的开始时间晚于结束时间时默认无数据。

（八）部门管理

设计模型实现了部门信息的添加、编辑、删除和查询功能，满足了“部门管理”用例的要求。系统会检查必填项是否为空，并在删除部门信息前确认该部门无子部门。

（九）菜单管理

设计模型实现了系统菜单信息的管理，包括添加、编辑、删除和查询功能，满足了“菜单管理”用例的功能性需求。系统会验证菜单名称、父级菜单是否为空，并在删除菜单信息前确认该菜单无子菜单。

（十）权限管理

设计模型实现了权限信息的添加、编辑、删除和查询功能，满足了“权限管理”用例的要求。系统会验证权限名称、父级权限是否为空，并在删除权限信息时提示用户该权限已被分配给用户或角色的情况。

（十一）角色管理

设计模型实现了角色的添加、修改、删除、查询以及权限分配等功能，满足了“角色管理”用例的功能性需求。系统会确保角色名称唯一且信息完整，并遵循最小权限原则进行权限分配。

（十二）用户管理

设计模型实现了用户的添加、修改、删除、查询、重置密码和分配角色等功能，满足了“用户管理”用例的要求。系统会确保用户名唯一且信息完整，并在重置密码后及时通知用户。

（十三）系统公告

设计模型实现了系统公告的发布、修改、删除和查询功能，满足了“系统公告”用例的需求。系统会验证公告标题和内容是否为空，并确保公告能够及时发布和更新。

（十四）图标管理

设计模型实现了图标管理功能，满足了“图标管理”用例的需求，允许用户查看系统中引入的图标。

（十五）缓存管理

设计模型实现了缓存的查询、删除、清空和同步等操作，满足了“缓存管理”用例的功能性需求。

（十六）登录日志

设计模型实现了登录日志的记录和管理功能，满足了“登录日志”用例的需求。系统会记录用户的登录时间、登录 IP 地址等信息，并支持查询操作。

二、非功能性需求实现

（一）安全性

设计模型实现了以下安全性需求：

* 认证需求（Req-Authentication-001）：系统通过密码验证用户身份，确保只有授权用户可以使用软件。
* 权限控制需求（Req-Authorization-001）：在进行关键操作（如数据备份、恢复、删除库存记录、修改系统关键配置等）时，系统会要求输入正确的用户密码。
* 审计性需求（Req-Audit-001）：系统记录运行日志，包括登录登出时间、货物出入库操作记录、库存盘点记录、系统配置更改情况等，以供审计。

（二）兼容性

设计模型实现了以下兼容性需求：

* 与相关标准的兼容性（Req-Compatibility-001）：系统遵循仓储物流行业的相关业界标准，包括数据格式标准、接口规范等。
* 版本兼容性（Req-Compatibility-002）：新版本完全兼容旧版本的数据格式和操作习惯，保证数据的平滑迁移。

（三）可配置性

设计模型实现了可配置性需求（Req-Config-001），允许用户定制仓库的分区数量、货架布局、库存预警阈值、货物分类规则等配置参数，并支持不同品牌的扫码设备和库存盘点设备。

（四）可扩展性

设计模型为未来可能的扩展预留了接口，满足了可扩展性需求（Req-Extend-001）。系统支持接入企业 ERP 系统，实现数据实时同步，并为扩展智能仓储功能（如自动分拣、智能补货）提供了基础。

（五）可伸缩性

设计模型支持管理 100 个仓库、200 种货物，并为未来业务扩展到 200 个仓库、3000 种货物做好了准备，满足了可伸缩性需求（Req-Scalability-001）。系统在扩展时无需修改核心架构即可适应规模的增大。

（六）本地化与国际化

设计模型目前仅支持中文界面，满足了本地化需求（Req-Intl-001）。未来将根据市场需求逐步支持其他语言。

（七）可移植性

设计模型考虑了可移植性需求，系统将来需移植至多种主流服务器操作系统（如 Windows Server、Linux）环境运行。

## 1.5 设计和实现约束

无

## 1.6 参考文献

[1].软件工程.齐治昌，谭庆平，宁洪.北京:高等教育出版社，2012

[2].需求分析与设计.马素霞译.北京:机械工业出版社，2009

[3].面向使用的软件设计.刘正捷译.北京:机械工业出版社，2011

[4].文档模板：软件需求规格说明书.佛山大学，2025

[5].软件需求构思及描述文档.Mini-12306项目团队.

# 2 界面设计模型

*包括屏幕瞬时快照的静态图示、界面流、屏幕类图，见第八章。*

## 2.1 用户界面设计

### 2.1.1界面1的设计模型

#### 2.1.1.1 外观设计



#### 2.1.1.2 UML类表示

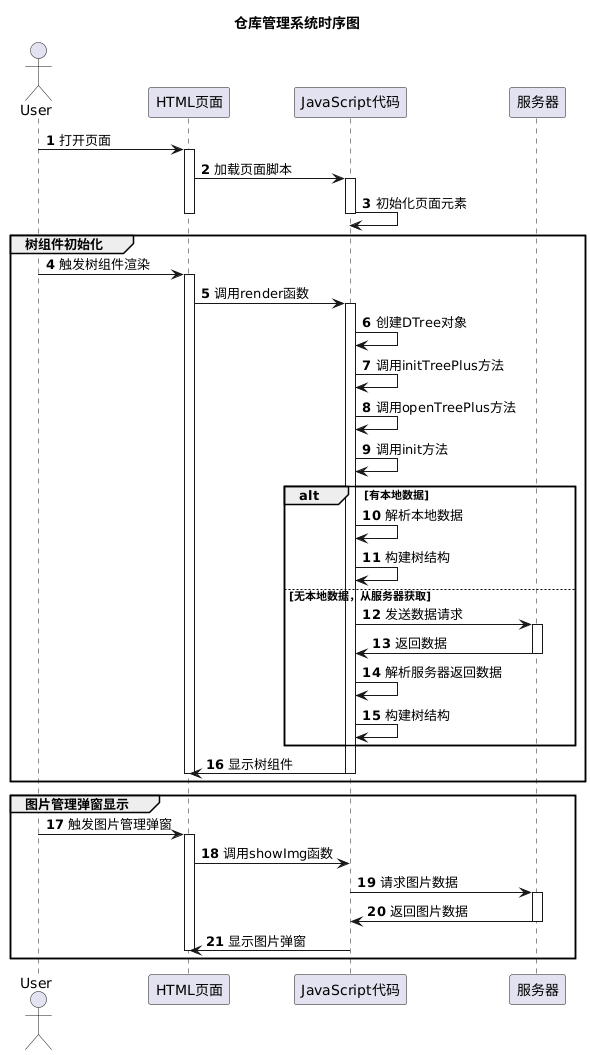


### 2.1.2界面2的设计模型

## 2.2 用户界面流设计

*界面流设计的主要目标是，确定屏幕之间的跳转关系*





# 3体系结构模型

见体系结构设计说明书

# 4 接口设计

1. 接口概述

仓库管理系统的接口设计旨在提供一个标准化的接口，以便前端应用能够与后端服务进行交互。所有接口均遵循RESTful API标准，使用HTTP协议进行数据传输，数据格式采用JSON。

接口请求地址格式为：域名 + 请求URL。

示例域名：http://warehouse管理系统.learnerhub.net

1. 接口定义表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 接口名称 | 接口地址 | 请求方式 | 功能描述 |
| 1 | 客户列表查询 | /customers | GET | 查询客户列表 |
| 2 | 添加客户 | /customers | POST | 添加新客户信息 |
| 3 | 修改客户信息 | /customers/{customerId} | PUT | 修改指定客户的详细信息 |
| 4 | 删除客户 | /customers/{customerId} | DELETE | 删除指定客户 |
| 5 | 商品列表查询 | /goods | GET | 查询商品列表 |
| 6 | 添加商品 | /goods | POST | 添加新商品信息 |
| 7 | 修改商品信息 | /goods/{goodsId} | PUT | 修改指定商品的详细信息 |
| 8 | 删除商品 | /goods/{goodsId} | DELETE | 删除指定商品 |
| 9 | 商品进货记录 | /inport | POST | 记录商品进货信息 |
| 10 | 查询进货记录 | /inport | GET | 查询商品进货记录 |
| 11 | 商品销售记录 | /sales | POST | 记录商品销售信息 |
| 12 | 查询销售记录 | /sales | GET | 查询商品销售记录 |

3. 接口详细设计

3.1 客户管理接口

3.1.1 查询客户列表

请求方法：GET

请求URL：/customers

请求参数：

name（可选）：客户名称，用于模糊查询。

page（可选，默认为1）：分页参数，当前页码。

size（可选，默认为10）：分页参数，每页显示数量。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"total": 100,

"currentPage": 1,

"pageSize": 10,

"customers": [

{

"customerId": 1,

"name": "客户A",

"contact": "张三",

"phone": "13800138000",

"email": "customerA@example.com"

},

...

]

},

"msg": "OK"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "请求参数错误"

}

3.1.2 添加客户

请求方法：POST

请求URL：/customers

请求参数：

name：客户名称，必填。

contact：联系人，必填。

phone：联系电话，必填。

email：电子邮箱，可选。

请求头：

Content-Type: application/json

请求体：

json

{

"name": "客户B",

"contact": "李四",

"phone": "13800138001",

"email": "customerB@example.com"

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"customerId": 2,

"name": "客户B",

"contact": "李四",

"phone": "13800138001",

"email": "customerB@example.com"

},

"msg": "客户添加成功"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "客户信息格式错误"

}

3.1.3 修改客户信息

请求方法：PUT

请求URL：/customers/{customerId}

请求参数：

customerId：客户ID，路径参数。

name：客户名称，可选。

contact：联系人，可选。

phone：联系电话，可选。

email：电子邮箱，可选。

请求头：

Content-Type: application/json

请求体：

json

{

"name": "客户B更新",

"contact": "王五",

"phone": "13800138002",

"email": "customerB\_updated@example.com"

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"customerId": 2,

"name": "客户B更新",

"contact": "王五",

"phone": "13800138002",

"email": "customerB\_updated@example.com"

},

"msg": "客户信息修改成功"

}

错误响应：

json

{

"code": 404,

"data": null,

"msg": "客户不存在"

}

3.1.4 删除客户

请求方法：DELETE

请求URL：/customers/{customerId}

请求参数：

customerId：客户ID，路径参数。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": null,

"msg": "客户删除成功"

}

错误响应：

json

{

"code": 404,

"data": null,

"msg": "客户不存在"

}

3.2 商品管理接口

3.2.1 查询商品列表

请求方法：GET

请求URL：/goods

请求参数：

name（可选）：商品名称，用于模糊查询。

category（可选）：商品分类。

page（可选，默认为1）：分页参数，当前页码。

size（可选，默认为10）：分页参数，每页显示数量。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"total": 50,

"currentPage": 1,

"pageSize": 10,

"goods": [

{

"goodsId": 1,

"name": "商品A",

"category": "类别1",

"price": 100.0,

"stock": 100

},

...

]

},

"msg": "OK"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "请求参数错误"

}

3.2.2 添加商品

请求方法：POST

请求URL：/goods

请求参数：

name：商品名称，必填。

category：商品分类，必填。

price：商品价格，必填。

stock：库存数量，必填。

请求头：

Content-Type: application/json

请求体：

json

{

"name": "商品B",

"category": "类别2",

"price": 200.0,

"stock": 50

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"goodsId": 2,

"name": "商品B",

"category": "类别2",

"price": 200.0,

"stock": 50

},

"msg": "商品添加成功"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "商品信息格式错误"

}

3.2.3 修改商品信息

请求方法：PUT

请求URL：/goods/{goodsId}

请求参数：

goodsId：商品ID，路径参数。

name：商品名称，可选。

category：商品分类，可选。

price：商品价格，可选。

stock：库存数量，可选。

请求头：

Content-Type: application/json

请求体：

json

{

"name": "商品B更新",

"category": "类别3",

"price": 250.0,

"stock": 30

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"goodsId": 2,

"name": "商品B更新",

"category": "类别3",

"price": 250.0,

"stock": 30

},

"msg": "商品信息修改成功"

}

错误响应：

json

{

"code": 404,

"data": null,

"msg": "商品不存在"

}

3.2.4 删除商品

请求方法：DELETE

请求URL：/goods/{goodsId}

请求参数：

goodsId：商品ID，路径参数。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": null,

"msg": "商品删除成功"

}

错误响应：

json

{

"code": 404,

"data": null,

"msg": "商品不存在"

}

3.3 商品进货接口

3.3.1 记录商品进货信息

请求方法：POST

请求URL：/inport

请求参数：

goodsId：商品ID，必填。

supplierId：供应商ID，必填。

quantity：进货数量，必填。

price：进货单价，必填。

date：进货日期，可选，默认为当前日期。

请求头：

Content-Type: application/json

请求体：

json

{

"goodsId": 1,

"supplierId": 1,

"quantity": 100,

"price": 100.0,

"date": "2024-07-22"

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"inportId": 1,

"goodsId": 1,

"supplierId": 1,

"quantity": 100,

"price": 100.0,

"date": "2024-07-22"

},

"msg": "进货记录添加成功"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "进货信息格式错误"

}

3.3.2 查询商品进货记录

请求方法：GET

请求URL：/inport

请求参数：

goodsId（可选）：商品ID，用于过滤特定商品的进货记录。

supplierId（可选）：供应商ID，用于过滤特定供应商的进货记录。

startDate（可选）：开始日期，用于过滤特定时间段的进货记录。

endDate（可选）：结束日期，用于过滤特定时间段的进货记录。

page（可选，默认为1）：分页参数，当前页码。

size（可选，默认为10）：分页参数，每页显示数量。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"total": 50,

"currentPage": 1,

"pageSize": 10,

"inportRecords": [

{

"inportId": 1,

"goodsId": 1,

"supplierId": 1,

"quantity": 100,

"price": 100.0,

"date": "2024-07-22"

},

...

]

},

"msg": "OK"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "请求参数错误"

}

3.4 商品销售接口

3.4.1 记录商品销售信息

请求方法：POST

请求URL：/sales

请求参数：

goodsId：商品ID，必填。

customerId：客户ID，必填。

quantity：销售数量，必填。

price：销售单价，必填。

date：销售日期，可选，默认为当前日期。

请求头：

Content-Type: application/json

请求体：

json

{

"goodsId": 1,

"customerId": 1,

"quantity": 10,

"price": 120.0,

"date": "2024-07-22"

}

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"saleId": 1,

"goodsId": 1,

"customerId": 1,

"quantity": 10,

"price": 120.0,

"date": "2024-07-22"

},

"msg": "销售记录添加成功"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "销售信息格式错误"

}

3.4.2 查询商品销售记录

请求方法：GET

请求URL：/sales

请求参数：

goodsId（可选）：商品ID，用于过滤特定商品的销售记录。

customerId（可选）：客户ID，用于过滤特定客户的销售记录。

startDate（可选）：开始日期，用于过滤特定时间段的销售记录。

endDate（可选）：结束日期，用于过滤特定时间段的销售记录。

page（可选，默认为1）：分页参数，当前页码。

size（可选，默认为10）：分页参数，每页显示数量。

请求头：

Content-Type: application/json

请求体：无

响应格式：

成功响应：

json

{

"code": 0,

"data": {

"total": 50,

"currentPage": 1,

"pageSize": 10,

"salesRecords": [

{

"saleId": 1,

"goodsId": 1,

"customerId": 1,

"quantity": 10,

"price": 120.0,

"date": "2024-07-22"

},

...

]

},

"msg": "OK"

}

错误响应：

json

{

"code": 400,

"data": null,

"msg": "请求参数错误"

}

4. 错误码说明

400：请求参数错误。

404：资源不存在（如客户、商品等）。

500：服务器内部错误。

5. 示例请求和响应

以下是一些示例请求和响应，用于说明接口的使用方法。

示例1：查询客户列表

请求：

http

GET /customers?page=1&size=10

响应：

json

{

"code": 0,

"data": {

"total": 100,

"currentPage": 1,

"pageSize": 10,

"customers": [

{

"customerId": 1,

"name": "客户A",

"contact": "张三",

"phone": "13800138000",

"email": "customerA@example.com"

},

...

]

},

"msg": "OK"

}

示例2：添加新客户

请求：

http

POST /customers

Content-Type: application/json

{

"name": "客户B",

"contact": "李四",

"phone": "13800138001",

"email": "customerB@example.com"

}

响应：

json

{

"code": 0,

"data": {

"customerId": 2,

"name": "客户B",

"contact": "李四",

"phone": "13800138001",

"email": "customerB@example.com"

},

"msg": "客户添加成功"

}

示例3：查询商品列表

请求：

http

GET /goods?page=1&size=10

响应：

json

{

"code": 0,

"data": {

"total": 50,

"currentPage": 1,

"pageSize": 10,

"goods": [

{

"goodsId": 1,

"name": "商品A",

"category": "类别1",

"price": 100.0,

"stock": 100

},

...

]

},

"msg": "OK"

}

示例4：添加新商品

请求：

http

POST /goods

Content-Type: application/json

{

"name": "商品B",

"category": "类别2",

"price": 200.0,

"stock": 50

}

响应：

json

{

"code": 0,

"data": {

"goodsId": 2,

"name": "商品B",

"category": "类别2",

"price": 200.0,

"stock": 50

},

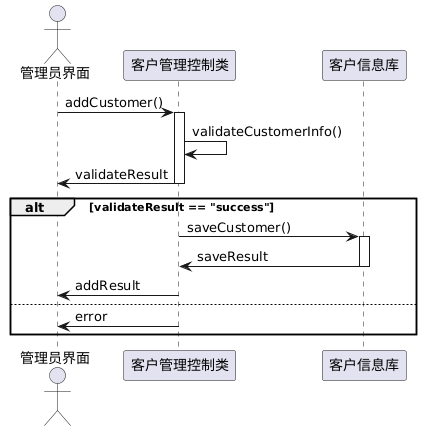
"msg": "商品添加成功"

}

# 5用例设计模型

*以UML交互图的形式逐个用例地表示其实现方案，。在用例设计步骤生成的设计类图应该整合至相应的子系统、构件或类设计模型之中，不必在此示出。*

# 5.1功能性需求



1. **客户管理用例实现的顺序图**

实现对客户信息的有效管理，包括添加、编辑、删除和查询操作，方便企业对客户资源进行维护和利用。

1. 管理员界面（CustomerUI）：

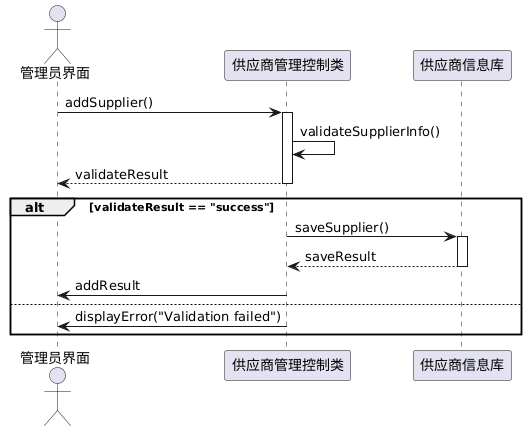
* 管理员在客户管理界面输入客户信息（如客户名称、联系电话、银行账号等）。
* 管理员点击“添加客户”按钮，向客户管理控制类发送addCustomer请求。

2. 客户管理控制类（CustomerManager）：

* 接收到addCustomer请求后，调用validateCustomerInfo方法对客户信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveCustomer方法将客户信息保存到客户信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 客户信息库（CustomerLib）：

* 接收到validateCustomerInfo请求后，对客户信息进行验证，确保必填项不为空，联系电话和银行账号为有效数字。
* 返回验证结果validateResult。
* 接收到saveCustomer请求后，将客户信息保存到数据库中。
* 返回保存结果saveResult。



**(b) 供应商管理用例实现的顺序图**

对供应商信息进行全面管理，包括添加、编辑、删除和查询，保障企业供应链的稳定。

1. 管理员界面（SupplierUI）：

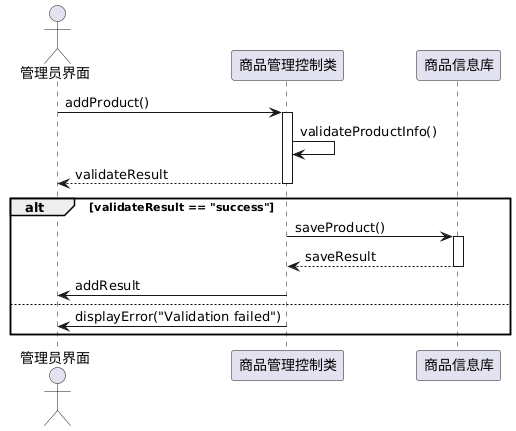
* 管理员在供应商管理界面输入供应商信息（如供应商名称、联系人、联系电话等）。
* 管理员点击“添加供应商”按钮，向供应商管理控制类发送addSupplier()请求。

2. 供应商管理控制类（SupplierManager）：

* 接收到addSupplier()请求后，调用validateSupplierInfo()方法对供应商信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveSupplier()方法将供应商信息保存到供应商信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 供应商信息库（SupplierLib）：

* 接收到validateSupplierInfo()请求后，对供应商信息进行验证，确保必填项不为空，联系电话和银行账号为有效数字。
* 返回验证结果validateResult。
* 接收到saveSupplier()请求后，将供应商信息保存到数据库中。
* 返回保存结果saveResult。



**(c) 商品管理用例实现的顺序图**

对商品信息进行有效管理，包括添加、编辑、删除和查询，方便企业对商品进行库存管理和销售。

1. 管理员界面（ProductUI）：

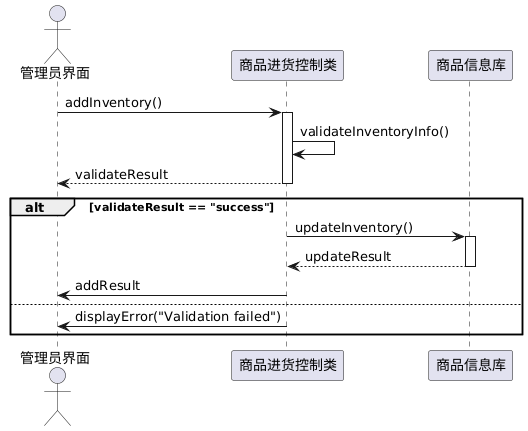
* 管理员在商品管理界面输入商品信息（如商品名称、规格、价格等）。
* 管理员点击“添加商品”按钮，向商品管理控制类发送addProduct()请求。

2. 商品管理控制类（ProductManager）：

* 接收到addProduct()请求后，调用validateProductInfo()方法对商品信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveProduct()方法将商品信息保存到商品信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 商品信息库（ProductLib）：

* 接收到validateProductInfo()请求后，对商品信息进行验证，确保必填项不为空，价格、数量、预警值为有效数字。
* 返回验证结果validateResult。
* 接收到saveProduct()请求后，将商品信息保存到数据库中。
* 返回保存结果saveResult。



**(d) 商品进货用例实现的顺序图**

记录商品的进货信息，更新商品库存数量，确保企业库存充足。

1. 管理员界面（InventoryUI）：

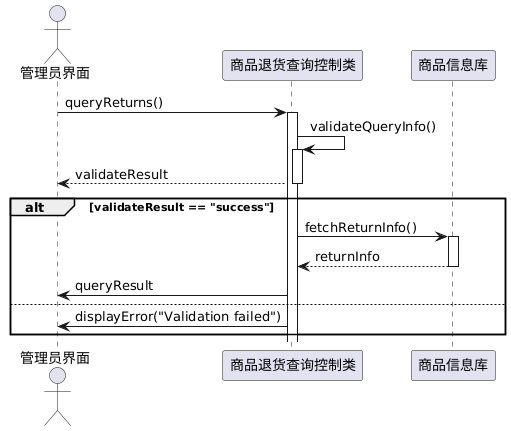
* 管理员在商品进货管理界面输入进货相关信息（如供应商名称、商品名称、数量、价格、支付类型等）。
* 管理员点击“添加进货”按钮，向商品进货控制类发送addInventory()请求。

2. 商品进货控制类（InventoryManager）：

* 接收到addInventory()请求后，调用validateInventoryInfo()方法对进货信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用updateInventory()方法更新商品库存信息。
* 更新完成后，返回更新结果updateResult。
* 将进货结果addResult返回给管理员界面。

3. 商品信息库（InventoryLib）：

* 接收到validateInventoryInfo()请求后，对进货信息进行验证，确保必填项不为空，价格、数量为有效数字。
* 返回验证结果validateResult。
* 接收到updateInventory()请求后，更新商品库存信息。
* 返回更新结果updateResult。



**(e) 商品退货查询用例实现的顺序图**

方便用户查询商品的退货信息，了解退货情况，以便进行库存管理和客户服务。

1. 管理员界面（ReturnUI）：

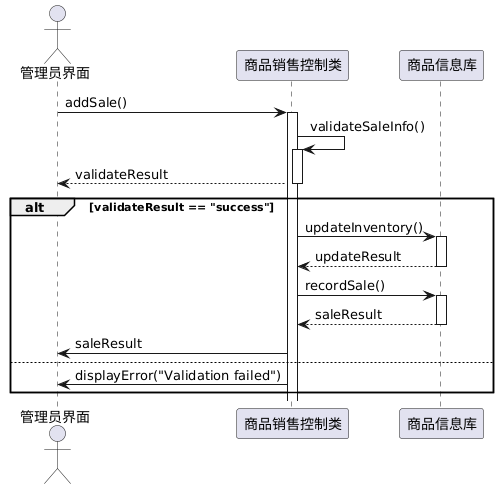
* 管理员在商品退货查询界面选择查询条件（如供应商名称、商品名称、开始时间和结束时间等）。
* 管理员点击“查询退货信息”按钮，向商品退货查询控制类发送queryReturns()请求。

2. 商品退货查询控制类（ReturnManager）：

* 接收到queryReturns()请求后，调用validateQueryInfo()方法对查询条件进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用fetchReturnInfo()方法从商品信息库获取退货信息。
* 获取完成后，返回退货信息returnInfo。
* 将查询结果queryResult返回给管理员界面。

3. 商品信息库（ProductLib）：

* 接收到fetchReturnInfo()请求后，根据查询条件从数据库中检索退货信息。
* 返回退货信息returnInfo。



**(f) 商品销售用例实现的顺序图**

记录商品的销售信息，更新商品库存数量，实现企业的销售业务。

1. 管理员界面（SalesUI）：

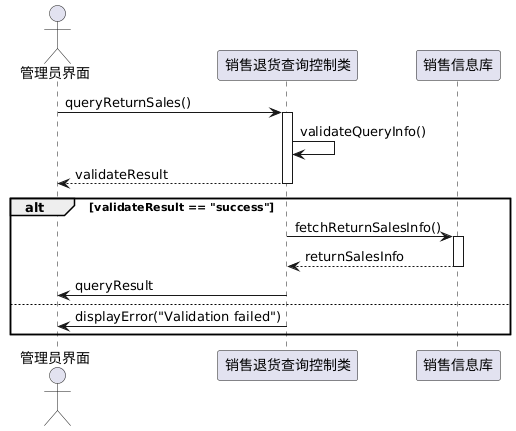
* 管理员在商品销售管理页面输入销售相关信息（如客户名称、商品名称、数量、价格、支付类型等）。
* 管理员点击“添加销售”按钮，向商品销售控制类发送addSale()请求。

2. 商品销售控制类（SalesManager）：

* 接收到addSale()请求后，调用validateSaleInfo()方法对销售信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用updateInventory()方法更新商品库存信息。
* 更新完成后，调用recordSale()方法记录销售信息。
* 记录完成后，返回销售结果saleResult。
* 将销售结果saleResult返回给管理员界面。

3. 商品信息库（InventoryLib）：

* 接收到validateSaleInfo()请求后，对销售信息进行验证，确保必填项不为空，价格、数量为有效数字。
* 返回验证结果validateResult。
* 接收到updateInventory()请求后，更新商品库存信息。
* 接收到recordSale()请求后，记录销售信息到数据库中。
* 返回销售结果saleResult。



**(g) 销售退货查询用例实现的顺序图**

方便用户查询商品的销售退货信息，了解销售退货情况，以便进行库存管理和客户服务。

1. 管理员界面（ReturnQueryUI）：

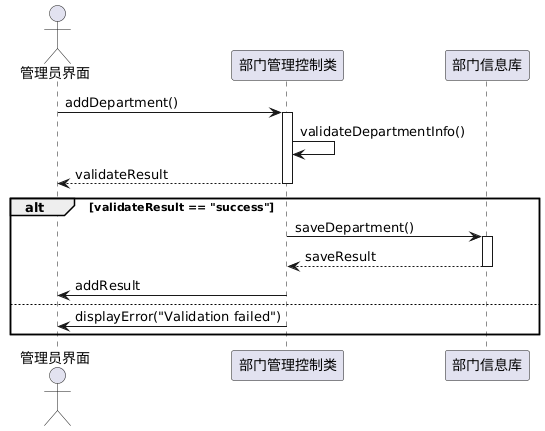
* 管理员在销售退货查询界面选择查询条件（如商品名称、客户名称、开始时间和结束时间等）。
* 管理员点击“查询退货信息”按钮，向销售退货查询控制类发送queryReturnSales()请求。

2. 销售退货查询控制类（ReturnSalesManager）：

* 接收到queryReturnSales()请求后，调用validateQueryInfo()方法对查询条件进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用fetchReturnSalesInfo()方法从销售信息库获取销售退货信息。
* 获取完成后，返回销售退货信息returnSalesInfo。
* 将查询结果queryResult返回给管理员界面。

3. 销售信息库（SalesLib）：

* 接收到fetchReturnSalesInfo()请求后，根据查询条件从数据库中检索销售退货信息。
* 返回销售退货信息returnSalesInfo。



**(h) 部门管理用例实现的顺序图**

对企业的部门信息进行管理，包括添加、编辑、删除和查询，便于组织架构的维护和管理。

1. 管理员界面（DepartmentUI）：

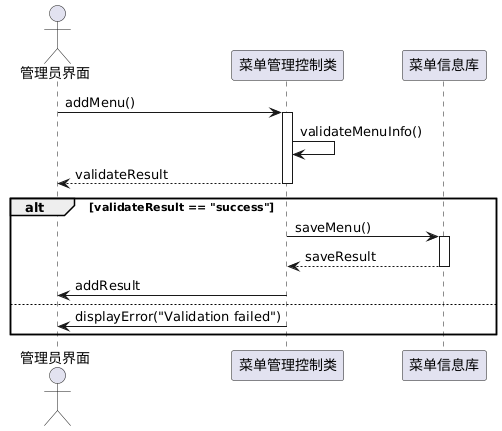
* 管理员在部门管理界面输入部门信息（如父级部门、部门名称、部门描述等）。
* 管理员点击“添加部门”按钮，向部门管理控制类发送addDepartment()请求。

2. 部门管理控制类（DepartmentManager）：

* 接收到addDepartment()请求后，调用validateDepartmentInfo()方法对部门信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveDepartment()方法将部门信息保存到部门信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 部门信息库（DepartmentLib）：

* 接收到validateDepartmentInfo()请求后，对部门信息进行验证，确保必填项不为空。
* 返回验证结果validateResult。
* 接收到saveDepartment()请求后，将部门信息保存到数据库中。
* 返回保存结果saveResult。



**(i) 菜单管理用例实现的顺序图**

对系统的菜单信息进行管理，包括添加、编辑、删除和查询，实现系统菜单的定制化。

1. 管理员界面（MenuUI）：

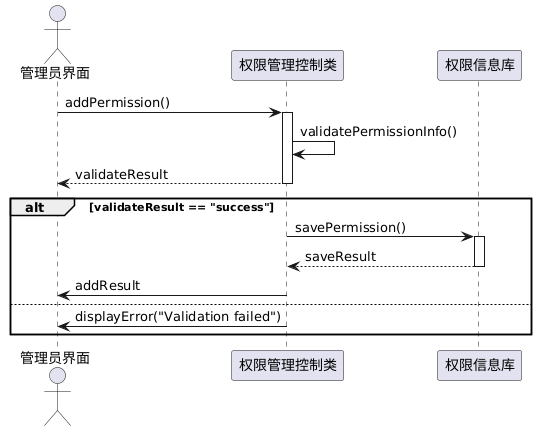
* 管理员在菜单管理界面输入菜单信息（如菜单名称、菜单图标等）。
* 管理员点击“添加菜单”按钮，向菜单管理控制类发送addMenu()请求。

2. 菜单管理控制类（MenuManager）：

* 接收到addMenu()请求后，调用validateMenuInfo()方法对菜单信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveMenu()方法将菜单信息保存到菜单信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 菜单信息库（MenuLib）：

* 接收到validateMenuInfo()请求后，对菜单信息进行验证，确保必填项不为空。
* 返回验证结果validateResult。
* 接收到saveMenu()请求后，将菜单信息保存到数据库中。
* 返回保存结果saveResult。



**(j) 权限管理用例实现的顺序图**

对系统的权限信息进行管理，包括添加、编辑、删除和查询，确保用户只能访问其被授权的功能和数据。

1. 管理员界面（PermissionUI）：

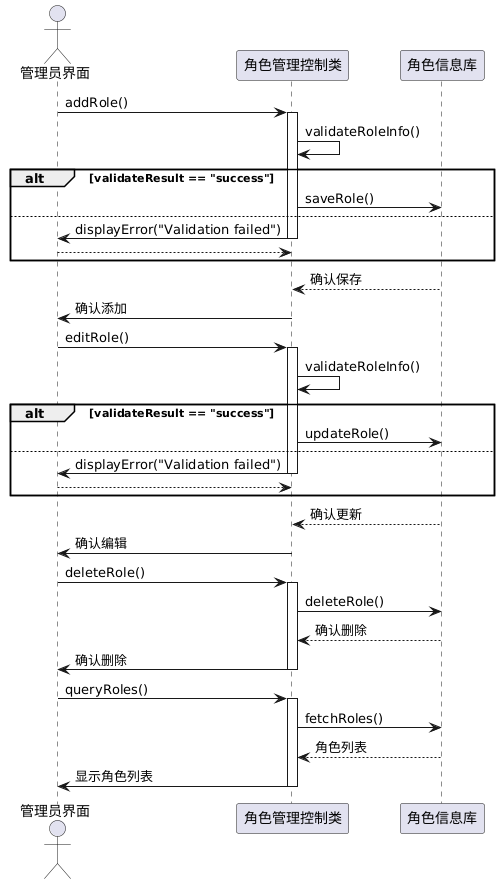
* 管理员在权限管理界面输入权限信息（如权限名称、权限编码等）。
* 管理员点击“添加权限”按钮，向权限管理控制类发送addPermission()请求。

2. 权限管理控制类（PermissionManager）：

* 接收到addPermission()请求后，调用validatePermissionInfo()方法对权限信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用savePermission()方法将权限信息保存到权限信息库。
* 保存完成后，返回保存结果saveResult。
* 将添加结果addResult返回给管理员界面。

3. 权限信息库（PermissionLib）：

* 接收到validatePermissionInfo()请求后，对权限信息进行验证，确保必填项不为空。
* 返回验证结果validateResult。
* 接收到savePermission()请求后，将权限信息保存到数据库中。
* 返回保存结果saveResult。



**(k) 角色管理用例实现的顺序图**

对系统的角色进行管理，包括角色的添加、修改、删除、查询以及权限分配等操作，以实现对用户访问权限的细粒度控制。

1. 管理员界面（RoleUI）：

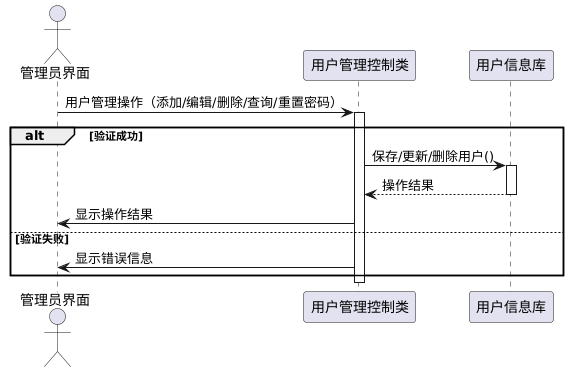
* 管理员在角色管理界面输入角色信息（如角色名称、角色描述等）或选择要编辑的角色。
* 管理员执行添加、编辑、删除或查询操作，向角色管理控制类发送相应的请求。

2. 角色管理控制类（RoleManager）：

* 根据管理员的操作，调用addRole()、editRole()、deleteRole()或queryRoles()方法。
* 对于添加和编辑操作，调用validateRoleInfo()方法对角色信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveRole()方法将角色信息保存到角色信息库，或调用updateRole()更新现有角色信息。
* 如果是删除操作，调用deleteRole()方法从角色信息库中删除角色信息。
* 查询操作则调用fetchRoles()方法从角色信息库获取角色列表。
* 将操作结果返回给管理员界面。

3. 角色信息库（RoleLib）：

* 接收到验证、保存、更新或删除请求后，对角色信息进行相应的验证、保存、更新或删除操作。
* 返回操作结果。



**(l) 用户管理用例实现的顺序图**

对系统的用户进行管理，包括用户的添加、修改、删除、查询、重置密码以及分配角色等操作。

1. 管理员界面（UserUI）：

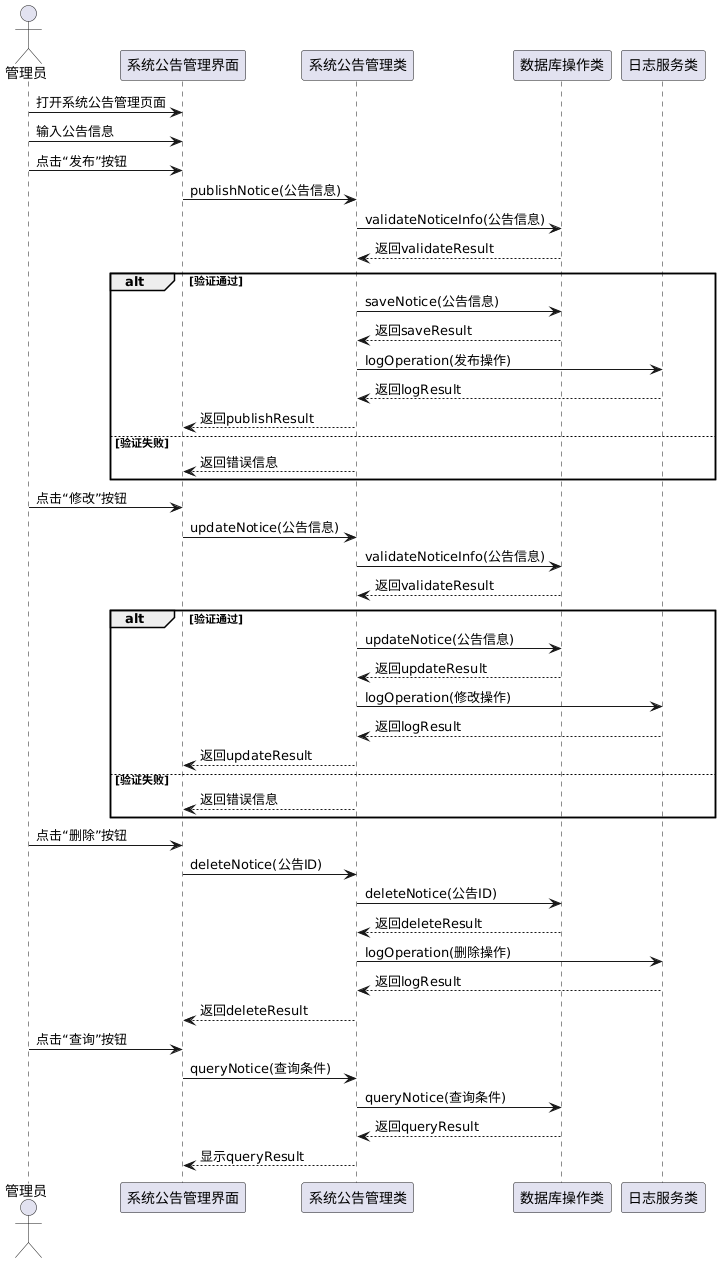
* 管理员在用户管理界面输入用户信息（如用户名、密码、角色等）或选择要编辑的用户。
* 管理员执行添加、编辑、删除、查询或重置密码操作，向用户管理控制类发送相应的请求。

2. 用户管理控制类（UserManager）：

* 接收到请求后，根据操作类型调用validateUserInfo()、saveUser()、updateUser()、deleteUser()或resetPassword()方法。
* 对用户信息进行验证，确保所有必填项不为空，并且符合系统要求。
* 如果验证通过，执行相应的数据库操作来添加、更新或删除用户信息。
* 对于重置密码操作，生成新密码并通过安全的方式通知用户。
* 查询操作则从数据库检索用户信息。
* 操作完成后，返回操作结果operationResult。

3. 用户信息库（UserLib）：

* 接收到用户管理控制类的请求后，执行相应的数据库操作（添加、编辑、删除、查询）。
* 返回操作结果operationResult。



**(n) 系统公告用例实现的顺序图**

系统公告功能允许超级管理员、系统管理员和基础数据管理员发布、修改、删除和查询系统公告。通过系统公告功能，管理员可以向系统用户传达重要信息，如系统更新、维护通知或其他重要事项。公告的发布和管理操作会被记录在日志中，以便后续审计和监控。

1. 管理员界面（NoticeManagerUI）：

* 管理员通过系统菜单进入系统公告管理界面。
* 管理员在系统公告管理界面输入公告信息（如公告标题、公告内容等）。
* 管理员选择操作类型（发布、修改、删除、查询）并点击相应按钮：
* 点击“发布”按钮，发送publishNotice请求。
* 点击“修改”按钮，发送updateNotice请求。
* 点击“删除”按钮，发送deleteNotice请求。
* 点击“查询”按钮，发送queryNotice请求。

2. 系统公告管理类（NoticeManager）：

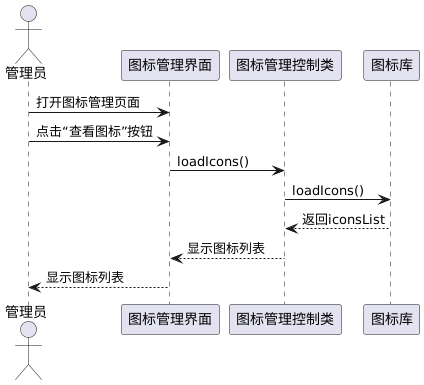
* 接收到publishNotice请求后：
* 调用validateNoticeInfo方法对公告信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用saveNotice方法将公告信息保存到数据库。
* 保存完成后，调用logOperation方法记录操作日志。
* 将发布结果publishResult返回给管理员界面。
* 接收到updateNotice请求后：
* 调用validateNoticeInfo方法对公告信息进行验证。
* 验证完成后，返回验证结果validateResult。
* 如果验证通过，调用updateNotice方法更新公告信息。
* 更新完成后，调用logOperation方法记录操作日志。
* 将修改结果updateResult返回给管理员界面。
* 接收到deleteNotice请求后：
* 调用deleteNotice方法删除公告记录。
* 删除完成后，调用logOperation方法记录操作日志。
* 将删除结果deleteResult返回给管理员界面。
* 接收到queryNotice请求后：
* 调用queryNotice方法查询公告信息。
* 查询完成后，将查询结果queryResult返回给管理员界面。

3. 数据库操作类（DBService）：

* 接收到validateNoticeInfo请求后，对公告信息进行验证，确保必填项不为空。
* 返回验证结果validateResult。
* 接收到saveNotice请求后，将公告信息保存到数据库中。
* 返回保存结果saveResult。
* 接收到updateNotice请求后，更新数据库中的公告信息。
* 返回更新结果updateResult。
* 接收到deleteNotice请求后，删除数据库中的公告记录。
* 返回删除结果deleteResult。
* 接收到queryNotice请求后，从数据库中查询公告信息。
* 返回查询结果queryResult。

4. 日志服务类（LogService）：

* 接收到logOperation请求后，记录公告管理的操作日志。
* 返回日志记录结果logResult。



**(m) 图标管理用例实现的顺序图**

图标管理功能允许超级管理员、系统管理员和基础数据管理员查看系统中引入的图标。通过图标管理功能，用户可以方便地浏览和选择系统支持的图标，以便在其他功能模块中使用。

1. 管理员界面（IconsUI）：

* 管理员通过系统菜单进入图标管理界面。
* 管理员在图标管理界面点击“查看图标”按钮，请求加载图标列表。

2. 图标管理控制类（IconsManager）：

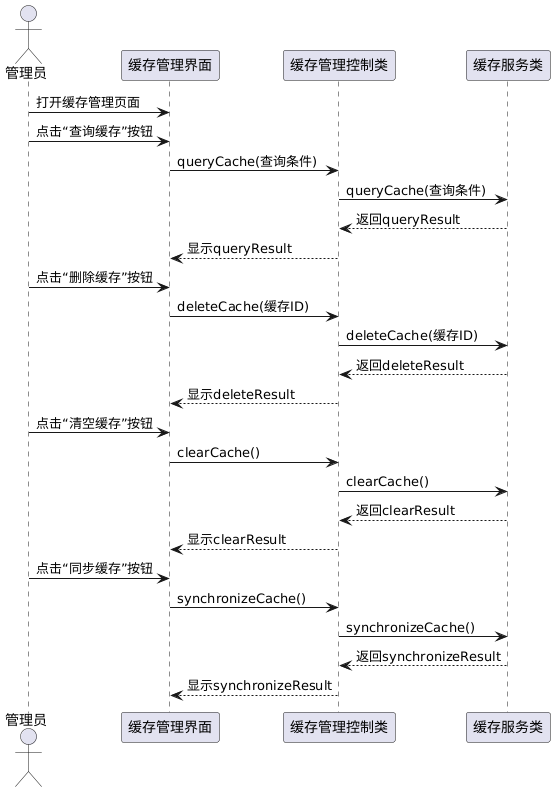
* 接收到“查看图标”请求后，调用loadIcons方法从图标库中加载图标数据。
* 图标管理控制类将加载到的图标数据返回给管理员界面。

3. 图标库（IconsLib）：

* 接收到loadIcons请求后，从系统图标库中获取所有可用的图标资源。
* 返回图标列表iconsList给图标管理控制类。

4. 异常处理：

* 如果图标加载失败（如网络异常或图标库为空），图标管理控制类将返回错误信息，并提示管理员。



**(o) 缓存管理用例实现的顺序图**

缓存管理功能允许超级管理员、系统管理员和基础数据管理员对系统的缓存进行管理，包括缓存的查询、删除、清空和同步等操作。通过缓存管理功能，用户可以优化系统的性能，确保缓存数据的准确性和一致性。

1. 管理员界面（CacheManagerUI）：

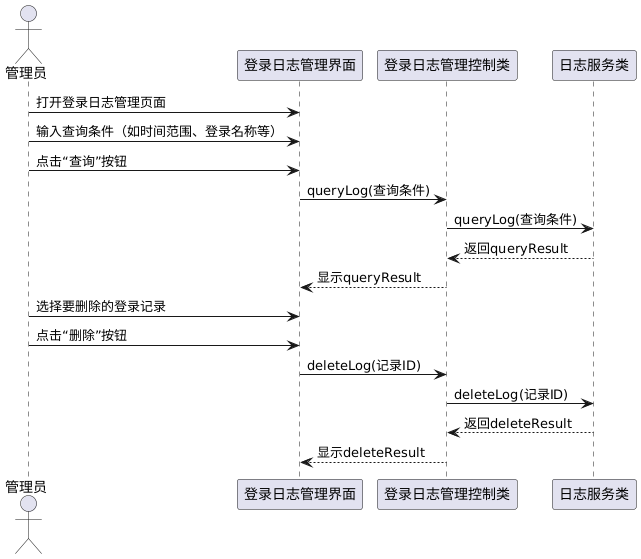
* 管理员通过系统菜单进入缓存管理界面。
* 管理员选择操作类型（查询缓存、删除缓存、清空缓存、同步缓存）并点击相应按钮：
* 点击“查询缓存”按钮，发送queryCache请求。
* 点击“删除缓存”按钮，发送deleteCache请求。
* 点击“清空缓存”按钮，发送clearCache请求。
* 点击“同步缓存”按钮，发送synchronizeCache请求。

2. 缓存管理控制类（CacheManager）：

* 接收到queryCache请求后：
* 调用queryCache方法从缓存服务中查询缓存数据。
* 返回查询结果queryResult。
* 接收到deleteCache请求后：
* 调用deleteCache方法删除指定的缓存记录。
* 返回删除结果deleteResult。
* 接收到clearCache请求后：
* 调用clearCache方法清空所有缓存。
* 返回清空结果clearResult。
* 接收到synchronizeCache请求后：
* 调用synchronizeCache方法同步缓存数据。
* 返回同步结果synchronizeResult。

3. 缓存服务类（CacheService）：

* 接收到queryCache请求后，从缓存中查询指定的缓存数据。
* 返回查询结果queryResult。
* 接收到deleteCache请求后，删除指定的缓存记录。
* 返回删除结果deleteResult。
* 接收到clearCache请求后，清空所有缓存。
* 返回清空结果clearResult。
* 接收到synchronizeCache请求后，同步缓存数据。
* 返回同步结果synchronizeResult。



**(p) 登录日志用例实现的顺序图**

登录日志功能允许超级管理员、系统管理员和基础数据管理员查看和管理用户的登录信息，包括登录时间、登录IP地址等。通过登录日志功能，管理员可以方便地进行审计和监控，及时发现异常登录行为，保障系统的安全性。

1. 管理员界面（LoginfoManagerUI）：

* 管理员通过系统菜单进入登录日志管理界面。
* 管理员可以选择操作类型（查询登录日志、删除登录记录）并点击相应按钮：
* 点击“查询”按钮，发送queryLog请求。
* 点击“删除”按钮，发送deleteLog请求。

2. 登录日志管理控制类（LoginfoManager）：

* 接收到queryLog请求后：
* 调用queryLog方法从日志服务中查询登录日志数据。
* 返回查询结果queryResult。
* 接收到deleteLog请求后：
* 调用deleteLog方法删除指定的登录记录。
* 返回删除结果deleteResult。

3. 日志服务类（LogService）：

* 接收到queryLog请求后，从日志数据库中查询符合查询条件的登录日志。
* 返回查询结果queryResult。
* 接收到deleteLog请求后，删除指定的登录记录。
* 返回删除结果deleteResult。

## 5.2系统配置

## 一、运行环境

|  |  |  |
| --- | --- | --- |
| *序号* | *项目* | *详细信息* |
| *1* | *后台软件环境* | *Windows 10* 或 *Windows Server 2012 R2* 及以上版本系统 |
| *2* | *前台软件环境* | *Windows 8* 或 *macOS 10.10* 及以上版本系统 |
| *3* | *数据库* | *Navicat* |

### 二、开发环境

|  |  |  |
| --- | --- | --- |
| *序号* | *项目* | *详细信息* |
| *1* | *操作系统* | *开发编译系统：JDK 21,*  *操作系统：windows系列* |
| *2* | *编程语言* | *JAVA、Java Javascript、HTML、CSS* |
| *3* | *编程工具* | *IntelliJ IDEA、Eclipse* |
| *4* | *网络平台* | *100M Ethernet* |

# 6子系统设计模型

*如果在体系结构模型中未对当前软件系统划分子系统，那么可省略此部分。*

## 6.1传感器监测子系统的设计模型

### 6.1.1设计类图

*描述传感器监测子系统的结构、其中各设计元素的职责及其关系。*



### 6.1.2交互图

*描述传感器监测子系统中各设计元素之间的动态协作关系。*



### 6.1.3类设计模型

*针对传感器监测子系统的设计类图中出现的每个类，逐个描述其职责、协作者、类的属性（包括属性的名称、类型、作用范围、初始值、约束条件、属性说明等）、类的方法（包括方法的名称、功能说明、参数的名称和类型、返回类型，以及可选的前提条件、出口断言、实现算法等）。*

### 6.1.4状态图（可选）

*子系统1的状态图*。

### 6.1.5活动图（可选）

*与子系统1相关的活动图。*

## 6.2子系统2的设计模型

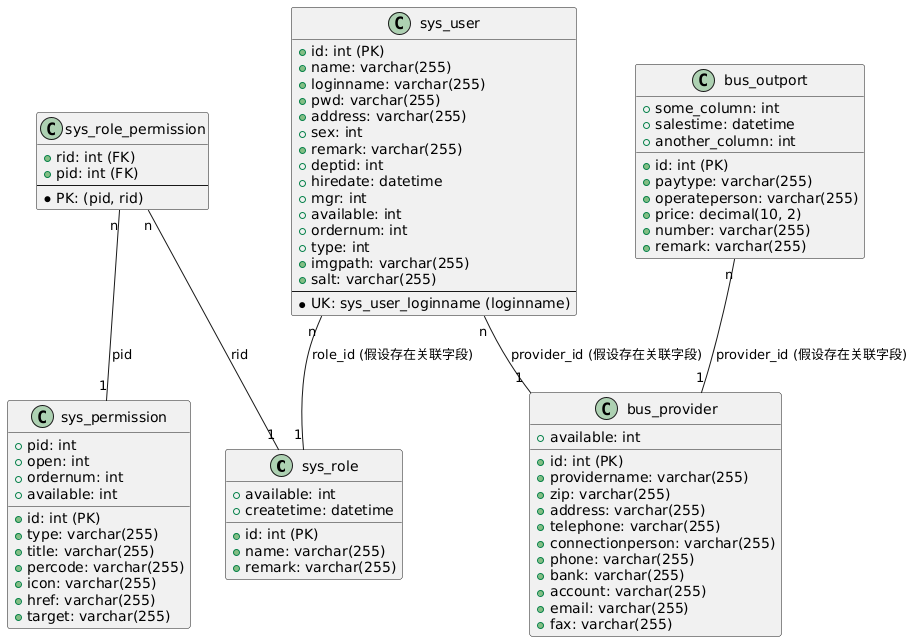
# 7.构件设计模型

*此部分的内容类似子系统设计模型部分的内容，但需要附加对构件的定制和组装机制的描述。如果当前软件系统中未包含构件，那么可省略此部分。*

无

# 8.数据设计模型

## 8.1 持久数据的组织结构



## 8.2 持久数据操作（可选）

*给出存储过程和触发器的定义。如果数据操作直接在软件代码里实现，这里就不用写。*