金融质押监管系统

软件架构文档

版本 <2.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2022/11/09 | 1.0 | 创建 | 洪湘、吕泽宇、蓝煜斌 |
| 2021/12/10 | 2.0 | 修改 | 洪湘、吕泽宇、蓝煜斌 |
|  |  |  |  |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 参考资料 4

2. 用例视图 4

3. 逻辑视图 5

3.1 概述 5

3.2 在构架方面具有重要意义的设计包 5

4. 进程视图 7

5. 部署视图 8

6. 实现视图 9

7. 技术视图 9

8. 质量属性的设计 9

8.1 性能设计 9

8.2 易用性设计 10

8.3 可修改性设计 10

软件架构文档

# 简介

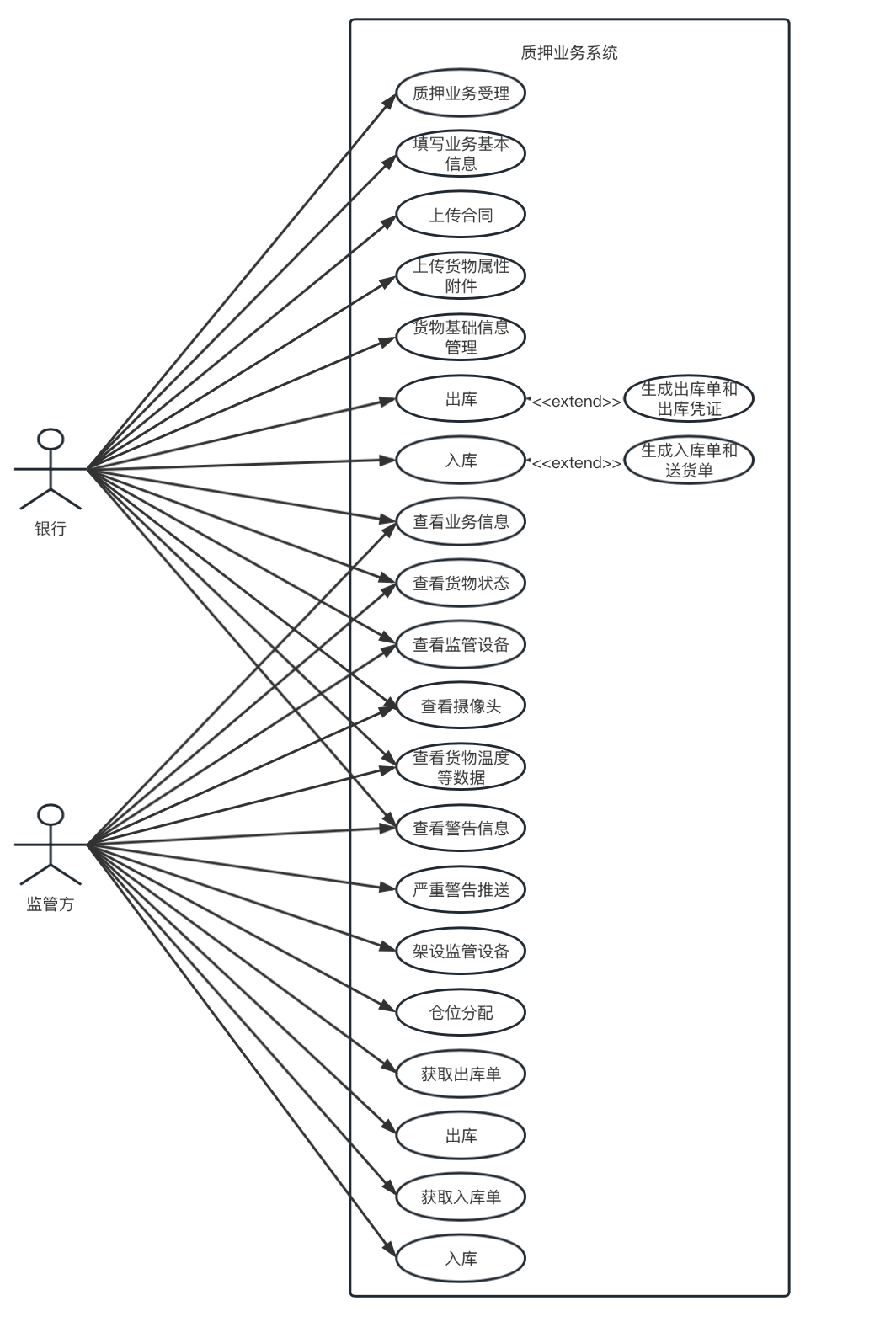
## 目的

本文档将从构架方面对系统进行综合概述，其中会使用多种不同的构架视图来描述系统的各个方面。它用于记录并表述已对系统的构架方面作出的重要决策。

## 参考资料

1. 沈备军，陈昊鹏.软件工程原理[M].北京:高等教育出版社,2013.2

# 用例视图



Actor：

银行：各个进行金融质押业务的银行

监管方：受理银行的质押业务，负责仓库管理与监管设备的安装与部署

Usecase：

质押业务受理：

银行用户选择新建业务，

系统向WMS发送进货通知，WMS根据货物以及监管要求分配库位，并反馈给系统。

填写业务基本信息：

银行用户填写业务编号、银行、借款方、监管方、借款金额、货物名称、货物属性、货物数量、监管开始时间、监管结束时间等基本信息。

上传合同：

银行用户上传借款合同、监管合同。

上传货物属性附件：

银行用户上传货物属性附件。

货物基础信息管理：

银行用户管理货物基础信息：添加货物，填写货物基本信息：货物名称、货物属性、货物长宽高、货物体积 、货物重量等；删除货物。

出库：

银行选择一单业务进行出库操作，银行下达出库指令，选择出库数量，系统生成出库单和出库凭证。

入库：

银行选择一单业务进行入库操作，下达入库指令，选择入库数量，系统生成入库单和送货单。

查看业务信息：

银行、监管方查看业务编号，银行，借款方，监管方，借款金额，仓位，货物名称，货物属性，货物数量，在库数量，监管开始时间，监管结束时间，借款合同，监管合同，其他附件，以及历史操作列表：操作名称（出库、入库），货物数量，时间，指令，相关附件等信息。

查看货物状态：

银行、监管方选择查看一单业务的货物状态（未入库、已入库、监管中、异常、已结束等），监管设备、摄像头、温度数据等。

查看监管设备：

银行、监管方查看业务的监管设备。

查看摄像头：

银行、监管方查看业务的摄像头。

查看货物温度等数据：

银行、监管方查看货物温度数据。

查看警告信息：

银行、监管方查看警告信息列表。

严重警告推送：

监管方向银行发送邮件等告知严重警告。

管理监管设备：

监管方将监管设备与业务、仓位进行绑定，查看设备绑定列表。

仓位分配：

监管方进行仓位分配。

获取出库单：

监管方获取出库单。

出库：

监管方按照出库单对货物进行出库。

获取入库单：

监管方获取入库单。

入库：

监管方按照入库单对货物进行入库

# 逻辑视图

## 概述

## 

本项目的逻辑视图采用B/S与C/S结合的架构，其中包括服务端的Server包，客户端包括UserClient和DriverClient，浏览器端包括管理员的Browser包。

## 在构架方面具有重要意义的设计包

### **Client包**

Client包是主要用于服务用户，负责完成与用户的交互任务，与Server端相连，接受用户请求，并通过网络向服务器提出请求，对数据库进行操作。

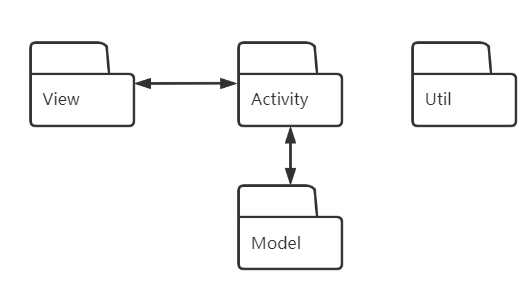
Client包主要采用MVC架构，包含Activity、View、Model、Util等包。

Activity包：Client包主要的业务逻辑，即MVC中的Controller。

View包：Client中与用户交互的界面。

Model：Client中与后台或其他模块进行通信的模块。

Util：Client的工具包。



### **Browser包**

Browser包主要用于服务管理员，负责完成与管理员的交互任务，与Server端相连，接受管理员请求，并通过网络向服务器提出请求，对数据库进行操作。

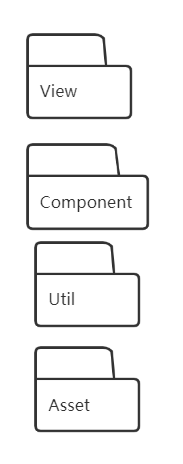
Browser包主要包括View、Component、Util、Asset等四个包。

View包：与管理员交互的界面。

Component包：Browser包中的各种组件。

Util包：Browser包中使用各种工具。

Asset包：Browser包中的静态资源包。



### **Server包**

Server包负责数据的管理，接受Client端和Browser端的请求，将数据提供给Client和Browser，Client和Browser将数据呈现给用户。

Server包采用了层次结构，分为4层，由4个主要包组成：Controller、Service、Dao和DB。

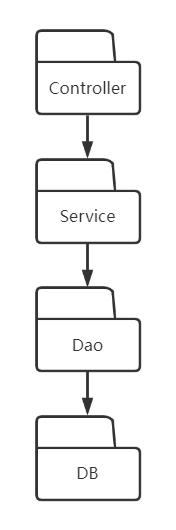
具体介绍如下：

Controller层：主要负责具体业务模块流程的控制，会调用Service层的接口来控制业务逻辑。Controller负责请求转发，接受页面过来的参数，传给Service处理，接到返回值，再传给前端。

Service层：主要负责业务模块的应用逻辑与应用设计。Service层业务实现，具体调用到已经定义的DAO的接口，封装Service层的业务逻辑有利于通用的业务逻辑的独立性和重复利用性 。

Dao (Data Access Object):数据存储对象层：主要做数据持久层的工作，负责与数据库进行联络的一些任务都封装在此

DB层：数据库层，数据存储在数据库中。

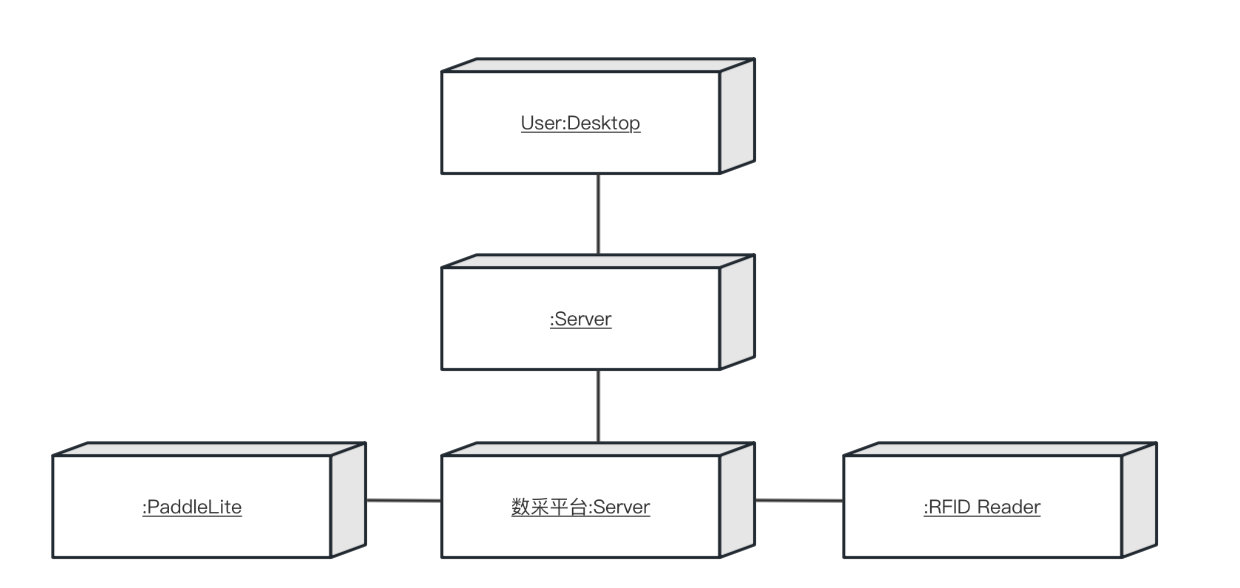


上述分层架构实现了“高内聚，低耦合”的特性，使得同一层只需要负责与上下层进行交互，降低了设计的复杂度。同时使用Spring框架，这是一个轻量级的控制反转(IoC)和面向切面(AOP)的容器框架，也是面向接口的编程，由容器控制程序之间的依赖关系，而非传统实现中，由程序代码直接操控。（依赖）控制权由应用代码中转到了外部容器，控制权的转移，是所谓反转。依赖注入，即组件之间的依赖关系由容器在运行期决定，即由容器动态地将某种依赖关系注入到组件之中，也能起到解耦的作用。

# 进程视图

本项目中的进程均为框架本身创建，不涉及自己创建的进程，因此不需要进程视图。

# 部署视图



## User: Desktop(电脑web端)

银行、监管方用来管理业务、监管设备的客户端。（使用电脑网页端）

## :Server

业务质押系统、WMS系统的服务器，银行、监管方均连接到该服务器获取数据。

## 数采平台: Server

管理RFID读写器、摄像头及其上传数据的系统。

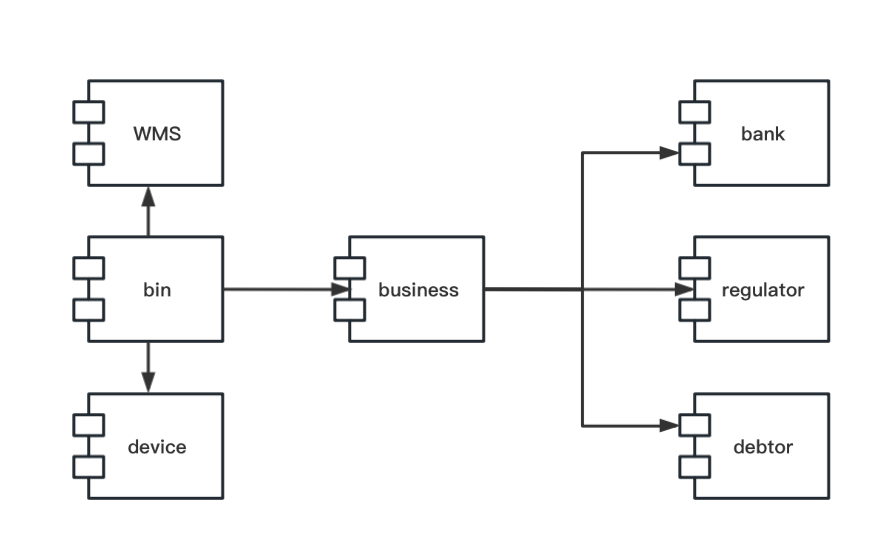
## :RFID Reader

读RFID温度标签的硬件。

## ：PaddleLite

边缘端对摄像头拍照内容进行识别的开发板。

# 实现视图



WMS：仓库

bin：库位，每个库位都有所属仓库及种类

device：设备，每个库位和业务绑定的设备

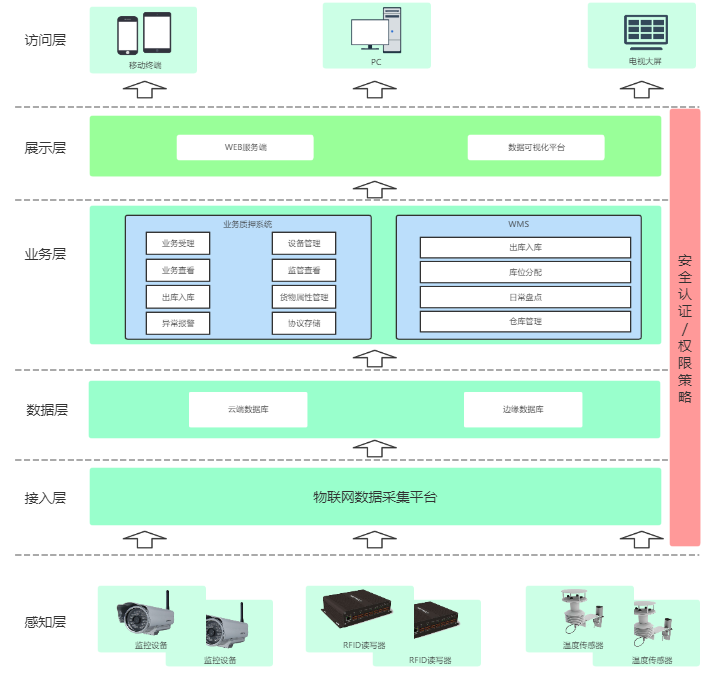
business：业务，每单业务都有银行、监管方、借款方和仓位

bank：银行用户

regulator：监管方用户

debtor：借款方用户

# 技术视图



感知层提供摄像头、温度传感器、RFID读写器等监控设备。

接入层在物联网数据采集平台对监控设备进行注册管理，并进行数据的接收与转发。

数据层使用云端数据库和边缘数据库。

业务层实现业务质押系统和WMS，使用JavaScript语音进行开发，后端采用Spring Boot后端框架，前端使用React框架，开发工具使用Intellij IDEA。

展示层分为WEB服务器和数据可视化平台。

访问层包括移动终端、PC和电视大屏。

# 质量属性的设计

## 性能设计

根据本项目的非功能性需求中的1000个并发用户，最长响应时间为3秒（不包括网络延迟），所以我们首先聚焦于性能战术。

在设计关系型数据库时，尽量满足常见的设计范式，减少数据冗余，提高数据查询效率。如果性能依旧无法达到要求，我们将采用分表的方式，限制每张表的大小，以此减少从大表中查找和读出数据的时间。同时可以将热门属性与冷门属性分离，从而快速取出某些属性，达到性能的提升。

并且我们最大限度的简化了系统的逻辑层次，将一部分数据处理工作分配至客户端以提升执行效率和服务器的并发处理能力。

## 易用性设计

我们将用户接口分离，限制可能出现的用户接口的频繁变化所造成的影响。

在用户帮助上，我们采用第一次使用时进行新手引导，以及后续在软件中查看带操作步骤图解的帮助文档来帮助用户使用我们的软件。  
 在界面设计上，我们秉承以人为本的原则为用户提供尽量简洁的页面和便捷的操作。采用柔和的对比色调防止视觉疲劳。顾客搜索商品时，在焦点自动定位到输入框的同时，提示历史搜索记录方便用户使用。温度超出设定范围时自动提醒也为司机用户提供了便捷的服务。

## 可修改性设计

首先我们对于后端进行分层，并将接口与实现分离，从而使得不同层解耦，以减小系统中模块修改引起的成本变化。

同时我们将一些基本不变的基础信息采用公有化的模块来处理，对于重要的关键信息采用私有的模块来处理，模块之间采用抽象化的共有接口来进行联系，这样在修改某个模块时就能防止代码污染扩散。