



**本科毕业设计（论文）**

**面向个体小商户基于RFID的智能化仓存管理系统**

**学 院 艺术与设计学院**

**专 业 数字媒体技术**

**年级班别 2016级2班**

**学 号 3116007889**

**学生姓名 彭梓浩**

**指导教师 韦宇炜**

**2020年 05 月**

**面向个体小商户基于RFID的智能化仓存管理系统**

**彭梓浩**

**艺术与设计学院**

**摘 要**

跟着大社会经济发展增速日益提升和大众生活尺度规范标准的日益进步，我们的日常生存环境的进行和社会的正常运作对万物智能化的要求变得愈来愈高。自动化和智能化的刚需已经体现在生活中的方方面面，比如智能家居的主动开关和环境检测，基于用户画像的内容推荐以及仓库存量的智能化管理复合软件系统。

传统的仓库管理首要采用人力和笔纸的录入来对仓库的存储状况，货物进出来进行信息的记录和处理，这样的方式在以往的通讯效率和交易需求之下还勉强能够满足平时的运营需求。但是随着电子商务平台的发展和移动数据运输能力的爆炸式增长，那样的传统管理方式已经远远落后于目前的日常运营所需。于是诞生了基于RFID或其他射频识别协议的通讯方式来开发的自动化仓库管理系统，率先应用于如集装箱和大型仓库管理的企业级系统中。但其中运用到的企业级物联网设备的成本和复杂的信息管理系统让大多数依然有自动化管理需求的个体小商户望而却步。

本设计要通过汲取企业级的仓库管理系统的设计逻辑并且针对个体小商户的需求特质进行本地化的改良来设计开发的基于RFID的智能化仓存管理系统。产品的主要平台在利用了基于H5技术栈的微信小程序以提供更方便的入口以及结合低成本的可穿戴RFID射频识别设备来解决个体小商户的需求痛点。

**关键词：**Canvas，数据可视化，微信小程序，RFID，仓库管理系统

**Abstract**

With the increasing growth of large socio-economic development and the increasing progress of the standard of living standards for the masses, the requirements of the intelligentization of all things in the conduct of our daily living environment and the normal operation of society have become higher. The needs of automation and intelligence have been reflected in all aspects of life, such as active switching and environmental detection of smart homes, content recommendation based on user portraits, and intelligent management complex software systems for warehouse inventory.

The traditional warehouse management mainly uses human and pen entries to record the storage status of the warehouse, and information is recorded and processed in and out of the goods. This method can barely meet the usual operating requirements under the previous communication efficiency and transaction requirements. . But with the development of e-commerce platforms and the explosive growth of mobile data transport capabilities, that traditional management approach has lagged far behind the current daily operations. Therefore, an automated warehouse management system developed based on the communication method of RFID or other radio frequency identification protocols was born, and it was the first to be used in enterprise-level systems such as container and large warehouse management. However, the cost of enterprise-level IoT equipment and the complex information management system used in it make most individual small businesses that still have automated management requirements prohibitive.

This design is to design and develop an intelligent warehouse management system based on RFID by absorbing the design logic of an enterprise-level warehouse management system and making localized improvements to the needs and characteristics of individual small businesses. The main platform of the product uses the WeChat applet based on the H5 technology stack to provide a more convenient entry point and combines low-cost wearable RFID radio frequency identification devices to solve the pain points of individual small businesses.

Keywords: Canvas, data visualization, WeChat applet, RFID, warehouse management system

**目 录**

[1. 绪论 1](#_Toc34572271)

[1.1 课题简介 1](#_Toc34572272)

[1.2 课题研究背景以及意义 1](#_Toc34572273)

[1.3 课题核心研究内容 2](#_Toc34572274)

[2. 相关技术研究 3](#_Toc34572275)

[2.1 案例分析 3](#_Toc34572276)

[2.2 产品需求分析 4](#_Toc34572277)

[2.3 产品功能模块设计 6](#_Toc34572278)

[2.4 产品技术选型 9](#_Toc34572279)

[3. 产品开发和测试 12](#_Toc34572280)

[3.1 数据库设计 12](#_Toc34572281)

[3.2 后端接口设计与开发 17](#_Toc34572282)

[3.3 前端设计和开发 28](#_Toc34572283)

[结论 43](#_Toc34572284)

[参考文献 44](#_Toc34572285)

[致谢 45](#_Toc34572286)

1. 绪论

1.1 课题简介

本课题是应用了基于RFID的物联网技术栈结合现代化的轻量级移动web应用微信小程序开发的管理系统。在产品的设计过程中充分地理解企业级的仓存管理系统的设计需求逻辑如货物的进出原则，实时的数量监测和临界值报警等等，并且结合当代的中小型个体小商户对仓库管理特型需求比如种类繁多，进出库过程较密集而短促以及缺少一个宏观的数据统计等等来对程序的用户体验做了整体的优化提升。

1.2 课题研究背景以及意义

作为制造业强国和商品进出口的世界第一大国，仓库的高效运转是每一家中国企业在整个产品供应链中最为重要的一环。仓库的稳定管理，全部都可以维护得好的运行，是非常考验人的，作为管理者他要有很高的小智慧，通用的大知识，然后去选取一个厉害的适合他自己业务的系统来管理整个项目。

经过总结，我发现WMS主要从三个方面对传统的仓库管理方案进行了革新：数据的记录，数据的传输，货物的操控。

数据的记录：以往的货物的数量和状态，进出库的信息主要是通过手工记账也就是它的媒介是纸张，那么使用纸张就有着难以长期保存和记录速度缓慢的硬缺点。在货物的数量级增长的情况下必然会面对数据记录太慢，人工校对出错，信息互通不到位等等的缺陷情况，这些场景的出现最终就会导致货物物资的损失和账单的不对称使得整个供应链失去了正常运作的能力。信息化的仓库管理系统解决方案从此一发不可收拾。

数据的传输：由于传统的数据记录大多通过笔纸记录，其数据的传输实质也是实际的物流传输，到了信息化的早期，传真机的出现为信息的传输提供了巨大的方便，但是离不开纸张的信息记录介质。到了20世纪80年代，第三代计算机网络的规范被慢慢定义下来，同时由于社会经济发展的强烈需求，计算机网络的民用化需求日益强烈。开发者可以使用C,Java,PHP等等编程语言构建简单的http服务应用，使得数据可以在互联网中安全且高效地传输。随着Node.js的发展，开发者可以使用JavaScript 开发更加轻量化的http服务和Web应用。

货物的操作：传统对货物的操作和校验主要通过人工的手动工作来实现，这种方法最大缺点是效率极低，误差率高以及人力的成本较高。大致的过程是仓库管理人员根据客户提出的需求来手工填写各种票据并且送到机房输入和物流人员校验货物，但货物流向和质量没有一个可靠的方式可以持续监控。后来便出现了条形码管理系统，它的本质是通过制定一个编码规则对每一件货物进行编码以便在数据的层面对每一件货物进行追踪和识别。而纯数字的录入对于机器的读写是有困难的，于是条形码的利用宽度不等的黑条和空白来映射不同的数字来供给机器使用红外或摄像机来识别，再通过与数据库的信息交换结合由业务驱动的机械设计可以在极短的时间完成大量的货物运输操作。但是条形码始终是需要一个识别的过程以及较短的距离来进行数据的读取，而且其本质还是基于纸张和墨水的记录来对物品进行识别，依然有保存稳定性的不足和被篡改的风险。然而基于射频识别技术的RFID能够做到读取快，精度高，难篡改，低成本的优点，非常适合应用于仓库管理系统的开发实现上。

1.3 课题核心研究内容

本课题研究的主要内容是结合RFID（Radio Frequency Identification）也就是射频识别技术开发一款智能化的仓存管理系统，并且它的整个使用流程需要贴合用户使用习惯，简化流程，专注于个体的小商户。本次设计与开发将会从流行的仓库管理系统中学习其设计的理念和产品使用的逻辑，并且将其优劣特质解耦，去掉个体小商户所不需要的过剩功能，增加他们特殊需求的功能。从开发的扩展性，数据的稳定性以及对业务的需求层面上从流行的数据库软件Redis，Mongo，MySQL中选择合适的解决方案和最佳实践。从技术栈和运行的性能上从Node.js的后端框架Express，Egg中选择适合的后端解决方案。从ColorUI，Vant-weapp，iView中选择适合仓库管理系统业务的微信小程序UI库以提供高效率的开发体验和用户体验。最终能够开发出能够击中实际的仓存管理业务痛点的微信小程序，并且能够总结出对选取的技术栈的开发经验以及所有的不足并对程序的未来和个人技术的未来做出一个合理的规划。

2. 相关技术研究

2.1 案例分析

本案例分析旨在从成功的现代化仓库管理系统中分析其产品设计的逻辑和主要解决的需求痛点。

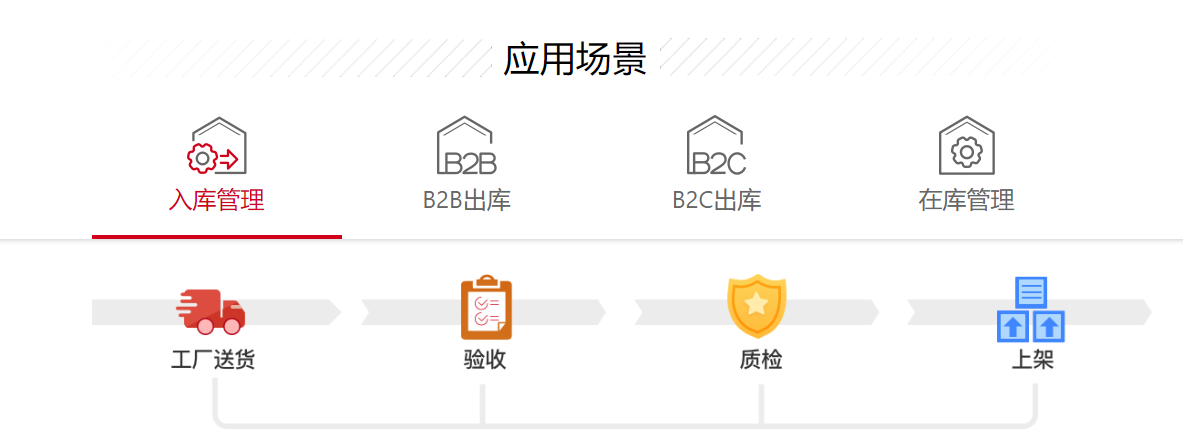


天权仓库管理系统，是京东物流供应链产业平台的一个子系统，主要的功能模块包括掌控总体数据走向的全局运营管理，紧贴业务数据的基础数据管理，多模式入库状态的入库管理，可调整组装的库内管理，包含多层复检的出库管理以及可自定义规则的报表管理。

作为京东物流下的供应链产业平台的一个重要分支系统，天权在满足基础的仓库状态管理以及出入库任务功能之外还提供了对库存状况的云计算分析出入库出库分装拆散等等业务的最佳策略，为复杂的实际业务提供辅助。



在入库管理方法上该系统从工厂送货，验收，质检到上架都有相应的数据功能和异常处理。实现了基于RFID的一键验收，适用于全品类的商品快速入库并且通过比对数据库中的规则以确保入库质量并且支持分批次验收和灵活管理验收方式，并且配套有办公移动化的小程序和app辅助物流师工作，支持自定义的条码贴合业务的识别处理，在异常情况出现时会进入异常处理的程序分支单独管理；在B2B业务中有对于大货量程序适配主要的流程是波次安排，波次下发，生成拣货任务，多人拣货，装箱复核，出库操作。系统从波次生成的策略，拣货任务生成策略，装箱集货策略，多级包装收发货和箱码扫描识别都提供有功能支持；在B2C业务中更注重与快递合作的大并发量的任务稳定性优化，主要的功能包括订单分配，拣货路径规则，分拣，快递/承运商交接，打包发货，复核，出库异常处理的功能，其中拣货路径规则能通过对物流基础的相关配置和定位规则的设置来进行多种拣货路径的规划；在库的管理中能对验收情况，上架情况，库内查询，出库查询，业绩汇总，异常查询以及盘点结果做到实时的监控和统计。



天权作为一款企业级的仓库管理系统，拥有大量细粒度的功能以及京东提供的云计算带来的最佳策略支持。这些功能在京东物流的用户量下发挥出了它极大的优势和提供了优秀的服务，但是同时也会带来更高的使用成本和维护成本，对于个体小商户而言更是造成了价格过高而功能又过剩的情况。其中RFID一键验收，转残少货异常处理，货量临界值预警，多维度订单分配，出入库单复核以及业绩结果盘点是值得保留的核心功能，而其他的基于云计算的功能如灵活多维度组波策略，拣货路径规划等等的功能可以忽略，以达到适合个体小商户最佳的产品性价比。

2.2 产品需求分析

通过分析成功案例以及对个体小商户的仓存管理需求的考究，我可以了解到如淘宝的商户中，他们货品的出入库频率比一般的商户会更高，通过人手的记录和计算就会出现效率过低以及误差率高的情况，所以根据现有的技术，本产品选取最为通用的RFID射频识别技术以及条形码识别来完成一个高效率的自动化商品识别功能。

产品功能需求包括：

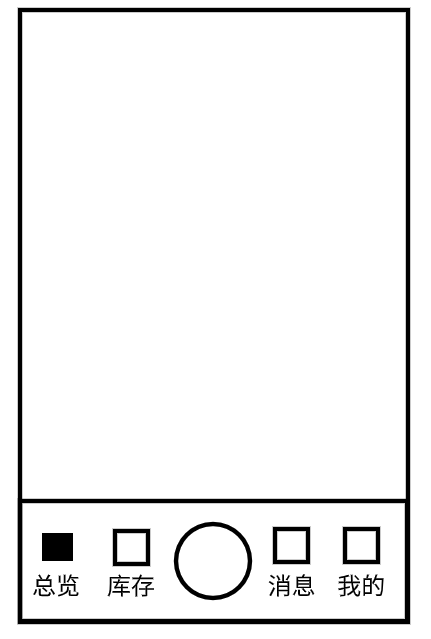
1. 商品录入功能
   1. 能够录入商品名称
   2. 能够录入商品价格
   3. 能够自定义商品的类型选项并记录商品类型
   4. 能够记录商品的库存数量
   5. 能够存储商品对应的图片信息
   6. 能够记录用于该商品的识别码或编号包括
      1. 唯一id号
      2. 条形码
      3. RFID的唯一识别码
   7. 能够对商品所属的仓库做关系型的逻辑存储
2. 商品信息读取功能
   1. 可以通过手机扫描商品的条形码识别到商品的详情信息
   2. 可以通过RFID识别查询到商品的详情信息
3. 成员管理模块
   1. 按照功能和状态给予用户不同的角色以及权限，包括：
      1. 普通用户：未加入任何仓库，可以加入或创建仓库
      2. 仓库创建者：拥有仓库管理的最高权限，可以修改仓库元数据，可以邀请成员加入
      3. 仓库管理员：拥有仓库管理的二级权限，可以修改所属子仓库的元数据或指派任务
      4. 仓库员工：拥有仓库管理的三级权限，不可修改仓库元数据，只可接受或创建任务
   2. 成员加入其他仓库只能通过仓库的创建者或管理员通过生成唯一邀请码来加入相应的仓库实例，仓库的所有信息只对仓库的成员可见。
   3. 所有成员有对比自己低权限的员工做权限修改，移除成员的权限
   4. 用户通过微信授权得到的openid作为用户的身份的唯一识别码
4. 进出库单的定制化生成，信息包括：
   1. 进出库任务的生成日期时间
   2. 预期的进出货商品清单
   3. 实际的进出货商品清单
   4. 指定的进出货任务负责人
   5. 进出库任务的完成状态包括：
      1. 新任务
      2. 进行中
      3. 已完成
      4. 意外终止
   6. 进出库单包含商品的所属仓库
   7. 进出库任务的生成能对仓库相关的工作人员有提醒
5. 可以查看仓库内所有库存商品物品的信息包括：
   1. 商品的名称
   2. 商品的添加事件
   3. 商品的描述
   4. 商品的图片
   5. 商品的所有进出库记录
6. RFID操作管理模块
   1. 可以连接RFID读写设备
   2. 可以与RFID读写设备通讯以满足业务需求
   3. 可以管理包括查看编辑RFID读写设备的状态

以上是基于对现有的仓库管理系统所包含的产品功能以及针对个体小商户的特点进行删减和添加之后确定的产品的基本需求，接下来的工作是根据所列出来的产品基本需求来进行抽象划分产品不同页面的功能模块设计。

2.3 产品功能模块设计

根据以上的需求分析主要分化出以下的功能模块区域：

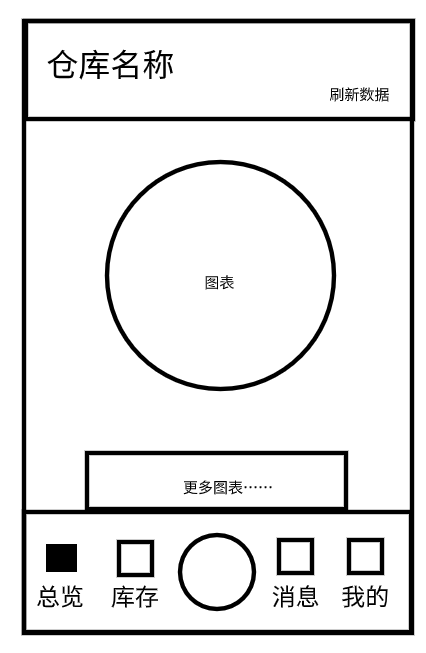
1. 仓库数据统计总览
2. 库存货物信息列表
3. 仓库进出库任务列表
4. 个人信息展示和编辑页面

为了给以上四个信息含量较高而且密集的页面模块在产品中有清晰的指引和入口，产品使用置底的导航栏Tab条为以上页面作为入口。

Tab栏从左至右分别为：总览，库存，消息，我的。

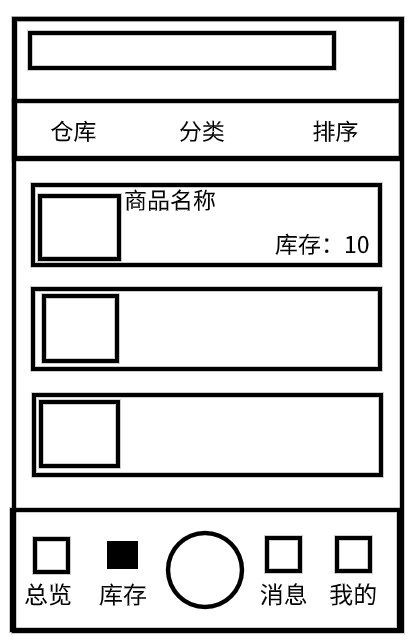
因为本产品加入RFID识别以支持物联网的功能，并且该功能的逻辑与其他功能不产生直接的闭环，却有非常高的日常使用率，所以把该功能的入口置于Tab栏的中央以方便用户进入，同时还能起到美化用户界面的功能的。

设计的逻辑原型图如右图所示。

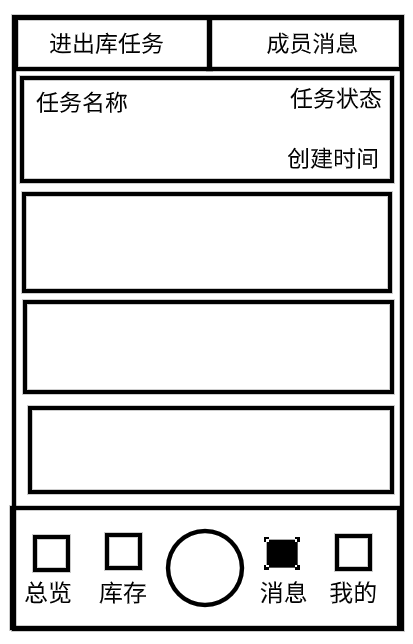


总览页面需要包含对仓库数据的可视化统计，可以使用条形图，饼图，折线图等等的可视化图表方式对仓库的商品数量，进出库数量，商品种类的比例进行统计并呈现出来。页面的头部需要包含所属仓库的名称以及一个刷新数据并重新统计的按钮。

设计的逻辑原型图如右图所示。

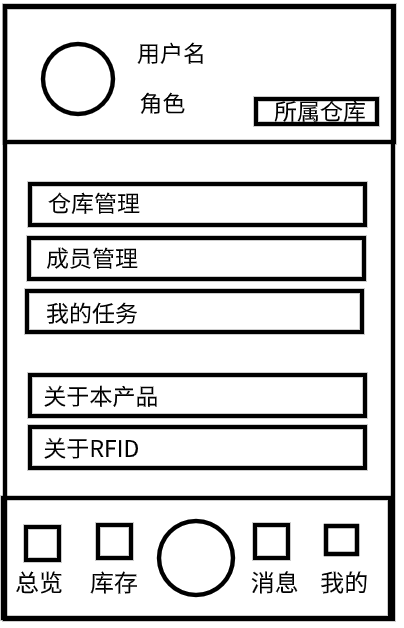
库存页面需要展示一个商品列表包含用户所属仓库的所有商品，包括商品的名称所属子仓库，图片，时间以及库存，在这个页面可以直观地看到商品的具体库存，点解商品的栏目还可以看到商品的详情页面。并且此页面模块的头部还应该有一个用于搜索和筛选目标商品的工具以更快捷地找到用户需要查看的商品。

设计的逻辑原型图如右图所示。



消息页面需要包含两部分的消息内容，包括进出库相关的任务信息和成员的加入推出相关的消息。在进出库相关的任务消息中可以点击看到该任务的详情信息包括进度信息。

设计的逻辑原型图如右图所示

我的页面需要包含登录用户的基础信息包括用户名，头像，用户的权限（角色），用户所属的仓库名称。以及还需要包含各种管理功能页面的入口包括：仓库管理，成员管理，我的任务。还需要包含对本产品的技术帮助和使用指南。

设计的逻辑原型图如右图所示。

啊大大

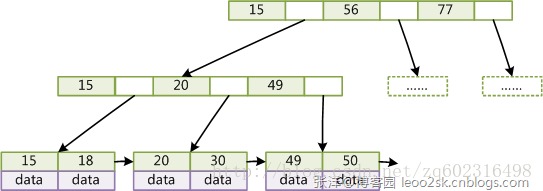
2.4 产品技术选型

1. 数据库技术选型：

关系型数据库(SQL) VS 非关系型数据库(NoSQL)

关系型数据库非常常用，在我的实习期间，我使用它完成了好几种不同的项目包括图片识别系统，测试机管理系统等等，他的实用性实在是令我难以忘怀。

关系型数据库拥有表格状的数据规划，index是他的一种索引技术，用于帮助这个数据库获取数据，且非常高效的一种算法。MySQL使用B+树，叶子节点中存放实际的数据，而其他节点只存储和数据对应的key。



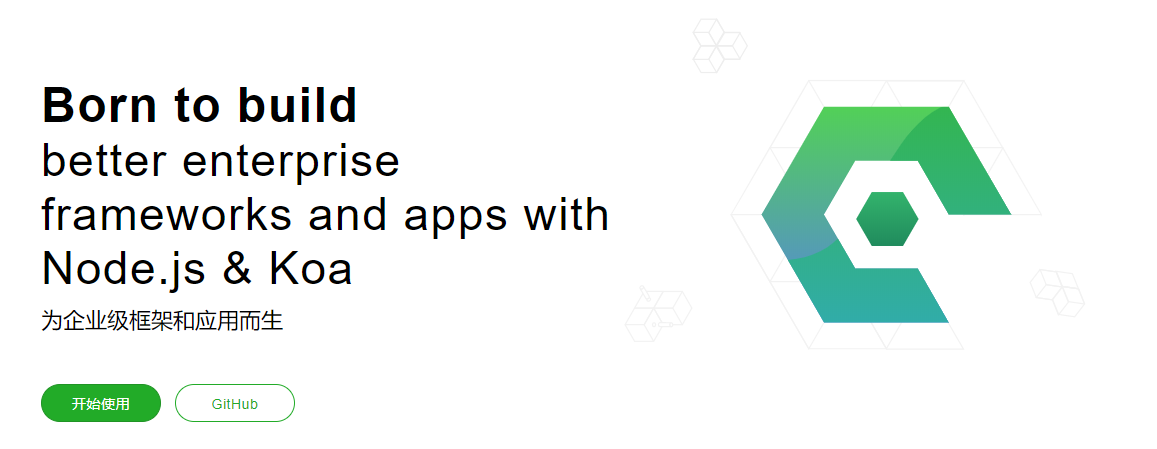
InnoDB对B+树也做了优化，如图所示，在B+树的每一个子节点中额外加重了一个横向的相邻指针使得遍历可以有顺序地访问指针的的B+树。这样可以优化区间访问的性能。MySQL的性能优化以及事务一致性使得他有不错的读写性能表现。关系型数据库拥有ACID的特性即原子性，一致性，隔离性，耐久性。意味着数据的存储不会随着时间的转移而发生丢失或改变，也不会因为空间位置的转移而发生差异，提供了非常好的稳定性，给事务运行的稳定状态提供了必要的支持。在开始使用之前需要严格定义表格规划包括字段名，字段保存的数据类型，数据递增的规定等等，其结构的稳定使得不同的表可以通过必然的设计使其产生“瓜葛”。

所以到了到底之后我得出的结论是要在合适的场景选用合适的数据库，在这个仓库管理系统的复合开发中，我认为存储与业务紧密相关的数据如商品数据，消息数据，用户数据等等这些有较强的关系约束的数据使用MySQL数据库存储。而为登录过程提供校验的cookie-session则可以使用Redis数据库存储。但鉴于为个体小商户而设的仓库管理系统这一定位，关于session的存储使用后端的server服务器内存存储已经可以满足一切的业务所需。

2. 后端技术选型

后端技术作为数据的传输者，产品功能的核心支持，有着非常庞大的技术堆栈结构以及侧重于各种各样业务的优化而诞生的不同框架。使用Python编写的Django后端框架，高度的可定制化和扩展能力，同时需要较高的学习成本；基于大名鼎鼎的Java开发的Spring Boot框架，有着特意为使用云方法的大型引用程序构建的高度目的，有非常高的文档支持程度。以上的框架都是非常优秀的企业级框架，能够满足复杂的业务需求。而这次我选择的是基于JavaScript语言的Egg框架，基于Koa，由阿里巴巴团队开发并开源。



JavaScript的诞生是因为以前的表单提交总是需要在php里面写复杂的检验代码并且前后端混合导致结构非常的臃肿使得项目的维护性差，扩展性低，于是就发明了JS这一门弱类型解释性脚本语言，使得它可以直接运行在浏览器端并提供表单验证的功能，使前后端做到第一次的逻辑分离。于是JavaScript因为其学习成本较低且无需编译的特点使得其在浏览器端发展越来越快，于是一个叫做Node.js的使用非阻塞I/O事件模型的JavaScript运行平台诞生了。由于JavaScript语言本身基因的特点，Node.js作为服务端应用十分适用于高并发的I/O业务请求，并且非常适用于开发者方便地搭建响应速度快又易于扩展的后端服务应用。

在仓库管理系统中，后端所面临的最大挑战是稳定的高并发查询和插入操纵，整个业务流程不涉及复杂的数据运算，纯I/O的业务且高扩充性的需求使得Node.js非常适用。

Egg作为由阿里维护的后端框架，在github上的贡献者超过一百人，专注于企业级框架和应用而生，开发者可以基于egg再构造出适合业务和团队的上层框架，大大降低学习和维护的成本。整个框架的使用遵循微核心+插件化的机制，使得要扩展功能的难度大大降低。

3. 前端技术选型

因为是开发微信小程序，开发技术就是使用又微信提供封装的WXML+WXSS+JS作为开发语言。有流行的跨平台开发框架可以打包成微信小程序来“曲线救国”，例如uniapp，但是我在其社区了解之后发现它存在的坑还是十分的多，而且本次产品只有微信小程序一个终端，所以直接选用原生开发。另外，使用适合的UI框架会大大提升开发效率，我从Vant-weapp，iView，ColorUI，WeUI，四个主流的微信小程序UI库中挑选最适合的UI库。Vant-weapp由有赞维护，有非常丰富UI组件如选择器Picker，图表Icon，更重要的是它的默认样式比其他三者都适合开发后台管理系统。而WeUI虽然由微信开发，但是已经停止维护，而且它所包含的组件比较简陋，封装度不高，主要的卖点是贴合微信的设计语言，所以最终选用的UI库的Vant-weapp。



3. 产品开发和测试

3.1 数据库设计

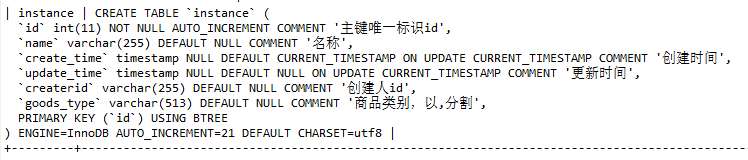
数据库的表结构主要通过不同表记录不同数据并通过id给表数据之间产生关联一对多，多对一，多对多的关系。

为了做到一次开发足以让多个仓库使用，字段逻辑的关系设计需要足够抽象，只需要把通用的逻辑字段记录在表格，比如物品的进库出库行为，物品的名称，库存，价格等等，而无需关心如颜色，重量，形状等更加贴合业务的字段，以更好的兼容各种各样的实际业务。一个个人仓库在数据库中的表现是一个仓库实例。一个人只能创建一个实例，但是一个实例可以包含多个子仓库。例如：小明名下的智能超市有三家分店，他要给他的仓库使用智能仓库系统，就可以创建一个实例名为智能超市，然后创建三个子仓库分别对应三个分店的仓库管理情况。

一个实例需要包含的字段信息有：

1. id：自动递增主键，作为实例的唯一标识
2. name： 名称，用户可以在创建的时候给实例创建名称，后期可以修改。
3. create\_time：创建时间，timestamp，在数据被insert的时候自动填入
4. update\_time：实例信息更新的时间，timestamp
5. createrid：填入实例创建用户的唯一标识id
6. goods\_type：实例包含的商品类别，以逗号分隔。

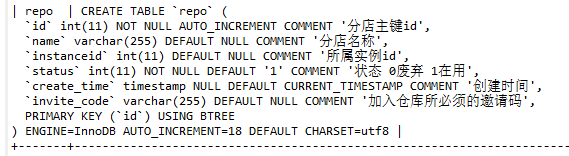
Instance表格的创建SQL：



一个子仓库需要包含的字段信息有：

1. id：自动递增主键，作为子仓库的唯一标识
2. name：名称，用户可以在创建的时候给子仓库创建名称，后期可以修改。
3. create\_time：创建时间，timestamp，在数据被insert的时候自动填入
4. update\_time：仓库信息更新的时间，timestamp
5. instanceid：所属实例的id，仓库和实例的关系是多对一的关系。
6. invite\_code：其他用户想要加入仓库必须的邀请码，由仓库创建者通过接口生成

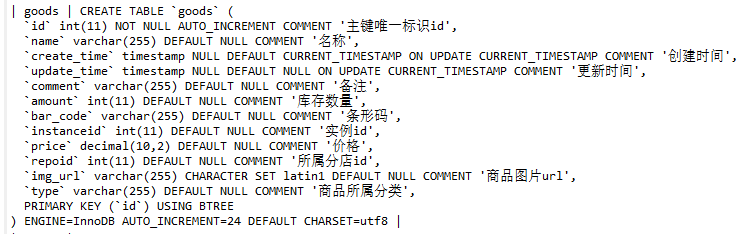
repo表格的创建SQL：



一个商品需要包含的字段信息：

1. id：自动递增主键，作为商品的唯一标识
2. name：商品的名称
3. create\_time：创建时间timestamp
4. update\_time：更新时间timestamp
5. comment：备注
6. amount：商品目前的实时数量
7. bar\_code：商品的条形码信息
8. instanceid：商品所属的实例id
9. pirce：商品的价格
10. repoid：商品所属的子仓库id
11. img\_url：商品的图片url
12. type：商品的类型，数据来源自instance的goods\_type

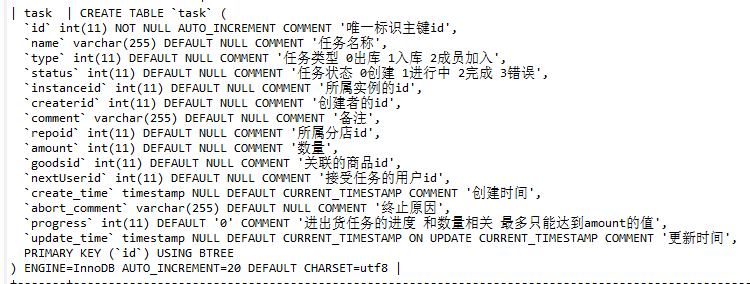
goods表格的创建SQL：



一个消息（任务）需要包含的字段信息：

1. id：自动递增唯一主键
2. name：任务名称
3. type：任务类型，0出库，1入库，2成员出入
4. status：任务状态，0创建，1进行中，2完成，3出错
5. instanceid：任务所属的实例id
6. creater\_id：创建任务的用户id
7. comment：任务的备注
8. repoid：任务所属的子仓库id
9. amount：进出库任务的商品目标数量
10. goodsid：进出库任务的商品id
11. nextUserid：指定对任务进行操作的用户id，或实际最后一个对任务进行操作的用户id
12. create\_time：任务创建的时间timestamp
13. abort\_comment：任务被强制终止的原因
14. progress：进出库任务的进度，对应amount，最大不可超过amount
15. update\_time：任务更新的时间

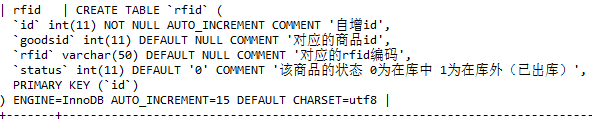
task表的创建SQL：



一个RFID识别标签所需包含的字段信息：

1. id：自动递增主键
2. goodsid：该标签所属的商品的id
3. rfid：该识别标签的唯一EPC码
4. status：该识别标签的状态，0为在库中，1为在库外

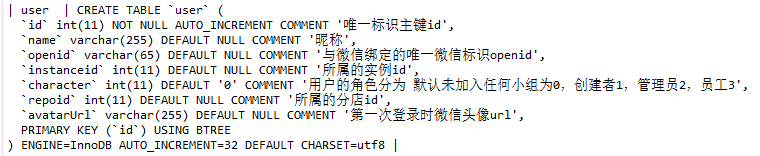
rfid表的创建SQL：



一个用户user所需包含的字段信息：

1. id：自动递增主键
2. name：用户的昵称，默认为微信昵称
3. openid：用户微信登录的唯一识别码
4. instanceid：用户所属的实例id
5. character：用户的角色，0为初始用户，1为创建者，2为管理员，3为普通员工
6. repoid：用户所属的子仓库id
7. avatarUrl：用户的头像url

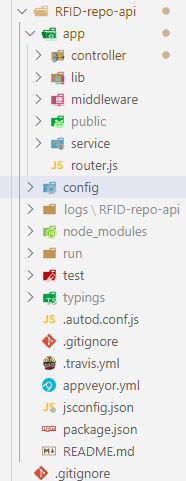
user表的创建SQL：



3.2 后端接口设计与开发

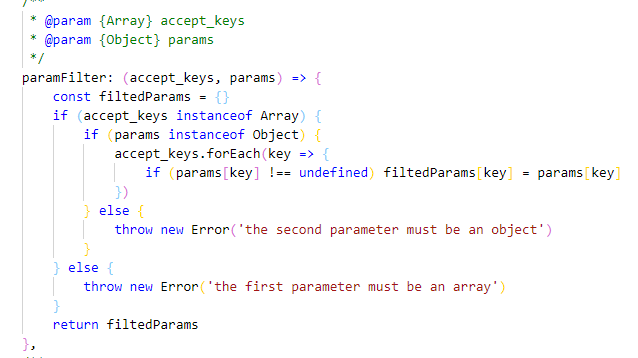
后端的核心功能主要通过对不同表的查询，插入，修改，删除来组合完成。

使用Egg.js 开发后端服务的好处就是无需太多顾虑代码的结构，遵循MVC结构，在controller层控制参数的过滤，在service层做实际的业务逻辑，比如对数据库的操作等等。接口的模块根据数据库的表而分开，一个controller或一个service只控制一个表，使得逻辑更加统一，代码简洁，方便维护调试和扩展。接口风格遵循RESTful架构，可以概括为资源，统一接口，URI和无状态。



项目的结构如右图所示，app/下为所有的主要代码，controller/文件夹为控制层，主要作用是过滤参数，验证权限，处理参数，再继续下一步的操作。Service/文件夹是服务层，用于实际的业务逻辑如数据库的操作等等。Lib/文件夹是封装好的通用函数以提高开发效率。Middleware/文件夹中间件，巧用中间件可以让接口的功能封装得更加简洁，把通用的功能如权限验证写成中间件。Public文件夹是公共文件夹，开放权限所有人都可访问。Router.js配置所有暴露的接口。Config/是配置文件夹分为默认配置config.default.js，开发配置config.dev.js，生产配置config.prod.js，以及配置插件的入口plugin.js。

为了后端服务的稳定性和数据库的安全性，我们需要在controller过滤参数，以接受接口所需的参数并过滤一切额外的参数，于是封装了一个函数专门用于过滤后端收到的json格式的参数paramFilter



调用的时候只需要在controller层使用js的模块化功能require即可，传入所需的参数和收到的参数。

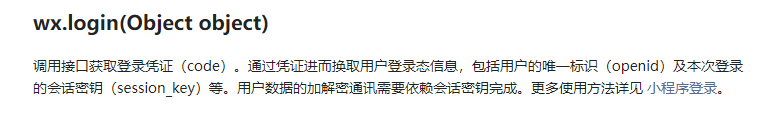


1. **实现用户的注册登录**

因为基于微信小程序，采取流行的静默注册方式进行登录，调用接口之后检查数据库中有无当前用户，如果没有的话自动注册并返回登录成功。



整个流程是：用户在微信小程序通过点击登录按钮并同时选择同意授权用户信息得到包含用户基本信息的userInfo，此时在小程序端调用wx.login，返回得到一个登录凭证code，带着code和userInfo调用登录接口。后端以code作为参数调用微信提供的<https://api.weixin.qq.com/sns/jscode2session>接口即可得到微信用户的唯一标识openid，带着openid和userInfo到service层进行数据库层面的服务。





在service层使用SQL语句查询openid，如果查到了用户就返回登录成功，如果返回空就直接insert以创建新用户，并返回用户的信息。

查询SQL: SELECT openid FROM user WHERE openid=’openid’;

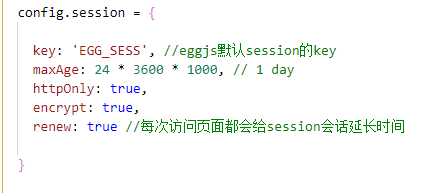
插入SQL: INSERT user set name=name, openid=openid…

1. **使用cookie+session保持及验证登录状态**

因为http协议是无状态的，所以我没有办法单单从请求半身了解到请求来自谁，此时我们就需要对请求进行验证，与此同时我不希望每一次都需要通过查库来验证得知用户的信息，或者能够通过伪造请求假冒他人请求，所以加入cookie+session以保证安全性。

做法是在服务端存储session，存储之后服务端通过md5或其他加密算法得到一个唯一的16位或32位的密文，这一串密文就是验证用户身份的证据，非常难以破解，而且每一次请求都会更新这一串密文，整个请求的过程非常安全且可以确保来请求的用户的确是来自那个用户。

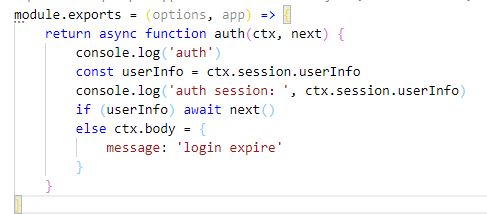
在egg中使用session非常方便，通过配置化的手段在config.js中配置即可。



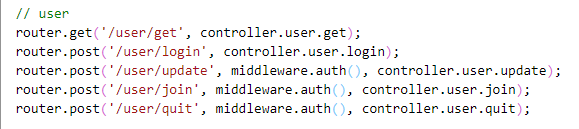
在微信小程序端，虽然没有官方的cookie接口以支持cookie传输，不过可以在请求头中加入cookie字段以模拟cookie，此方法在egg中一样可以被识别为cookie并找到存储的session信息。



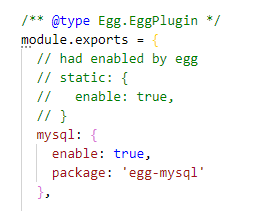
在接口中加入中间件(middleware)以拦截请求，有上述合法cookie的才可以继续通行并完成请求，如果没有的话一律返回login expire（登录过期）。



在egg中，得到app.router，并使用router.[METHOD]即可向外暴露出不同方法的请求，在第二个参数写入中间件即可。



要想在egg中实现对mysql数据库的操作也非常简单，得益于egg基于koa封装的mysql库，我只需在plugin中配置mysql插件。



在service层中

查询操作：this.app.mysql.select(‘table’,{ where:{} })

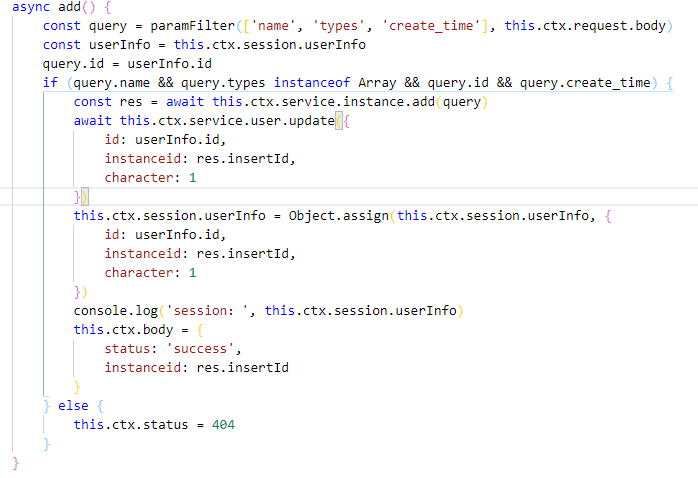
插入操作：this.app.mysql.insert(‘table’,{name:’name’,age:’10’})

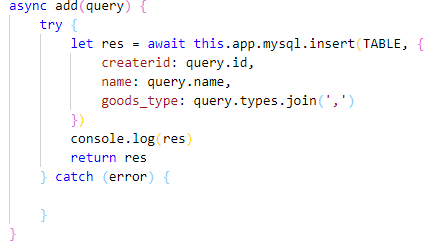
修改操作：this.app.mysql.update(‘table’,{id:id,name:’name’})

删除操作：this.app.mysql.delete(‘table’,{id:id})

也可以手写SQL语句：this.app.mysql.query(`select \* from user where id=1`)

1. **新用户创建实例**
2. 过滤参数拿到name名称，types商品种类，create\_time创建时间
3. 在session中拿到用户的id
4. 在instance表中插入数据，修改user表中该用户的instanceid为插入的instance的id
5. 返回状态码200，以及实例的id



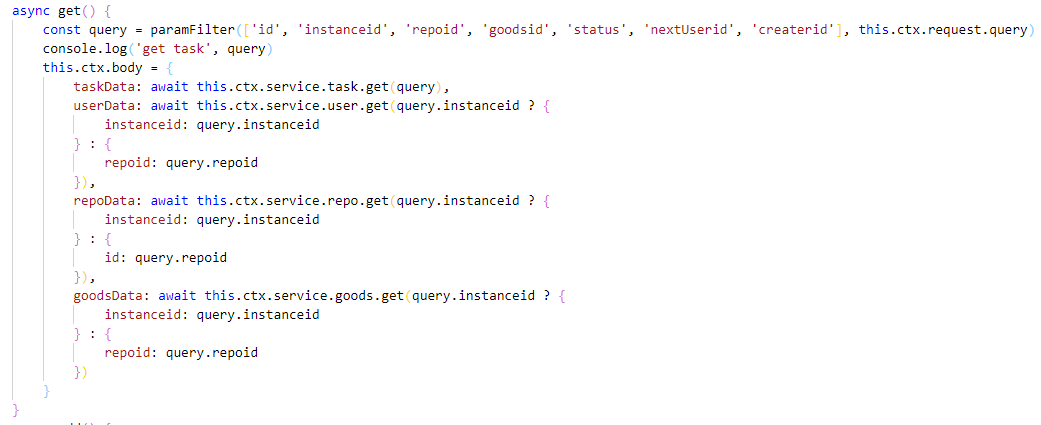


并且在router.js中向外暴露一个post请求。



1. **获取与当前仓库有关的任务消息列表**

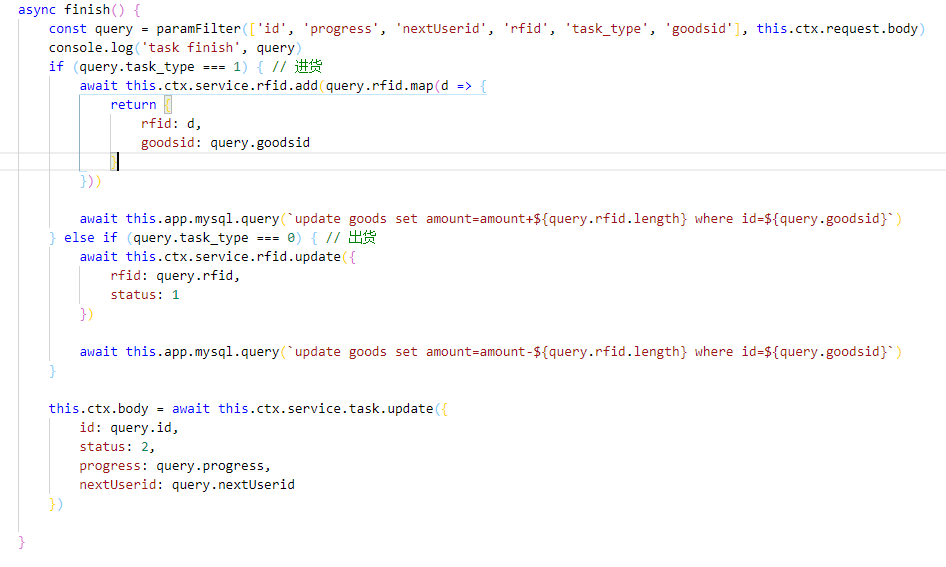
这个功能需要在单个页面展示较多的信息，如果用发送多个请求的方法会大大降低效率，使得应用性能下降，所以多次查库的操作都集成在一个task的get接口中完成，并通过格式化数据手段之后返回。



通过在控制层调用多个不同的服务得到不同的数据信息并返回一次完成查询多个表数据。

1. **任务完成接口**

通过不同的type识别不同的任务进入不同的处理方式。

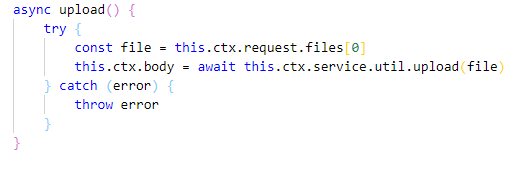


1. **图片上传功能的实现**

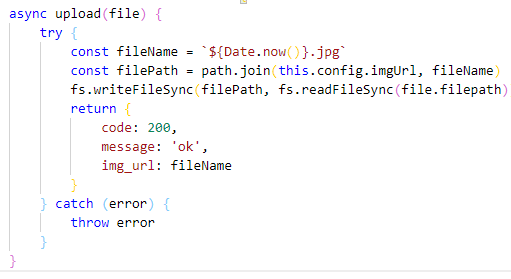
图片上传的功能主要用于商品图片的上传以及用户头像的更改。存储图片的方法不同于其他存储在数据库中的格式方法，一张图几MB的大小无法存在MySQL的字段中，所以更通用的做法是把图片的二进制数据存放在服务器中或者使用cos对象存储数据库存储，并生成url存放在MySQL的相应字段中以实现。

这次我把图片存放在egg中的public公共文件夹，以对所有用户有访问的权限。

在前端通过formData的方式传输图片文件路径，后端从request.files得到并传到service层加载图片得到二进制流。

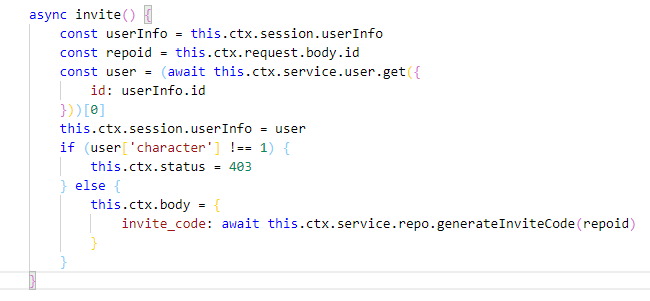


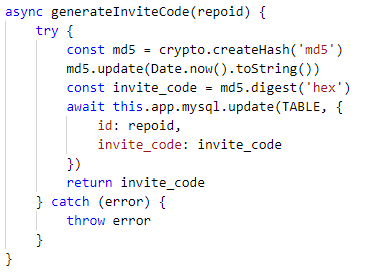
此时就需要用到nodejs的用于文件处理的fs模块。使用fs.writeFileSync功能同步写入二进制流文件到public文件夹中并生成时间戳为文件名，最后进行业务操作把生成的图片url存入数据库中即可完成流程



1. **仓库加入邀请码的生成**

为了生成不可以破解不可碰撞的邀请码，以防止其他人未经允许加入仓库导致隐私数据泄露，此处使用md5加密的方式仓库id的方式生成邀请码hash。使用了nodejs的crypto加密库。





1. **其他接口的实现**

其他的接口功能实现主要通过不同的查询，插入，更新，删除操作数据库来实现，比如查询当前用户所在仓库的所有商品，则通过获取用户的所在实例id后，传入id为参数发送get请求得到商品列表：select \* from goods where instanceid=instanceid

在后端中controller拦截不同的参数作出不同的处理就可以完成所有的后端服务。

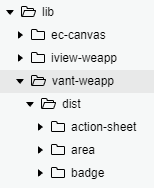
以下是以实现的所有接口：



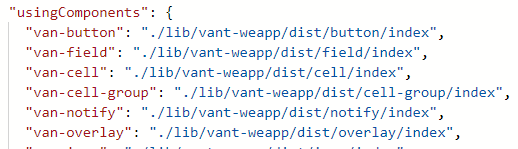
3.3 前端设计和开发

1. **微信小程序开发特性与原理**

微信小程序是基于html5构建的web应用，在微信提供的小程序框架中运行。一个完整的小程序页面由wxml, wxss, json, js构成。wxml 类似html是一种标签语法的声明文档以表示页面的dom结构，与html不同的是有微信专门优化封装的专用组件。Wxss类似css是层叠样式表，语法与css类似，作用同样是编写样式。Js中是实例化本页面的代码，一个Page由提供动态绑定功能的数据data，生命周期函数onReady,onload……以及自定义的函数构成。遵循MVVM（model, view, view-model）的风格，在wxml确定视图层结构之后，根据js的data解析出数据结构model并渲染到视图层中，此时整个生命周期中页面监听data的变化生成view-model，一旦数据产生变化view-model则会监听到通过diff算法找到最低成本更新视图的策略并驱动视图层的渲染。

**1. UI库的引入使用**

因为微信小程序的独特性，在json中定义component为true之后每一个wxml都是一个可以单独引入的组件。在vant-weapp的github上下载编译好的版本dist/文件夹并复制到微信小程序的lib/文件夹下，然后在app.json中的”usingComponents”字段配置所需要的所有component即可。



**2. 封装及promise化函数request**

由于官方的wx.request({})以及其他api的函数风格会造成地狱回调的现象，在多个连环请求的状态下代码风格会变得臃肿而难以维护，过多的嵌套也会导致性能的降低。

此处在utils/中写promisefy.js进行封装并写成promise风格的api以方便使用。

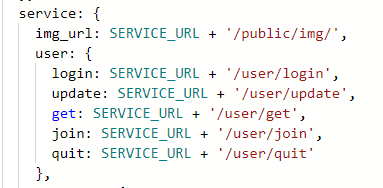


在这里对request的特定情况做了拦截和做出统一处理，这样的好处是可以大大减少重复代码，高效开发。在权限不足的情况后端一律返回403Forbidden，此处直接拦截；在返回message:”login expire”的时候直接重定向到登录页面；每一次请求后自动存储cookies。如果请求失败自动弹窗错误提示。

**3. 配置化的接口风格**

在app.js中，写一个service属性专门用于记录所有的接口，并用变量的方法配置根域名方便本地环境和远程环境的切换。

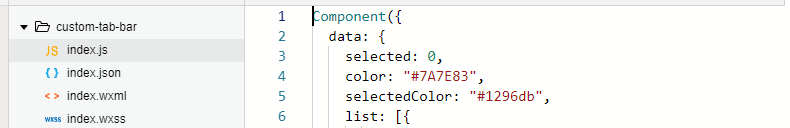




**4. 自定义置底的tabbar**

在app.json中配置。

编写一个custon-tab-bar并实例化成Component，在json中component字段为true，然后在这个wxml中编写tabbar的结构，在wxss中编写样式，在js中编写他的功能并指向不同的页面即可实现自定义的tabber。

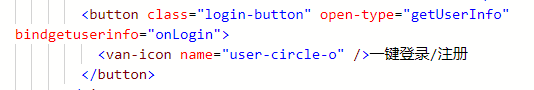


效果如下：



1. **功能页面的开发实现**

我的页面中使用<button open-type=”getUserInfo”>按钮以实现对用户授权的询问。



用户同意授权之后发起login请求完成登录逻辑，返回登陆成功之后便完成登录。



后端的登录注册代码如下：

async login(query) {let res = await this.app.mysql.get(TABLE, {openid: query.openid        })        if (!res) res = await this.register(query) console.log('login res：', res)        // 存储对应用户的userInfo，返回的时候伴随cookie，下次接口调用的时候识别cookie拿到该session      this.ctx.session.userInfo = resreturn res}    async register(query) {   await this.app.mysql.insert(TABLE, {     openid: query.openid,            name: encodeURI(query.userInfo['nickName']), // 如果是emoji昵称需要转码            avatarUrl: query.userInfo['avatarUrl'] }) return await this.app.mysql.get(TABLE, {            openid: query.openid})    }

登录完成之后点击创建仓库，即可进入创建仓库的页面。

表单和输入框在vant-weapp提供的组件上修改复合蓝色的主色调。



创建子仓库之后便可以看到当前仓库的状态并开始使用更多功能。

在库存页面编写了一个根据条件筛选商品以及通过关键词搜索商品的组件。实现如下：

配合js实现渲染出不同分类的条件选项并在选择之后发出请求得到数据即可实现。

点击左上角的加号可以进入添加商品页面，在该表单中可以添加一个商品的所有信息。

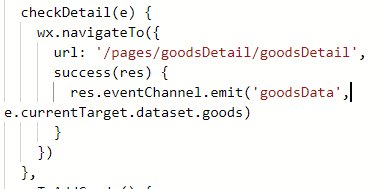
小程序用到的所有关于列表的渲染都使用了微信小程序api的循环渲染指令wx:for，如渲染出一个商品列表就可以写成:

<good wx:for=”{{goods}}” wx:key=”index">{{item.name}}</good>

商品详情的页面传参原理：

通过编写一个商品详情的前端页面，设计好固定的结构，样式和数据处理逻辑。点击库存页面中的商品列表之后通过点击事件的参数传递拿到在dom中指定的dataset，并带着商品的数据去到商品详情页面使用相应的数据渲染对应的页面。





任务消息页面的搭建：

使用一个顶部标签页组件，则可以实现在同一个页面通过滑动切换不同消息功能，同样通过循环渲染wx:for则可以完成任务列表的实现。

标签页的实现：<van-tabs active="{{ activeTab }}" bind:change="onTabChange" swipeable animated sticky> <van-tab title="仓库"></van-tab><van-tab title="成员"></van-tab></van-tabs>

任务列表的实现：<view class="task-goods-item" wx:for="{{task\_goods}}" wx:key="index"><view class="name">{{item.name}}</view>

<view class="id">#{{item.id}}</view>

<view class="detail" data-index="{{index}}" bindtap="onPreview">

<van-icon name="description" />详情</view>

<view class="type">

{{item.type\_text}}

</view>

<view class="status">

<van-tag plain type="{{item.status\_tag\_type}}">

{{item.status\_text}}

</van-tag>

</view>

<view class="operation" wx:if="{{item.status == 0}}">

<van-button size="mini" data-id="{{item.id}}" round type="primary" bindtap="accept" wx:if="{{user.character !== 1}}">接受</van-button>

<van-button size="mini" data-id="{{item.id}}" round wx:if="{{user.character == 1}}" bindtap="withdraw">撤回</van-button>

</view>

<view class="nextUser" wx:if="{{item.status !== 0 && item.nextUserid}}">

<van-button size="mini" data-item="{{item}}" round type="info" bindtap="progress" wx:if="{{item.status === 1}}">进度</van-button>

<image class="avatar" src="{{item.nextUser.avatarUrl}}"></image>

<text>{{item.nextUser.name}}</text>

</view>

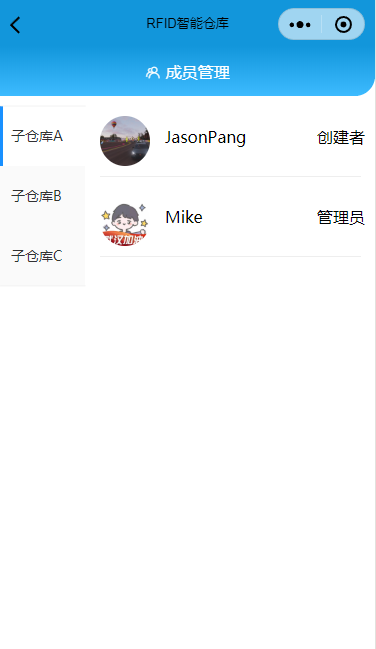
<view class="create-time">{{item.create\_time}}</view>

</view>

创建任务的表单页面，通过堆砌vant-weapp提供的输入框，选择器，并编写数据的处理逻辑就可以实现。页面在打开的时候马上请求数据得到可供选择的商品列表，仓库列表，以提供用户选择来限定用户的输入。用户完成表单输入之后可以预览整个任务清单检查是否有错误可以进行修改，如果没有错误点击提交就可以完成任务的创建。

任务消息的产生实际上是由事件或事务触发的数据库插入行为，当特定的事件发生如入库任务的生成，成员的加入都会触发到任务消息的产生，之后当用户进入到消息任务的页面的时候，通过查询数据库的请求就可以获取到最新的消息任务了，然后前端列表的渲染同样是依靠微信的循环渲染实现。以下成员加入之后的页面效果。

总览页面主要展示对仓库各项数据的统计以及可视化数据表达包括一个饼图，一个折线图，一个条形图，通过由百度维护并开源的js图表库echarts实现，在github中可以找到它的一个为为微信小程序适配的版本。把库放入lib/中再通过页面Import组件ec-canvas即可实现。



此处需要为不同的图表格式化数据使其适配。

适配代码如下：

let {

goodsData,

repoData,

taskData,

userData

} = res.data

// 折线图

const lineData = {

data: [],

xAxis: []

}

taskData = taskData.filter(t => t.type !== 2).sort((a, b) => b.create\_time - a.create\_time)

const latestTask = taskData.find(t => t.status === 0)

const noticeText = `[新任务]${latestTask.name} 任务类型：${config.task[latestTask.type].name} 商品：${goodsData.find(g => g.id === latestTask.goodsid).name} 数量：${latestTask.amount}`

const barDataType = ['outData', 'inData']

// 条形图

const barData = goodsData.map(g => {

g.inData = 0

g.outData = 0

return g

})

// const dataAmount = taskData.length

taskData.forEach((t, i) => {

const goodsName = goodsData.find(g => g.id === t.goodsid).name

const goodsAmount = goodsData.find(g => g.id === t.goodsid).amount

// 条形图数据格式化

const barDataIndex = barData.findIndex(b => b.id === t.goodsid)

barData[barDataIndex][barDataType[t.type]] += t.amount

// 折线图数据格式化

// 未有该任务类型的该商品

const date = new Date(t.create\_time)

const date\_text = `${date.getMonth() + 1}/${date.getDate()}`

t.name = `${goodsName} - ${config.task[t.type].name.substr(2, 2)}` //最后两个字

t.taskType = t.type

t.smooth = 'true'

const lineDataIndex = lineData.data.findIndex(e => (e.goodsid === t.goodsid && t.taskType === e.taskType))

const xAxisIndex = lineData.xAxis.indexOf(date\_text)

if (t.type === 0) {

t.amount = -t.amount

}

if (xAxisIndex < 0) {

lineData.xAxis.push(date\_text)

for (let i = 0; i < lineData.data.length; i++) {

lineData.data[i].data.push(0)

}

if (lineDataIndex < 0) {

t.data = Array.apply(null, Array(lineData.xAxis.length)).map(() => 0)

t.data[lineData.xAxis.length - 1] = t.amount

lineData.data.push(t)

} else {

for (let i = 0; i < lineData.data.length; i++) {

lineData.data[i].data.push(0)

}

lineData.data[lineDataIndex].data[lineData.xAxis.length - 1] = t.amount

}

} else {

if (lineDataIndex < 0) {

t.data = [t.amount]

lineData.data.push(t)

} else {

lineData.data[lineDataIndex].data[xAxisIndex] += t.amount

}

}

})

lineData.data = lineData.data.map(l => {

l.type = 'line'

return l

})

this.setData({

// 饼图数据格式化

pieData: goodsData.map(g => {

g.value = g.amount

return g

}),

lineData: lineData,

barData: barData,

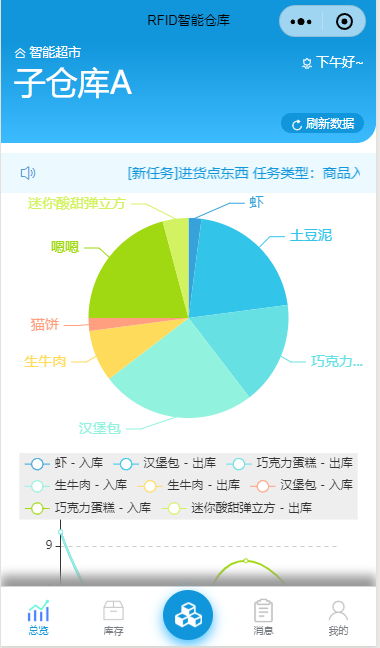
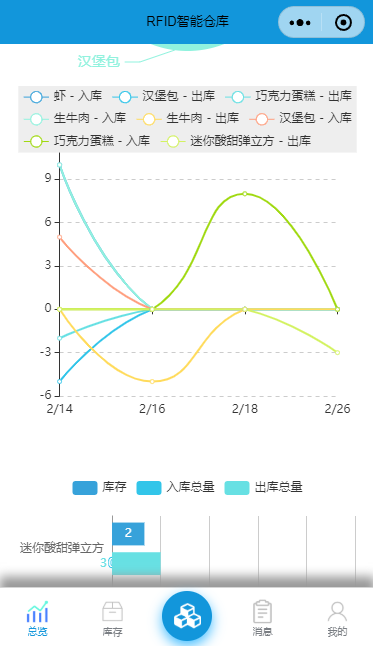
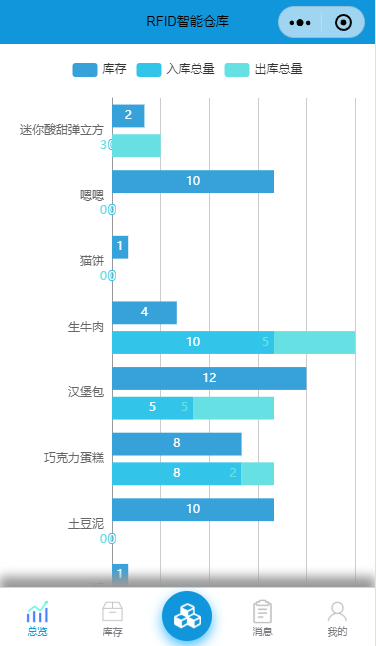
repo: repoData[0],

noticeText: noticeText,

dataReady: true

})

随着商品的越来越多以及进出库的任务越来越频繁之后图表就会显示出如下效果：

以下是关于RFID识别的开发和实现。

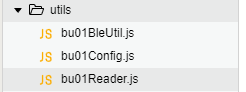
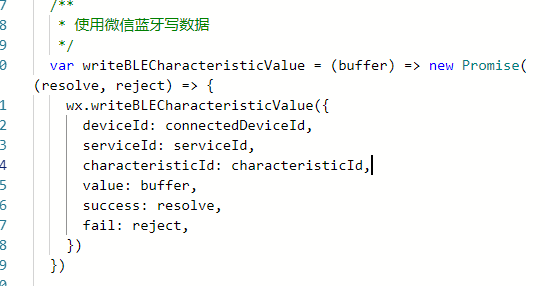
RFID的功能需要借助一台RFID识别读写设备，通过蓝牙和手机进行通讯，在设备识别到电子标签的时候在蓝牙的通讯中通知到微信小程序，微信小程序可以通过微信官方的蓝牙api再发送指令给电子标签写入数据等等以此完成了整个的通讯循环：

微信小程序<=>蓝牙<=>RFID设备<=>电子标签

使用右图的RFID设备，优点是作为一个穿戴式设备用户可以当作手表戴在手上进行进出库等等业务操作非常方便，无需携带其他读写设备，而且支持通过蓝牙连接，用户友好度高无需学习成本。

RFID技术是从雷达技术衍生而来的，它的技术原理是通过射频识别技术发射特定的频率和波长与电子信息的记录载体如电子标签和磁卡等等进行一种非接触的信息交换，从而就达到了识别，读取，写入的功能。这样的工作逻辑再配合一定的数据交换技术例如计算机网络，http请求，就可以组成一个庞大的物联网系统。

通过使用随设备附带的小程序开发SDK，我封装了几个用于通过微信蓝牙接口操作设备的API。

主要的代码逻辑就是

1. 微信小程序通过wx.bluetoothAdapter连接RFID设备
2. 设定RFID设备清点模式，有单次清点（清点到电子标签之后结束识别），连续清点（一直开启识别知道手动停止清点为止）
3. 设备读取到电子标签信息并通过蓝牙接口给微信小程序返回数据。
4. 微信小程序收到数据特征值如电子标签的唯一识别码EPC，并进行数据处理和发出http请求的操作以完成业务需求。
5. 断开设备连接。

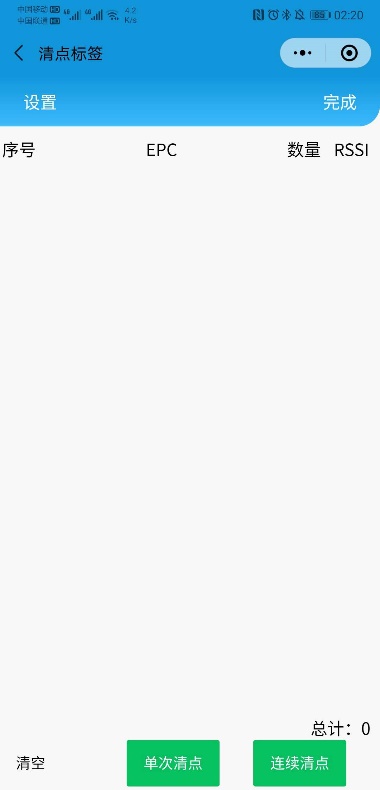
在进出库任务进度中，用户可以使用RFID设备进行清点。进入任务详情页面可以看到一个圆形进度条，意为进出库的数量进度，使用vant-weapp提供的van-circle组件实现。初始状态进度条为空白，即为0。

点击连接设备并开始清点，小程序会开启蓝牙设备搜索，搜索到设备之后进行连接。然后选择清点模式之后，把带有电子标签的商品靠近RFID设备之后，设备会发出蜂鸣嘀一声之后微信小程序就会接收到相应的EPC数据，根据EPC码小程序发送请求验证该EPC是否属于所需的商品，如果正确的话进度条就会继续行进直到满为止。如果不正确的话就会弹出提示窗提示错误原因。进度条满之后，用户点击确认完成即可完成一次出入库的任务。全程都在RFID的监控之下，保证结果的严谨正确。

用户还可以在【我的-我的任务】页面中监视到自己创建的任务所有任务的进行情况，实时远程进行监管。



RFID的使用还支持日常的查询工作。

在tabbar的中间栏是快捷功能的入口，打开可以见到一个各种快捷功能的入口。包括通过RFID查询设备，通过连接，识别的操作之后小程序会自动跳转到相应商品的详情页面，高度自动化。

还可以通过扫描条形码的功能直接跳转到相应商品的详情页面，实现原理是通过wx.scan通过得到的code在数据库中查询已记录的商品并跳转到详情页面获取数据。

至此结合RFID开发所有前端页面都已实现。

结论

经过了三个月的需求分析，技术研究，产品设计，开发调试终于把这一款结合RFID的智能仓存管理系统完成实现。从得到导师给我的课题开始，我就给这个项目查阅各种资料及文献去落实产品的实际功能和需求。需求的初始版本是需要使用语音控制微信小程序生成相应的进库单和出库单，但这一功能在我经过调查和研究之后发现其实质的应用实用性并没有想象中的高，大多数的仓库管理人员对于仓存物品的进出库及在库管理都是持着一个要质量不要数量的态度，也就是说他们更在意是否能够精准地下达一个进库单或出库单，而不在意如何快速地完成却没有办法保证质量或需要多次修改。在现有的技术中语音识别再语义控制的流程还需要一个较高精度的平台去进行，包含了对声音模型的训练，语义联想等等，这都会造成后期维护成本的加大，所以实在得不偿失。在设计产品功能的时候我尽量从一个使用者的角度去考虑排版布局配色功能，脱离了开发者我得到了一个全新视角去思考软件工程的本质。其中的设计经过了多次迭代才到了如今的版本，比如与RFID设施通讯的流程，用户提交任务的操作形式等等。在选用RFID设备的时候浏览了几十种不同的设备，从他们支持的协议，设施的辨认距离频率，附带的开发SDK的角度去考量其可用性。微信小程序的开发工作我还算得心应手，其总体的开发风格和Vue.js非常类似，dom结构，css样式，js逻辑分开的结构非常易于理解。在实际维护的过程中，我首次尝试的Egg.js后端服务也体现出了很高的性能，在多用户同时IO请求的场景下依然能够维持很好的性能表现，数据的获取在页面渲染的过程中不构成阻碍。

整体来说，这是一个适合个体小商户的基于RFID的智能化仓存管理系统，因为他易于上手，功能直达个体小商户的痛点，且维护成本很低，整个系统开发一次可以提供给很多家仓库共同使用。但是同时仍有很多可以改进的地方，留到日后继续迭代完善。

参考文献

[1]钟亚,刘瑛.基于RFID和ZigBee的快递管理系统[J].计算机系统应用,2019,28(11):126-131.

[2] 项进解,张永衡,张凯.基于RFID手持终端的智能仓库管理系统研究及实现[J].自动化与仪器仪表,2019(11):96-99.

[3] 王松波,陈凡健.“智慧物流”背景下的智能仓储系统设计研究[J].现代信息科技,2019,3(17):160-162.

[4] 王珏辉.基于响应式设计模式的Web前端开发研究[J].电脑编程技巧与维护,2019(04):31-32.

[5] 孟永焱. 基于微信的移动互联网智能交互系统的设计[D].北京邮电大学,2019.

[6] 毛黎霞.智慧物流背景下现代物流路径探析[J].现代营销(信息版),2019(12):60-61.

[7] 张志鹏,黄素娟,周永圣,王楠.基于React技术的单页APP的设计与实现[J].微型电脑应用,2019,35(10):71-74+105.

[8] 张静. 基于NodeJS的自适应浏览器高性能Web应用平台研究[D].杭州电子科技大学,2017.

[9] 潘英雯,逯力红,张宇.基于物联网的超市商品数据库检索系统的开发[J].物联网技术,2019,9(11):113-115.

[10] 韦宇炜. 基于OpenGL技术的三维游戏引擎开发[D].广东工业大学,2005.

致谢

经历了这么长的时间终于可以给这次应用的开发和论文撰写画上美满的句号。这期间经历了十分多的逻辑推敲，细节打磨，技术研究，当然也让我克服了不少的困难。整个过程中，我非常感谢韦宇炜教师的命题推荐以及细心的技术指导还有购买设备的资金支持。他谨严谨慎的工作风格，精益求精的科学态度都深深地感染了我，他的悉心教导也让我在技术上走到了更远的道路的同时也让我学到了技术之外的东西，让我获得了无限的启迪，打开了全新的思路。

感激所有给予我撑持和辅佐的老师以及同窗，他们在我的大学学习成长之路上起到了必不可少的作用，每一次在我遇到难以跨越的困难的时候是他们给予了我耐心的解答，又让我重拾面对困难的勇气。

最后要感谢自己，没有在最失落的时候气馁，没有在面对困难的时候退缩，唯有锲而不舍地追求卓越才能达到理想的成功。