

广州铠信挖掘机租赁管理系统

【原文对照报告-大学生版】

报告编号: bb916fd3c940c299

检测时间: 2019-04-03 20:41:48

检测字数: 25,428字

作者名称: 陈岩

所属单位:

检测范围:

- | | | |
|------------------|-----------------|-------------------|
| ◎ 中文科技期刊论文全文数据库 | ◎ 中文主要报纸全文数据库 | ◎ 中国专利特色数据库 |
| ◎ 博士/硕士学位论文全文数据库 | ◎ 中国主要会议论文特色数据库 | ◎ 港澳台文献资源 |
| ◎ 外文特色文献数据全库 | ◎ 维普优先出版论文全文数据库 | ◎ 互联网数据资源/互联网文档资源 |
| ◎ 高校自建资源库 | ◎ 图书资源 | ◎ 古籍文献资源 |
| ◎ 个人自建资源库 | ◎ 年鉴资源 | ◎ IPUB原创作品 |

时间范围: 1989-01-01至2019-04-03

检测结论:

全文总相似比 = 复写率 + 他引率 + 自引率 + 专业术语

2.38% = **2.38%** + **0.00%** + **0.00%** + **0.00%**

其他指标:

自写率: 97.62%

专业用语: 0.00%

高频词: 系统, 数据, 管理, 实现, 用户

典型相似性: 无

指标说明:

复写率: 相似或疑似重复内容占全文的比重

他引率: 引用他人的部分占全文的比重, 请正确标注引用

自引率: 引用自己已发表部分占全文的比重, 请正确标注引用

自写率: 原创内容占全文的比重

专业用语: 公式定理、法律条文、行业用语等占全文的比重

典型相似性: 相似或疑似重复内容占互联网资源库的比重, 超过60%可以访问

总相似片段: 34

期刊: 5 博硕: 13 外文: 0 综合: 1 自建库: 0 互联网: 15

颜色标注说明:

- 自写片段
- 复写片段 (相似或疑似重复)
- 引用片段
- 引用片段(自引)
- 专业用语 (公式定理、法律条文、行业用语等)

摘要 随着信息技术的发展,传统企业为提升运营效率而开始信息化建设。使用电子表格进行业务支撑已经不合时宜,定制化的基于网站技术的业务运营支撑系统渐渐流行起来。

业务运营支撑系统旨在整合计费结算,业务运营管理,客户管理,决策支持四个子系统,系统通过统一的数据建模来实现各系统间的数据交换和协调配合,以此来对公司的日常活动进行支撑。

同时,为了提高性能和用户体验,系统开发使用了基于JavaWeb的服务器端,vue.js实现的网页交互逻辑以及MySQL数据库提供的持久化方案。

关键词 信息化建设;业务运营支撑系统;JavaWeb;Vue.js

ABSTRACT With the development of information technology, traditional enterprises begin to build information technology in order to improve operational efficiency. It is out of time to use spreadsheet for business support. Customized Business & Operation Support System (BOSS) based on website technology is becoming more and more popular.

BOSS aims at integrating four subsystems: Billing and Settlement, Business Operation Management, Customer Management and Decision Support. The system realizes data exchange and coordination among subsystems through unified data modeling, so as to support the daily activities of the company.

In order to improve the performance and user experience, the system uses Java Web-based server side, web page interaction logic implemented by vue.js and persistence scheme provided by MySQL database.

KEY WORDS build information technology;; Business & Operation Support System; JavaWeb; Vue.js

目录

第一章 绪论 6

1.1 前言 6

1.2 选题背景和意义 6

1.3 现状和发展情况 6

1.3.1 市场现状 6

1.3.2 同类竞品分析 7

1.4 主要研究内容 7

1.4.1 BOSS (Business & Operation Support System) 实现 7

第二章 技术选型 8

2.1 设计模式 8

2.1.1 前后端分离&动静分离 8

2.1.2 MVVM模式 8

2.2 技术选型 9

2.2.1	Spring	9
2.2.2	Struts	2 10
2.2.3	SpringMVC	10
2.2.4	Hibernate	11
2.2.5	Mybatis	11
2.2.6	MySQL	11
2.2.7	JSP/静态HTML	11
2.2.8	Vue.js	12
2.2.9	Tomcat	12
2.2.10	Hessian	12
2.3	技术选型总结	13
2.3.1	平衡用户体验以及性能要求	13
2.3.2	最终选型	13
第三章	系统总体设计	13
3.1	用户需求分析	13
3.1.1	订单管理	14
3.1.2	客户管理	14
3.1.3	派单管理	14
3.1.4	租赁明细管理	14
3.1.5	对账单打印	15
3.1.6	收款管理	15
3.1.7	固定资产管理	15
3.2	系统架构设计	15
3.3	数据库设计	17
3.3.1	bean_customer 客户信息表	18
3.3.2	bean_contact 客户联系人	18
3.3.3	客户-联系人连接表	19
3.3.4	bean_driver 驾驶员信息	19
3.3.5	Bean_support 后勤人员信息	20
3.3.6	Bean_machine挖掘机信息	20
3.3.7	Log_oil 加油日志	21
3.3.8	Log_transport 转场日志	21
3.3.9	Log_error 异常处理日志	22
3.3.10	Log_maintain 维修记录表	23
3.3.11	Log_work 工单记录	23
3.3.12	Service_compact 合同表	24
3.3.13	Service_intention 客户意向登记	25
3.3.14	Sys_drawer 系统菜单管理	25
3.3.15	com_resource 文件系统	26

3.3.16 Bean_bank_account 公共银行账号 26

3.3.17 Bean_machine_type 机器图鉴 27

第四章 系统实现 27

4.1 后端核心组件 27

4.1.1 数据实体 (Entity) 28

4.1.2 持久层 (DAO) 29

4.1.3 通讯模型 30

4.1.4 服务层 (Service) 31

4.1.5 业务逻辑层 (Controller/filter) 32

4.1.6 权限拦截器 34

4.2 前端核心组件及模板 35

4.2.1 前端静态文件目录结构 35

4.2.2 主体框架设计 36

4.2.3 级联选择弹窗注册器 37

4.2.4 文件管理组件 39

4.2.5 标准数据表格 39

4.2.6 查看/编辑/新增/审核 四合一组件 42

4.2.7 概览页面 45

4.2.8 挖机地图 (基于高德地图API) 46

4.2.9 前端异常处理 47

4.3 数据库优化 48

4.3.1 将视图作为实体映射 48

4.3.2 使用触发器维护冗余字段 49

4.4 用户权限管理系统 (User Authority Management System) 49

4.4.1 技术选型 50

4.4.2 数据库设计 50

4.4.3 系统架构 51

4.4.4 页面设计 51

4.4.5 外部接口设计 53

第五章 总结与期望 54

5.1 系统的缺陷以及后续升级计划 54

5.1.1 业务逻辑支撑 54

5.1.2 前端组件化 55

5.1.3 建立日志 55

5.1.4 数据高速缓存 55

5.2 总结 55

第1章 绪论

1.1 前言

企业信息化建设是很多传统企业走向壮大的必经之路，在信息化的大潮中，很多小型传统企业意识到了这一点，仅仅依靠传统的电子

表格做业务支撑是远远不够的，耗时费力。利用互联网来组织业务才能更合理、高效地调配资源，这也是本课题致力解决的问题。

1.2 选题背景和意义

随着时代和互联网的发展，传统企业的业务运营方式渐渐变得繁琐，低效，运营成本高昂。为了降低企业的运营成本，提高运营效率以及降低人员成本成了必由之路。基于互联网的BOSS (Business & Operation Support System) 业务运营支撑系统应运而生，通过整合传统的ERP系统，CRM系统，计费系统，来打通各系统间数据交互的通道，还可以通过数据聚合分析来发现潜在的商业价值，优化企业业务流程。

由于机缘巧合，本人母亲工作的挖掘机租赁公司仍然处于传统的基于人工的业务运营系统下，随着业务的发展壮大，迫切地需要一套业务支撑系统。在对行业的了解和沟通后，我发现设备租赁行业仍缺少定制化的BOSS系统。作为即将步入软件行业的一员，可以在同时磨练自己专业技能的同时创造社会价值当然是一举两得的选择，于是就萌生了自己开发一套基础的BOSS系统的想法，希望能在完善后可以应用在实际生产当中。

1.3 现状和发展情况

1.3.1 市场现状

经过一定的市场调研（百度搜索引擎结果，几家大型设备的租赁商运营方式），发现大型工程设备的租赁管理系统尚有很大的空缺。

传统的ERP软件开发商如用友，金蝶等软件开发商暂未进入该市场，当前工程机械租赁公司的业务支撑主要有以下几种方式：

- 1) 基于Excel表格的传统记账方式，使用纸质回执的方式来控制业务流程，在面对大量业务时会产生较高的人工成本，并且易出错，且极难统计业务数据，洞察市场趋势。
- 2) 使用传统的ERP软件进行管理，效率较高，但缺乏自定义业务流程的支持，数据流通性差，且缺乏资产定位能力。
- 3) 使用企业订购的资产追踪系统附带的业务管理功能，初步将资产定位和业务结合在一起，但对业务流程的支撑较差。

由此可见，当前较多租赁企业的业务运营都处于高度依赖人工的状态下尤其是业务数据的收集工作。因此，当前的设备租赁的BOSS系统具有较高的市场价值。

1.3.2 同类竞品分析

荣峰科技开发的管租易是一个比较完善的租赁管理解决方案，在业内有一定的市场占有率，其以后台WEB客户端为核心，构建出包含微信公众号，安卓APP，GPS资产定位，等业务实施终端的一体化业务支撑系统，并拥有一定的业务统计能力，但由于其使用成本较高（标准配置下约15万/年），而且系统较复杂，因此限制了一些客户对业务系统的重视程度。

综上所述，开发一个较简单，并可以与传统EXCEL表格进行交互以存续历史业务数据的业务支撑系统就有了实际的意义，也就是本人所努力的目标。

1.4 主要研究内容

1.4.1 BOSS (Business & Operation Support System) 实现

全称业务运营支持系统，通常来说分为四个组成成分：计费及结算系统，营业与账务系统，客户服务系统以及决策支持系统，将其整合在一起便于数据通信以及纵向和横向管理。

本文主要探讨如何开发广州铠信公司所需要的挖掘机租赁管理系统，该系统使BOSS的一种实现，主要包括以下模块：

- 1) 基础数据管理（机手信息管理，客户信息管理，机器管理，机器型号管理，后勤人员管理）
- 2) 日志数据管理（异常反馈管理，维修单据管理，加油单据管理，转场管理，工单管理）
- 3) 业务数据管理（合同管理，挖机地图，客户意向管理）
- 4) 系统管理（用户管理）

系统为前后端分离架构，后端仅提供数据接口，前端维护用户操作逻辑以及视图渲染。引入高德地图API以实现挖机地图，Chart.js用于绘制图表。

第2章 技术选型

2.1 设计模式

2.1.1 前后端分离&动静分离

在早期的WEB应用开发中，大多数实现技术（PHP, Servlet, JSP等）都将业务逻辑和视图控制编写在同一个文件中，每个页面对应一个文件，处理代码间错综复杂的依赖关系和代码维护一直是业界难题，随着技术的发展，MVC模式（Model-View-Controller）渐渐成为公认的优秀设计思想，其对数据模型，视图，控制器的分层逻辑对之后的软件开发产生了深远的影响。近年来流行的前后端分离正是这一思想的衍生产物。

前后端分离，顾名思义，前端页面与后端接口完全分离开发，前端（浏览器）负责用户操作逻辑以及页面渲染，并通过API接口与后端（服务器）进行数据交互。后端则专注于实现API接口里的具体业务逻辑。在这种思想下，前后端都拥有了各自独立的两套MVC架构来实现各自的逻辑，并基于XML、JSON等数据交换格式，以及序列化和反序列化技术来实现前后端通信。需要注意的是，在后端的MVC结构中。View已近由原来的HTML文档演化为前端可以直接解析的JSON视图。

在这种开发模式下，WEB应用实现了静态化，HTML、CSS、JS等需要在浏览器中运行的静态文件可以存放到高性能的静态资源服务器（如Nginx等），而Web服务器（如tomcat, weblogic等）则可以有更多的资源来处理用户需求，以此实现了动静分离以及负载分担，提高了网站的性能以及稳定性。

从开发的角度来讲，前后端分离使得前后端开发人员直接面向接口编程，前端不用关心API的实现方式，后端不用关心页面的布局以及交互逻辑，这极大地提高了企业的开发效率，降低了运营成本以及后续运营的成本，往后的迭代升级只需要修改API或面向API进行向下兼容即可。

2.1.2 MVVM模式

MVVM全称Model-View-ViewModel（模型-视图-视图模型绑定）是前端MVC模式的改进版。如果你是新手前端开发者，一定会遇到类似都XXXX年了你怎么还在用Jquery?。不可否认的是，Jquery是一个非常强大的JS库，使得你对DOM的操纵得心应手，然而思想是在不断进步的。

试想你正在实现一个简单的数据表格，当用户加了一个筛选条件后我们需要重新渲染表格，最开始大家想的很简单：把TABLE下的DOM全部丢掉，重新渲染一遍。Jquery可以轻松胜任这项工作，但当你的页面布局比较复杂或者加入了一些动画效果时，就会产生严重的卡顿和画面撕裂，影响用户体验，这肯定不行。于是人们想出了先筛选出不需要重新渲染的DOM节点再对其他的DOM节点进行处理，这样显著的提高了界面性能，然而如果用JS和JQuery实现这个将是一件复杂的工作。

久而久之，人们发现处理视图渲染的工作有很高的复用性，MVVM应运而生，现在只需要修改Model, View-model的双向绑定就会帮你处理视图渲染的问题，并使用特定的JSON对象来描述一个虚拟DOM节点，并以此为依据来渲染DOM，以及实现部分更新的效果。大量的优秀MVVM框架如雨后春笋般爆发，其中的代表便是Vue.js、Angular等。在大家的努力下，我们拥有了可以在前端运行的JSP，只需要将模型放到想放的位置即可，数据处理交给框架，前端开发者更关心的是页面美感，用户也得到了更好的体验。

2.2 技术选型

2.2.1 Spring

Spring 自2004年发布以来十几年的发展，已经成为Java应用开发中最重要的轻量级框架之一。与Java官方定义的EJB3.0，由JBoss, ORACLE提供开源的和商业的解决方案不同，Spring 面向的是轻量级的，非分布式的中小型企业应用开发，其具有如下几个技术特点：

依赖注入：将类的实例化，装配工作交由Spring统一管理，从而实现各组件间的解耦以及性能优化。

低侵入性：配置工作通过XML或注解进行，对旧项目的适配相对友好。

独立于WEB服务器：Spring可以独立运行或者运行在各种web容器中，可以很轻易的移植，而不像EJB那样需要大量的适配工作。

开放的ORM整合：Spring 可以和多种第三方ORM解决方案整合，来简化底层数据库的访问工作。

AOP特性：Spring AOP可以对一些通用任务如事务，安全，日志进行统一的集中管理，提高代码复用率。

基于以上几种技术特点，我认为Spring是当下本项目的最佳解决方案，其组织各模块的模式也非常值得学习和探讨，因此，此项技术将用于实现后端的Service（服务层）以及DAO（数据访问对象）。

2.2.2 Struts 2

Struts2 源自传统的Struts 和 Webwork发展而来，是一个非常经典并且优秀的MVC框架，在前后端未分离，JSP大行其道的年代非常流行，以至于非常多的项目依然在使用它，并且其基于filter（拦截器）的实现思路非常适合新手学习和理解WEB应用的架构。本项目在初期也准备使用该技术，但经过调查研究后发现了其与项目需求相冲突的地方，这也直接导致了项目的第一此重构：

对前端广泛使用的JSON数据格式支持较差，参数的反序列化需要引入比较古老的JSON插件，序列化则需要直接处理response对象，对开发和后期维护产生了不利的影响。

参数隔离和性能问题，在struts的action中，前端传入的参数默认将填充至Action对象的对应字段中，不同路由对应的API参数因此集中在一起，给数据安全带来了隐患。

对跨域请求的支持很差，在动静分离的项目中，我们有两种方式区分浏览器请求的是静态资源还是API接口，第一种是使用Nginx配置一个反向代理，将发送给API的请求转发给WEB容器。第二种是使用跨域请求，这种方式对用户的带宽要求较高（需要额外发送一次请求用于验证）。

因此，尽管struts2 是一个我比较熟悉的WEB框架，但是项目的需求更为重要，因此我放弃使用它。

2.2.3 SpringMVC

SpringMVC作为Spring全家桶中的一员，因其与Spring无缝集成，正逐渐取代Struts2成为主流的Web框架，这里简单列举其与本项目的需求的契合之处：

跨域请求支持：当不准备使用反向代理时，静态资源和API是在不同的域下的，因此需要发送跨域请求来访问接口。

安全的参数装载方式：与Struts2 将参数装载至字段中不同，SpringMVC将请求参数装载为方法参数，开发者可以处理其中的参数映射，并且由于其核心Servlet为单例模式，方法参数可以在保证线程安全的情况下进行并发操作。

完整的JSON视图支持，SpringMVC将java对象和JSON中的序列化看作视图处理，只需要指定返回的视图为JsonView或者在方法上注解@ResponseBody 将其指定为返回体。

出于以上需求，我在最终实施的时候选择SpringMVC作为后端的MVC框架。

2.2.4 Hibernate

Hibernate作为一款成熟的ORM框架，其完整地实现了JPA方案。全映射作为其重要特性与JPA规范相结合使得开发人员不必关心对象和关系映射问题，直接面向对象编程，使得快速开发成为了可能，是个人开发小型系统的优秀选择。但其也存在着优化困难，效率低下等问题，关于如何解决这些问题大家都有各自的见解，我也将在下文中阐述我自己的解决方案。

2.2.5 Mybatis

Mybatis 也是老牌的ORM框架之一，但其实现方式却与Hibernate大相径庭，其实现逻辑更像是一个SQL语句的拼装器，开发人员可以直接操控SQL语句，通过各种条件来判断以实现动态地组装SQL语句，并根据映射返回一个包装完成的对象。其实现方式决定了其相较于Hibernate拥有更好的性能，而且较易优化，所以在对性能要求较高的移动互联网领域非常流行。当然，其非全映射的特性也限制了其级联操作的能力，并且开发者即使在工具的帮助下仍然需要花费大量的精力去维护映射关系，这也就是我选择Hibernate作为持久层解决方案的原因之一。

2.2.6 MySQL

MySQL 是一种典型的轻量级关系数据库系统，对于中小型企业网站来说，MySQL可以说是数据持久化的最优解决方案，其开源免费，体积小，性能好，使用成本低等优点深受中小型企业 and 开发者的喜爱。跟其他的关系型数据库系统（Oracle，SQL server）等相比虽然在集群，海量存储上有一定的不足之处，但对于中小企业来说绰绰有余。由于MySQL支持多种数据库引擎，开发者根据其具体需求来对每个表选择相应的数据库引擎，本项目中大多使用InnoDB引擎，其支持的事务机制，外键支持，相对简单的自增长主键是我选择InnoDB引擎的主要原因。

2.2.7 JSP/静态HTML

JSP是一种以Servlet发展而来的动态网页技术。在支持JSP的web容器启动时，容器将根据JSP生成对应的Servlet来对用户请求进行处

理，并将控制层传入的填入其中，最后返回给用户。在早期的web开发中，由于服务器性能，浏览器执行JS脚本的性能，网页渲染速度等多方面因素，JSP的设计思想是非常合理的，随着技术的发展，以及对性能和用户体验的要求，人们注意到JSP的缺点：网页中静态部分占大多数，而Servlet几乎将所有的内容当作动态内容处理，将内容逐行输出，这显然造成了一部分不必要的性能开销，在有高并发的需求下代价非常高昂。

数据模型缺失，在缺乏JS数组或对象作为前端的数据模型是，从前端收集数据就变得非常复杂且不稳定，使得一些常用组件（数据表格等）的实现变得非常复杂。在过去，这种限制使得用户界面往往存在功能有限，需要频繁刷新页面，给用户带来较差的体验。

JSP的调试必须在Web容器中进行，这使得前端开发人员需要一定的JAVA能力，而且给不明确的分工会给项目的开发带来不可预知的风险。

正是由于上面种种局限性，除了老项目依然在使用JSP，目前大部分项目都使用了前后端分离的开发模式，本项目也模拟了这种模式。

2.2.8 Vue.js

对于后端程序员来说，这就是前端版的JSP，二者的逻辑相似到连`{}`标记都是那么亲切，这也是2016年至今最流行的前端MVVM模板框架之一。其中View-model实现的模型视图双向绑定都是以前的JSP无法想象的，基于Vue.js的前端网页只需要通过接口从后端获取数据，然后再修改vue托管的数据模型即可，在下文中我将详细描述其在项目中的应用。

2.2.9 Tomcat

当下流行的轻量级WEB容器，性能稳定并且开源免费，因此十分受个人开发者和小型企业的青睐。对于javaee的初学者来说，其相对简单的架构更适合学习JavaWeb的实现原理。通常来说，Tomcat可以被看作一个控制JSP/Servlet生命周期的容器，使用类似线程生命周期的模式实现了Servlet规范，对于性能要求不高的项目来说，tomcat是相当符合应用需求的，这也是我选用tomcat的主要原因。

2.2.10 Hessian

Hessian是一个轻量级的RMI(远程方法调用)工具，采用二进制协议进行分布式应用间的数据通信，在本项目主要用于用户权限管理系统和租赁管理系统间的接口调用，处理用户登陆验证，菜单树生成等。

2.3 技术选型总结

2.3.1 平衡用户体验以及性能要求

用户体验永远是产品最先考虑的组成成分之一，在实现上主要由两部分组成，其一是产品设计，其二是技术实现，这里只讨论技术实现。由于早期的电脑性能限制以及设计观念，从前的IE内核渲染DOM时使用单线程，这也就使得用户的其对JS操纵DOM的支持非常孱弱，性能低下，也就使得当时的网页大多是静态页面。

随着技术的发展以及谷歌的崛起，Chrome内核的浏览器渐渐占据了市场的半壁江山，其优秀的渲染性能以及稳定性使得单页面富应用的实现成为了可能，其提供的交互体验远胜于从前的IE，因此渐渐获得用户的认可。有了具体的技术支撑，我们开发者当然也要跟进。

为了实现良好的用户体验，本系统抛弃了传统的iframe实现多页面的方案，完全使用JS实现单页面前端路由，并对其中的参数进行统一的管理，相关的技术实现将在后面的章节中阐述。

2.3.2 最终选型

经过多次项目重构后，我最终选择了下面的技术来实现本系统：

后端API/路由实现：SpringBoot(SpringMVC+Spring+Hibernate)。

前端路由实现：Vue.js, js, jquery

前端页面实现：mdui, layui。

WEB容器：Tomcat

持久化方案：MySQL。

远程服务调用：Hessian

以上是项目当前使用的解决方案，具体的业务实现主要使用JAVA以及Javascript。

第3章 系统总体设计

3.1 用户需求分析

经过与客户的沟通后，我们大致得出一下几点用户需求：

3.1.1 订单管理

接单员接到客户的订单之后，将订单详情录入系统，主要包括下单人（工地老板）姓名及电话，工地地址，需要的机型，工程内容，价格，工期等。

财务或老板，通过查询系统即可了解信息，无需一个个过问知情人。

后续工作人员按照订单办事，机手可以按照对应的订单号上报日报，文员找到对应的订单号录入，当订单完工，则按订单生成对应的对账单，找客户对账收款，收到款后，财务再找对应的订单销账。

整个管理系统以订单为主线，严格区分不同订单，机手日报要对应订单号。工地完工并收款销账后，就关闭订单。

3.1.2 客户管理

接单员在录入订单详情的时候也录入了下单员（工地老板）的姓名及电话，相应的工地施工员也录入在他工地老板的二级科目下。也就是查一个工地老板，他名下的施工员全都跑出来；查一个施工员，对应的老板也都跑出来。让工地老板和施工员都联系起来。方便客户管理及保存，并促进收款。

3.1.3 派单管理

每个机手一个手机客户端，机手管理人员将工地老板联系电话，工地地址，机型机号、对应的订单号等信息，通过平台发送给相应的机手。

机手接到信息后，前往指定的地点工作，每天在手机端找到对应的订单号，上报当天工作情况。

财务及相关信息使用人，可以通过系统查看到每个机手、每台机器分别在哪里干活。无需一一询问。

3.1.4 租赁明细管理

每个机手一个手机客户端，直接找到对应的订单号，录入当天工作情况，（或者由一个文员每天通过微信群获取机手汇报的日报，找到对应的订单号录入到系统中）。主要录入的信息有日期、机型机号、土方时间、破碎时间、运费、单据编号、机手等。

系统能根据这些信息再结合订单详情，自动生成对账单，能让我们知道应收金额、工地老板姓名电话及相关的租赁明细。

系统还需要有一个保存图片的功能，要保存的照片就是工时签收单的照片，以备后期查询并用于对账。

3.1.5 对账单打印

工地完工后，系统自动生成对账单，任何人都可以操作打印出对账单。

3.1.6 收款管理

我司工地负责人带着对账单去对账收款，收到款项后，财务根据收款信息核销账款，关闭订单。

结合客户管理功能，系统生成的对账单，只要前期数据录入无误，后期生成的对账单准确率是相对较高的。

收款查询功能要以工地、老板、施工员为单位，搜索出对应的收款信息，如总应收多少，已收多少，优惠多少，未收多少等。方便我们进行收款管理。

3.1.7 固定资产管理

固定资产的管理主要是对每台挖机的入账价值，机器折旧，修理修配等情况的核算，需要随时能查出该固定资产的现值、残值等。
其管理功能主要包括固定资产的新增、修改、退出、转移、删除、借用、归还、维修、计算折旧率及残值率等日常工作。

如果能知道每台机器在哪里状态如何，是否开工运行那就更好了。

3.2 系统架构设计

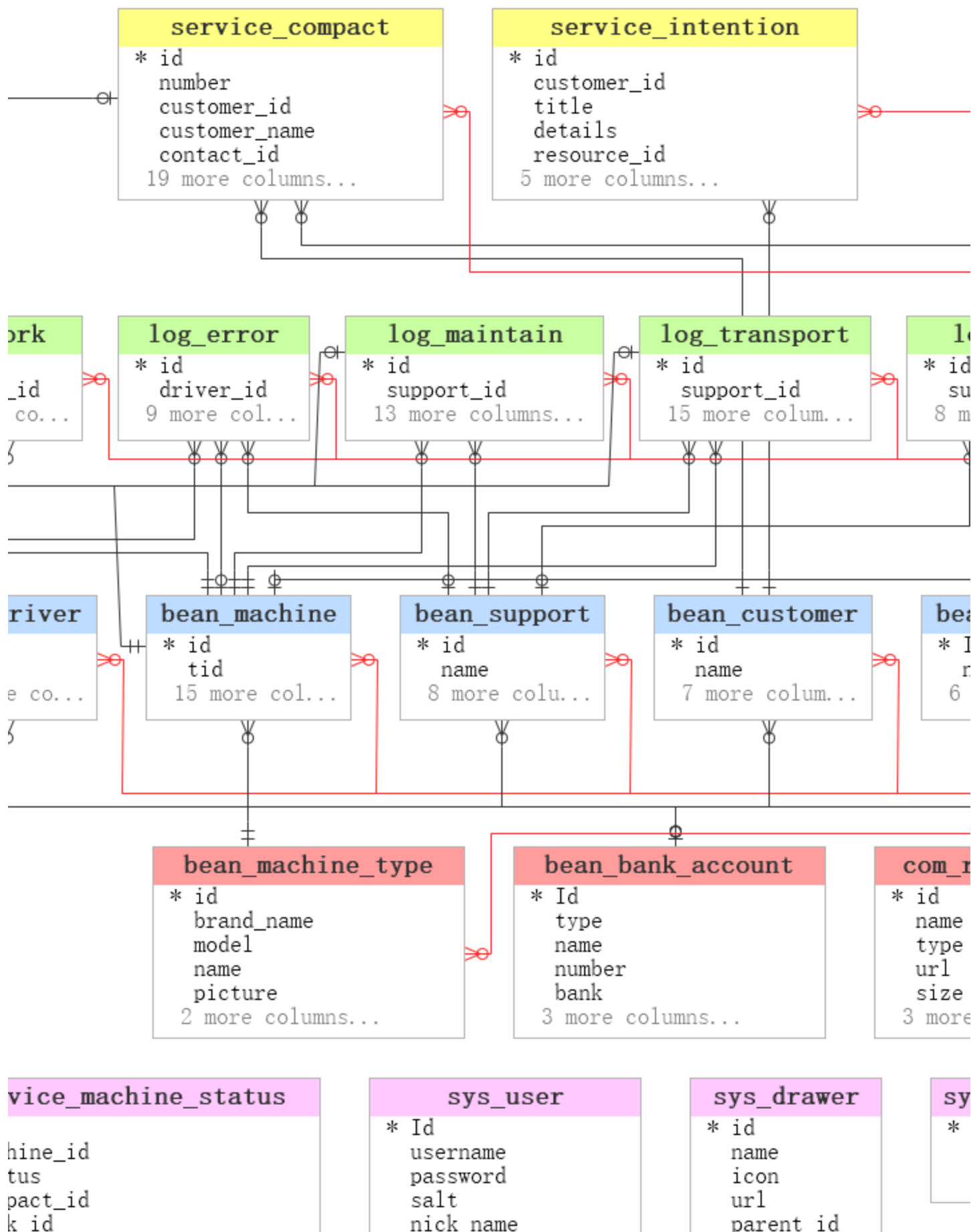


通过上面的需求分析，我将本系统划分为如下图的几个模块：

图3-1 系统架构图

其中，基础，日志中的每个模块都有一套CRUD（增删改查）操作来提供管理功能，业务模块中的挖机地图可以展示挖机当前的地理信息以及状态。资源系统中的模块提供系统内的公共资源（业务文件、银行账号）的支持，模块之间有关联操作以方便查询。

3.3



re columns...

11 more columns...

*

2

transport	com_compact_machine	com_customer_contact	cc
d	* Id compact_id machine_id	* Id customer_id contact_id	*

数据库设计

图3-2 数据库结构设计

考虑到企业ERP系统的复杂程度，本系统在开发之初就对业务流程进行了一定程度的简化，因此本系统并不是全功能的ERP系统，整体更侧重于生产控制管理。我结合客户的实际需求和自己的理解，以及优化实体关系等，将数据库分成了以下几个层次：

角色层：客户信息(bean_customer)，客户联系人(bean_contact)，挖掘机驾驶员(bean_driver)，后勤人员(bean_support)，挖掘机(bean_machine)。

日志层：异常处理(log_error)，加油(log_oil)，维修(log_maintain)，转场(log_transport)，机手工单(log_work)。

业务层：合同(service_compact)，客户意向(service_intention)。

系统层：系统菜单管理(sys_drawer)，系统用户管理(sys_user, sys_role, sys_role_drawer)，挖掘机状态机(service_machine_status)

公共资源层：银行账号(bean_bank_account)，文件系统(com_resource)，挖掘机资料(bean_machine_type)。

此外，数据库中还包含了一些用于处理多对多关系和系统后续升级预留的多对一关系，以及优化数据库性能的中间连接表，它们的表名都是以com开头的，以此表明他们的身份。

3.3.1 bean_customer 客户信息表

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
name	varchar(64)		客户公司名称
type	varchar(11)	普通	客户类型
status	varchar(11)	正常	客户状态
detail	varchar(256)	<空>	客户描述
resource_id	int(11) unsigned	<空>	资源ID
bank_account_id	int(11) unsigned	<空>	银行账户信息
create_time	timestamp	<INSERT-TIMEStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-1 客户信息表

约束：

名称	类型
customer_bank_account	bank_account_id -> bean_bank_account.Id

customer_resource	resource_id -> com_resource.id
-------------------	--------------------------------

表3-2 客户信息表-约束

3.3.2 bean_contact 客户联系人

字段:

名称	类型	默认值	备注
Id	int(11) unsigned	<auto_increment>	主键
name	varchar(30)	<空>	姓名
role	varchar(30)	<空>	角色
tel	varchar(50)	<空>	电话
person_id	varchar(50)	<空>	身份证号码
detail	varchar(255)	<空>	备注
resource_id	int(11) unsigned	<空>	资源ID
bank_account_id	int(11) unsigned	<空>	

表3-3 客户联系人表

名称	类型
contact_bank_account	bank_account_id -> bean_bank_account.Id
contact_resource	resource_id -> com_resource.id

表3-4 客户联系人表约束

3.3.3 客户-联系人连接表

名称	类型	默认值
Id	int(11) unsigned	主键
customer_id	int(11) unsigned	<空>
contact_id	int(11) unsigned	<空>

表3-5 客户-联系人连接表

3.3.4 bean_driver 驾驶员信息

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
name	varchar(16)		机手姓名
tel	varchar(16)		机手联系电话
person_id	varchar(32)		身份证号码
paper_id	varchar(32)	<空>	驾驶许可证号码
status	varchar(11)	空闲	机手状态
type	varchar(11)	普通	机手类型
resource_id	int(11) unsigned	<空>	资源ID
bank_account_id	int(11) unsigned	<空>	银行账号信息
head_pic	varchar(300)	resource	头像地址
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-6 驾驶员信息表

约束:

名称	类型
driver_bank_account	bank_account_id -> bean_bank_account.Id
driver_resource	resource_id -> com_resource.id

表3-7 驾驶员信息表约束

3.3.5 Bean_support 后勤人员信息

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
name	varchar(16)		后勤姓名
tel	varchar(16)		后勤电话
status	varchar(11)	空闲	后勤状态
type	varchar(11)	普通	后勤种类
bank_account_id	int(11) unsigned	<空>	银行账号信息
resource_id	int(11) unsigned	<空>	
head_pic	varchar(200)	<空>	头像地址
create_time	timestamp	<INSERT-TIMEStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-8 后勤人员信息表

约束:

名称	类型
support_bank_account	bank_account_id -> bean_bank_account.Id
support_resource	resource_id -> com_resource.id

表3-9 后勤人员信息表约束

3.3.6 Bean_machine挖掘机信息

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
tid	int(11) unsigned	0	机器型号ID
nameplate	varchar(50)	<空>	铭牌
code	varchar(32)	<空>	机器自编号
owner	varchar(20)	<空>	机主
payway	varchar(20)	<空>	购买方式
price	int(11)	0	参考价格
m_pay	int(11)	0	月供
status	varchar(11)	空闲	状态
worktime	float(7,2) unsigned	<空>	工作时间

xyz	varchar(64)	<空>	GPS坐标
resource_id	int(11) unsigned	<空>	资源ID
detail	varchar(256)	<空>	详细备注
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	
birth	timestamp	0000-00-00 00:00:00	
dead_line	timestamp	0000-00-00 00:00:00	

表3-10 挖掘机信息表

约束:

名称	类型
machine_machine_type	tid -> bean_machine_type.id
machine_resource	resource_id -> com_resource.id

表3-11 挖掘机信息表约束

日志层:

3.3.7 Log_oil 加油日志

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
support_id	int(11) unsigned	<空>	后勤ID
machine_id	int(11) unsigned	<空>	机器ID
litre	double(11,2)	0.00	加油量
money	double(11,2)	0.00	加油金额
details	varchar(256)	<空>	详细描述
resource_id	int(11) unsigned	<空>	资源ID
log_mode	varchar(11)	<空>	日志状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-12 加油日志表

约束:

名称	类型
oil_machine	machine_id -> bean_machine.id
oil_resource	resource_id -> com_resource.id
oil_support	support_id -> bean_support.id

表3-13 加油日志表约束

3.3.8 Log_transport 转场日志

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键

support_id	int(11) unsigned	0	后勤人员ID
support_name	varchar(16)	<空>	后勤人员姓名
support_tel	varchar(16)	<空>	后勤人员电话
machine_id	int(11) unsigned	0	机器ID
departure	varchar(256)	<空>	起点
destination	varchar(256)	<空>	终点
type	varchar(16)	<空>	类型
money	double(11,2)	<空>	转场金额
details	varchar(256)	<空>	描述
resource_id	int(11) unsigned	<空>	资源ID
log_mode	varchar(11)	<空>	状态
working	varchar(11)	<空>	执行状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	
start_time	timestamp	0000-00-00 00:00:00	
end_time	timestamp	0000-00-00 00:00:00	

表3-14 专场日志表

约束:

名称	类型
transport_machine	machine_id -> bean_machine.id
transport_resource	resource_id -> com_resource.id
transport_support	support_id -> bean_support.id

表3-15 专场日志约束表

3.3.9 Log_error 异常处理日志

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
driver_id	int(11) unsigned	<空>	提交机手
support_id	int(11) unsigned	<空>	后勤ID
machine_id	int(11) unsigned	<空>	机器ID
type	varchar(11)	<空>	异常种类
title	varchar(128)	<空>	异常标题
details	varchar(256)	<空>	异常描述
resource_id	int(11) unsigned	<空>	资源ID
log_mode	varchar(11)	<空>	日志状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-16 异常处理日志表

约束:

--	--

名称	类型
error_driver	driver_id -> bean_driver.id
error_machine	machine_id -> bean_machine.id
error_resource	resource_id -> com_resource.id
error_support	support_id -> bean_support.id

表3-17 异常处理日志约束表

3.3.10 Log_maintain 维修记录表

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
support_id	int(11) unsigned	0	后勤人员ID
support_name	varchar(16)	<空>	后勤人员姓名
support_tel	varchar(16)	<空>	后勤人员电话
machine_id	int(11) unsigned	0	机器ID
type	varchar(16)	<空>	维修类型
money	double(11,2)	<空>	维修金额
details	varchar(256)	<空>	维修详情
resource_id	int(11) unsigned	<空>	资源ID
log_mode	varchar(11)	<空>	日志状态
working	varchar(11)	<空>	执行状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	
start_time	timestamp	0000-00-00 00:00:00	
end_time	timestamp	0000-00-00 00:00:00	

表3-18 维修记录表

约束:

名称	类型
maintain_machine	machine_id -> bean_machine.id
maintain_resource	resource_id -> com_resource.id
maintain_support	support_id -> bean_support.id

表3-19 维修记录约束表

3.3.11 Log_work 工单记录

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
driver_id	int(11) unsigned	0	司机ID
driver_name	varchar(16)	<空>	机手姓名
driver_tel	varchar(16)	<空>	机手电话

machine_id	int(11) unsigned	0	机器ID
type	varchar(16)	<空>	类型
money	double	<空>	金额
details	varchar(256)	<空>	描述
resource_id	int(11) unsigned	<空>	资源ID
log_mode	varchar(11)	<空>	状态
working	varchar(11)	<空>	执行状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	
start_time	timestamp	0000-00-00 00:00:00	
end_time	timestamp	0000-00-00 00:00:00	

表3-20 工单记录表

约束:

名称	类型
work_driver	driver_id -> bean_driver.id
work_machine	machine_id -> bean_machine.id
work_resource	resource_id -> com_resource.id

表3-21 工单记录约束表

业务层:

3.3.12 Service_compact 合同表

字段:

名称	类型	默认值	备注
id	int(11) unsigned	<auto_increment>	主键
number	varchar(11)	0	合同编号
customer_id	int(11) unsigned	0	关联客户信息
customer_name	varchar(64)	<空>	
contact_id	int(11) unsigned	0	客户联系人信息
type	varchar(11)	<空>	合同种类
bucket	int(11) unsigned	0	挖斗数量
hammer	int(11) unsigned	0	破碎锤数量
limit_hour	double(11,2) unsigned	440.00	每月限制使用时间
total_hour	double(11,2) unsigned	<空>	实际使用时间
extra_hour	double(11,2) unsigned	<空>	总计超出时间
money_promise	double(11,2) unsigned	<空>	保证金
money_rent	double(11,2) unsigned	<空>	每月租金
money_total	double(11,2) unsigned	<空>	总金额
money_delay	double(11,2) unsigned	<空>	违约金
money_fact	double(11,2) unsigned	<空>	实收金额

money_free	double(11, 2) unsigned	<空>	优惠金额
dest	varchar(256)	<空>	租用地点
resource_id	int(11) unsigned	<空>	资源ID
status	varchar(11)	未审核	合同状态
create_time	timestamp	<INSERT-TimeStamp>	创建时间
edit_time	timestamp	0000-00-00 00:00:00	更新时间
enter_time	timestamp	0000-00-00 00:00:00	进场时间
start_time	timestamp	0000-00-00 00:00:00	开工时间

表3-22 合同表

约束:

名称	类型
compact_contact	contact_id -> bean_contact.Id
compact_customer	customer_id -> bean_customer.id
compact_resource	resource_id -> com_resource.id

表3-23 合同约束表

3.3.13 Service_intention 客户意向登记

名称	类型	默认值	备注
id	int(11)	<auto_increment>	主键
customer_id	int(11) unsigned	0	客户ID
Title	varchar(128)	<空>	标题
details	varchar(256)	<空>	描述
resource_id	int(11) unsigned	<空>	相关资源
service_mode	int(11)	<空>	服务状态
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	
start_time	timestamp	0000-00-00 00:00:00	
end_time	timestamp	0000-00-00 00:00:00	

表3-24 用户意向登记表

约束:

名称	类型
intention_customer	customer_id -> bean_customer.id
intention_resource	resource_id -> com_resource.id

表3-25 用户意向登记约束表

系统层:

3.3.14 Sys_drawer 系统菜单管理

字段:

名称	类型	默认值	备注
Id	int(11)	<auto_increment>	主键

Name	varchar (64)		导航名称
Icon	varchar (128)	<空>	导航图标
url	varchar (128)	<空>	导航URL
parent_id	int (11)	<空>	自关联

表3-26 系统菜单管理表

约束:

Parent_id为自关联字段，存储上级菜单ID，若无上级菜单则为空，用于存储级联菜单。

公共资源层:

3.3.15 com_resource 文件系统

字段:

名称	类型	默认值	备注
Id	int (11) unsigned	<auto_increment>	主键
Name	varchar (128)		文件名称
Type	varchar (16)		文件类型
url	varchar (512)		对应相对路径
Size	int (11) unsigned	0	资源大小
parent_id	int (11)	<空>	父资源ID
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-27 文件系统表

约束:

在文件系统中，没有父节点的节点被视为根节点，当本系统内其他角色（挖掘机驾驶员等）或日志（如加油单）附带其他媒体资源（图片，表格，PDF，视频）时将会创建一个对应的父节点通过树状结构来组织这些文件，并记录他们的存储位置以供下载或编辑。

3.3.16 Bean_bank_account 公共银行账号

字段:

名称	类型	默认值	备注
Id	int (11) unsigned	<auto_increment>	主键
Type	varchar (50)	<空>	账户类型
Name	varchar (50)	<空>	开户名
number	varchar (50)	<空>	账号
Bank	varchar (100)	<空>	开户行
Detail	varchar (255)	<空>	描述
create_time	timestamp	<INSERT-TimeStamp>	
edit_time	timestamp	0000-00-00 00:00:00	

表3-28 公共银行表

描述:

公共表，存储着各个角色的银行账户信息。

3.3.17 Bean_machine_type 机器图鉴

字段:

名称	类型	默认值	备注
Id	int(11) unsigned	<auto_increment>	主键
brand_name	varchar(32)	<空>	机器品牌
model	varchar(32)		型号
Name	varchar(50)	<空>	通俗名称
picture	varchar(300)	<空>	型号照片
Detail	varchar(256)	<空>	详细描述
resource_id	int(11) unsigned	<空>	资源ID

表3-29 机器图鉴表

描述：

存储基本的机器图鉴及其详细信息 待定。

第4章 系统实现

4.1 后端核心组件

在前文中，我已经介绍了本项目选择SpringBoot作为后端解决方案的原因。作为前后端分离的项目，本系统借鉴了以node.js为后台的微信小程序的设计思想，结合本人对前端页面交互逻辑的理解，形成了如下图的后台架构：

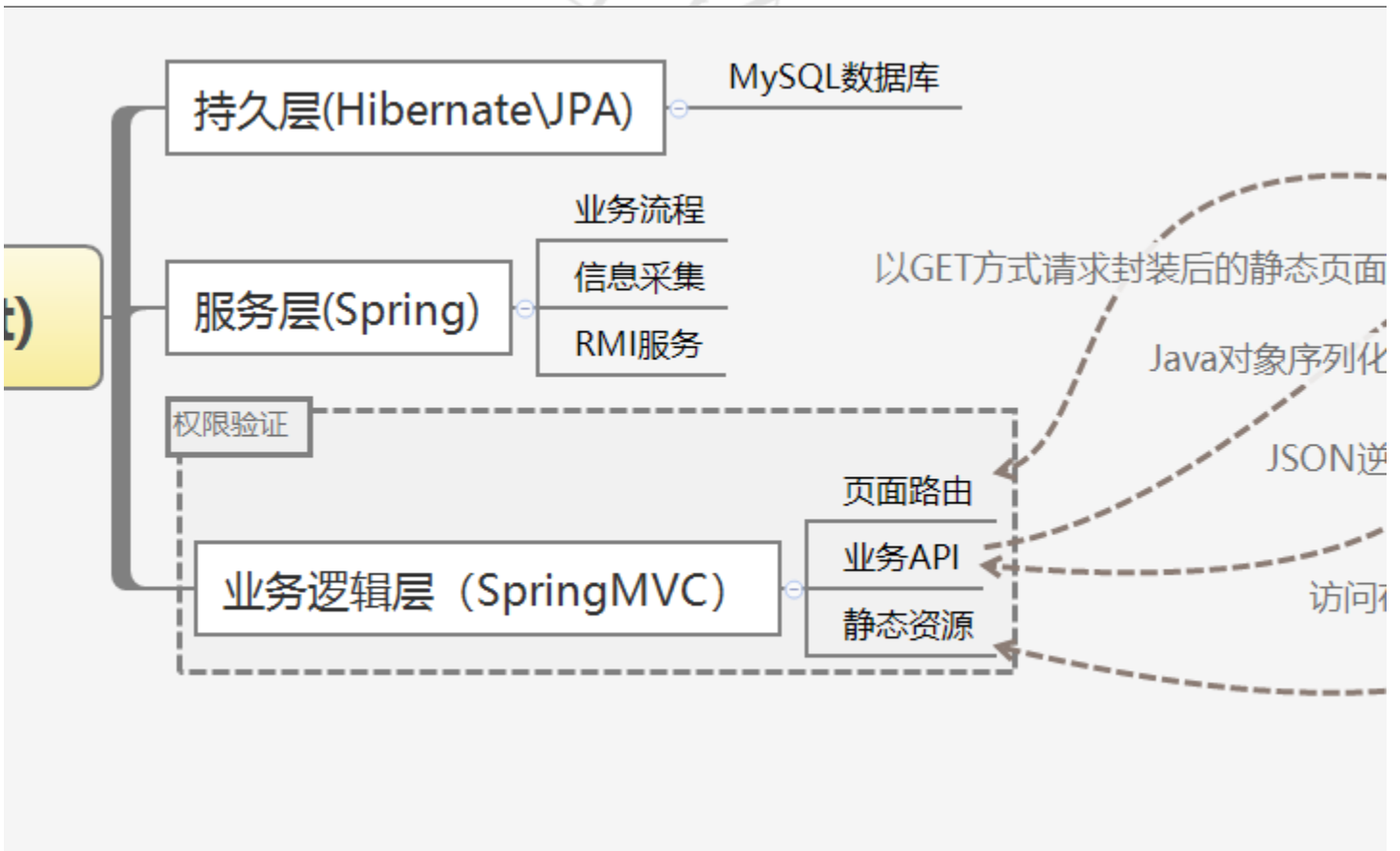


图4-1 后台架构图

下面将根据此图来逐步解析系统结构。

4.1.1

图4-2 数据实体

- 1) 实体的每个字段都拥有对应的Getter, Setter方法用于组装并重写了toString方法以方便调试。
- 2) JPA规范中的@Entity和@Table注解表示了其是一个实体, 并且指定了实体对应的数据库中的表。
- 3) @Id和@GeneratedValue指定主键字段以及其自增长特性。
- 4) @OneToOne 和 @ManyToOne 描述了实体间的关系, @JoinTable 指定了一个有连接表的多对一关系。
- 5) @Cascade配置了级联操作, 可以看到我给Resource和bankAccount字段配置了级联操作, 代表我无需单独处理resource和bankAccount字段的持久化问题, Hibernate会处理。

4.1.2 持久层 (DAO)

```
sean
```

```
ace JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>
findAll();

findAll(Sort var1);

findAllById(Iterable<ID> var1);

; T> List<S> saveAll(Iterable<S> var1);

();

; T> S saveAndFlush(S var1);

;eInBatch(Iterable<T> var1);

;eAllInBatch();

[ID var1);

; T> List<S> findAll(Example<S> var1);

; T> List<S> findAll(Example<S> var1, Sort var2);
```

DAO层负责实现实体类和数据库表间的关系。为此 Spring在Spring Data中提供了一个非常优秀的解决方案JpaRepository接口以及动态代理技术，我们先来看JpaRepository接口中定义了什么

图4-3 JPA模板仓库

如图，该接口继承至定义了分页逻辑的repository以提供分页功能，而且还具有模板查询的能力。对于开发者而言，只需要继承这个接口，并不需要实现。

```
public interface ContactDAO extends JpaRepository<Contact,Integer>
    public List<Contact> findAllByCustomerId(Integer id);
}
```

图4-8 DAO层

同时，开发者还可以按照JPA规范定义新的方法来进行快速查询。接口并不需要实现，在Spring初始化时，Spring会根据泛型内的类以及方法名生成对应的动态代理类，并装入对应的引用中，开发者并不关心SQL语句应该如何编写，这也是JPA规范存在的意义。

系统中所有的DAO层都继承了这个接口, 以此来完成整个DAO层的开发工作。

4.1.3 通讯模型

```
public class CommonResponseModel<T> {
    private String msg;
    private String code;
    private T t;
```

为了协调各个层面间的通讯以及前后端之间的通讯, 我设计了三种通讯模型 CommonResponseModel、Page4Navigator、RequestModel, 下面将一一介绍他们的用处。

图4-10 标准请求模型

```
public class RequestModel<T>{
    private T example;
    private int start;
    private int pageSize=10;
    private int navigatorSize;
```

标准返回模型, 包括了msg (信息) code (返回码 200 成功 500 失败) t (载荷)。这个模型和HttpResponse报文有相似之处, 主要用于向前端返回单个查询结果, 如根据id从数据库中查询一个对象, 执行更新操作后返回的对象等。、

图4-11 标准请求模型

```
public class Page4Navigator <T>{
    Page<T> pageFromJPA;
    int navigatePages;
    int totalPages;
    int number;
    long totalElements;
    int size;
    int numberOfElements;
    List<T> content;
    RequestModel<T> example;
    String responseStatus;
    String responseMessage;
    boolean isHasContent;
    boolean first;
    boolean last;
    boolean isHasNext;
    boolean isHasPrevious;
    int[] navigatepageNums;
```

标准请求模型, 前端根据该模型组装JSON, 在处理请求时Spring-MVC将会将其逆序列化为标准请求模型的实例, 其中的example用于

存放查询模板或者需要保存的单个数据等，start、pagesize、navigatorSize则存储分页请求信息。

图4-12 分页数据模型

Page4Navigator是对JPA规范的Page对象的封装,在dao层继承了JpaRepository接口后进行分页查询时返回的就是Page对象，其包含了当前页数，结果总数，是否是第一页，是否是最后一页等分页信息，我在其中加入了前端发送的requestModel模型。前端的数据表格中的数据都由这个模型提供。

这三个模型贯穿了服务层，业务逻辑层和前端，为模块间的通信制订了标准。

4.1.4 服务层 (Service)

```

@Service {

    @Autowired
    CustomerDAO;

    @Autowired
    CustomerDAO;

    Pageable listByExample(Customer customer,int start,int size,
        Sort(Sort.Direction.DESC, ...properties: "id"));
    Pageable= PageRequest.of(start,size,sort);
    ExampleMatcher=ExampleMatcher.matching()
        .onMatcher( s: "name",ExampleMatcher.GenericPropertyMatchers.contains()
        .onMatcher( s: "bankAccount.name",ExampleMatcher.GenericPropertyMatchers.contains()
        .onMatcher( s: "bankAccount.number",ExampleMatcher.GenericPropertyMatchers.contains()
    Customer> example=Example.of(customer,exampleMatcher);
    Pageable=customerDAO.findAll(example,pageable);
    Page4Navigator<Customer>(pageJPA,page);

    CustomerDAO update(Customer customer){
        save(customer);
        Page4Navigator<Customer> result=new Page4Navigator<>();
        result.content=new ArrayList<>( initialCapacity: 1);
        result.content.add(customer);
        return result;
    }
}
    
```

在系统的设计之初，我为每个实体都设计了对应的service层实现，他们都实现了listByExample(示例查询)和update(保存/更新)两个方法。同时由于因为逻辑相对比较复杂，所以有多种实现方式，下面先以CustomerService为例进行说明：

图4-13 客户服务层

可以看到，listByExample的参数为customer（装载了匹配条件的实体）、start(开始序号)、size（分页页面大小）、page(分页器大小)。其中包含的局部变量有，Sort(JPA规范中指定排序规则的工具类)、Pageable（分页过后的数据模型）ExampleMatcher（匹配规则），Example(匹配规则+示例=匹配器)。准备了以上几个模型后，就可以通过继承了JpaRepository的DAO层查询到封装好的page对象，再将这个page对象封装成Page4Navigator对象来进行通讯。

在Update方法中，我们使用customerDAO实现的save方法来进行持久化，并将持久化后的对象封装进一个不包含分页数据的Page4Navigator对象，以此实现各层间数据通信。

```
public Page4Navigator<LogTransport> listByExample(LogTransport logTransport, int start, int size, int page){
    Sort sort=new Sort(Sort.Direction.DESC, "id");
    Pageable pageable=PageRequest.of(start,size,sort);
    Specification<LogTransport> specification=new Specification<LogTransport>() {
        @Override
        public Predicate toPredicate(Root<LogTransport> root, CriteriaQuery<?> criteriaQuery, CriteriaBuilder cb) {
            List<Predicate> ps = new ArrayList<>();
            if(logTransport.getSupport()!=null){
                if(logTransport.getSupport().getName().trim().length()>0){
                    ps.add(cb.like(root.join( "support", JoinType.INNER).get("name"), "%"+logTransport.getSupport().getName()+"%"));
                }
            }
            if(logTransport.getMachine()!=null){
                if(logTransport.getMachine().getNameplate().trim().length()>0){
                    ps.add(cb.like(root.join( "machine", JoinType.INNER).get("nameplate"), "%"+logTransport.getMachine().getNameplate()+"%"));
                }
            }
            if(logTransport.getFromTime()!=null&&logTransport.getToTime()!=null){
                ps.add(cb.between(root.get("createTime"),logTransport.getFromTime(),logTransport.getToTime()));
            }
            if(logTransport.getLogMode()!=null&&logTransport.getLogMode().trim().length()>0){
                ps.add(cb.equal(root.get("logMode"),logTransport.getLogMode()));
            }
            if(logTransport.getType()!=null&&logTransport.getType().trim().length()>0){
                ps.add(cb.equal(root.get("type"),logTransport.getType()));
            }
            Predicate p[]=new Predicate[ps.size()];
            criteriaQuery.where(cb.and(ps.toArray(p)));
            return criteriaQuery.getRestriction();
        }
    };
    Page page=PA=logTransportDAO.findAll(specification,pageable);
    return new Page4Navigator<LogTransport>(page,PA,page);
}
```

如果有更加复杂的查询需求，我们除了继承JpaRepository以外，还需要继承JpaSpecificationExecutor以支持更复杂的多例查询，下面以LogTransportService(转场日志服务)为例来进行说明：

图4-14 复杂查询

在这个listByExample方法中，我实现了一个Specification接口中的toPredicate方法来组织多例查询的查询条件，如查询一个日期范围内的实例，该方法同样返回一个Page4Navigator对象。

4.1.5 业务逻辑层 (Controller/filter)

ontroller

```
ublic class PageRouteController {
```

```
@GetMapping(value = "/login")
public String login() { return "admin/frame/login"; }
```

```
@GetMapping(value = "/phoneLogin")
public String phoneLogin() { return "phone/frame/login"; }
```

//前台路由

```
@GetMapping(value = "/phone")
public String phone() { return "phone/frame/phoneFrame"; }
```

//后台路由

```
@GetMapping(value = "/admin")
public String adminHome() { return "admin/frame/adminFrame"; }
```

在前文我们提到，本系统的业务逻辑层被划分成三个部分，页面路由（PageRouteController），业务API（RestController），静态资源（Static Resource），这里主要介绍页面路由和API。

图4-15 后端路由

```
("/admin_drivers_query")
Navigator<Driver> list(@RequestBody RequestModel<Driver> requestModel) {
    Navigator<Driver> page=driverService.listByExample(requestModel.getExample(requestModel));
    .out.println(page);
    page;
}

("/admin_drivers_update")
Navigator<Driver> update(@RequestBody RequestModel<Driver> requestModel) {
    Navigator<Driver> page=driverService.update(requestModel.getExample());
    ResponseMessage("保存成功!");
    page;
}

"/admin_driver_info")
r findById(@RequestParam(value = "id") int id) { return driverService.findById(id);
}

("/admin_driver_export")
nResponseModel<Driver> export(@RequestBody RequestModel<Driver> requestModel) {
    SimpleDateFormat df = new SimpleDateFormat(pattern: "yyyy年MM月dd日HH时mm分ss");
    fileName="机手信息导出_"+df.format(new Date())+".xls";
    targetURL="resource/cacheFiles";
    targetPath=request.getServletContext().getRealPath(targetURL);
    getFolder=new File(targetPath);
    getFile=new File(targetFolder,fileName);
    if(!getFile.getParentFile().exists()){
        getFile.getParentFile().mkdirs();
    }

    L=targetURL+"/"+fileName;
    driverService.exportByExample(requestModel.getExample(),targetFile.getPath());
    ResponseModel<Driver> responseModel=new CommonResponseModel<>();
    responseModel.setCode("200");
    responseModel.setMsg(targetURL);
    return responseModel;
}
```

上图展示的是页面路由，其实是对get请求的封装，使前端不直接请求html页面，用于后期限制权限。

图4-16 机手查询

上图则是DriverController的主要代码，该控制器的类型为RestController，接受JSON数据并返回JSON数据，可以看到，控制器实际上是对service层所提供的服务的调用，list方法对应service的listByExample方法，update对应service的update方法，这一层主要

功能为对请求参数的解析以及返回结果的封装。

4.1.6 权限拦截器

```
requireLogins = new String[] {
    "login"

    request.getRequestURI();
    Jutils.remove(uri, remove: contextPath + "/");
    uri = uri;
    th(page, requireLogins)) {
        user = (User) session.getAttribute(s: "User");
        if (user == null) {
            // 认证失败
            String responseText = "<div style='text-align:center;width:100%;height:100px;'>
                <h1>认证信息失效！ 将在三秒后返回登陆界面</h1>\n" +
                "</div>\n" +
                "<script>\n" +
                "\tfunction skipToLogin(){\n" +
                "\t\twindow.location.href = 'login';\n" +
                "\t}\n" +
                "\tvar t=window.setTimeout(skipToLogin,3000); \n" +
                "</script>";
            response.setCharacterEncoding("utf-8");
            response.setContentType("text/html; charset=utf-8");
            PrintWriter out = response.getWriter();
            String charset = response.getCharacterEncoding();
            out.print(responseText);
            return false;

            // 授权失败
            StringWith(page,user.getAuthUrls())){
                String responseText = "授权失败！ 目标: "+page;
                response.setCharacterEncoding("utf-8");
                response.setContentType("text/html; charset=utf-8");
                PrintWriter out = response.getWriter();
                String charset = response.getCharacterEncoding();
                out.print(responseText);
                return false;
            }
        }
    }
}
```

拦截器是实现权限控制的主要方式之一，下面简单阐述登录拦截器的主要逻辑作为例子

图4-17 权限拦截器

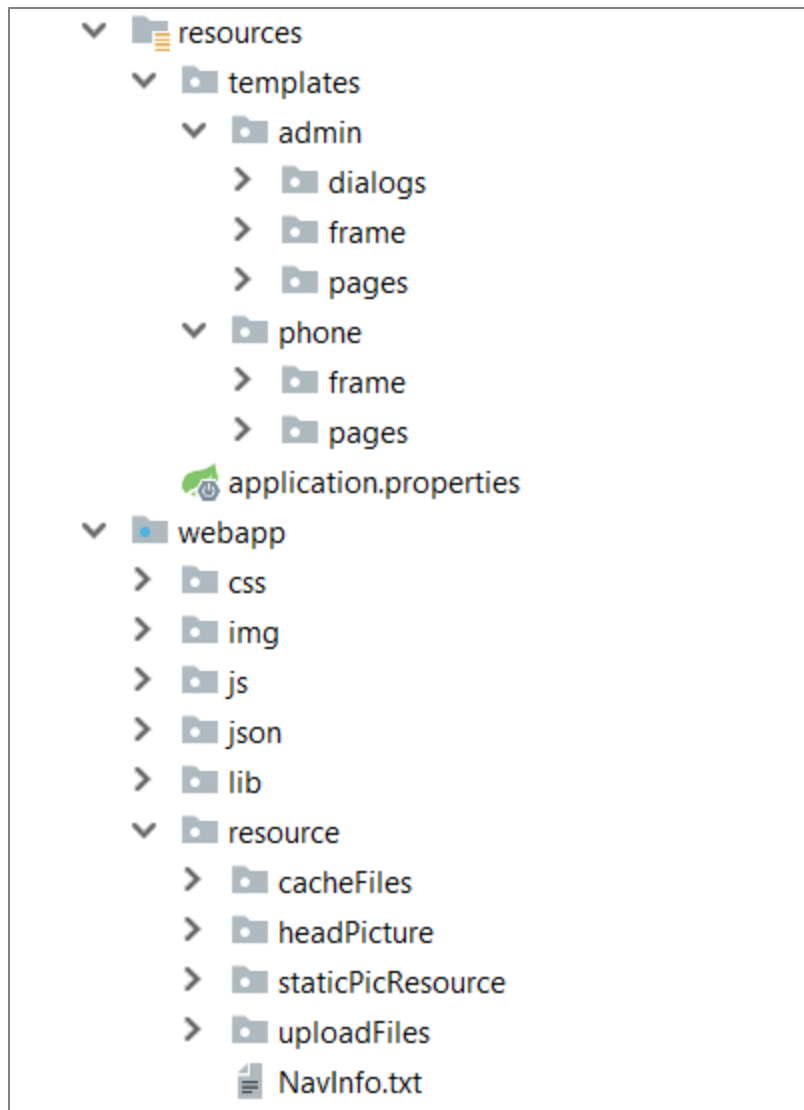
SpringBoot中实现拦截器相对较为简单，在上图中我们可以看到，拦截器实际上是在业务逻辑层处理请求之前对请求进行预处理，对符合要求的请求予以放行。

在登陆拦截器中，我们首先对所有admin开头的请求进行处理，如果用户未登陆，则返回一个可以定时跳转到登陆界面的网页，如果用户已经登陆，则校验用户当前是否拥有访问该URL的细分权限（用户所拥有的细分权限数据由UAMS用户权限管理系统提供，在4.4章节中有描述），如果有，则将请求转发给对应的contro-ller中的方法，如果没有则返回权限限制信息。

4.2 前端核心组件及模板

系统界面总体采用谷歌推出的Material Design设计风格，其界面类似安卓操作系统风格，搭配单页面设计能给用户良好的体验。

4.2.1 前端静态文件目录结构



前端目录结构借鉴了微信小程序的目录结构，项目共用一个CSS文件，html文件都有相同文件名的js文件维护页面逻辑，通过AJAX实现动态加载来提升性能和用户体验。

图4-18 前端文件结构

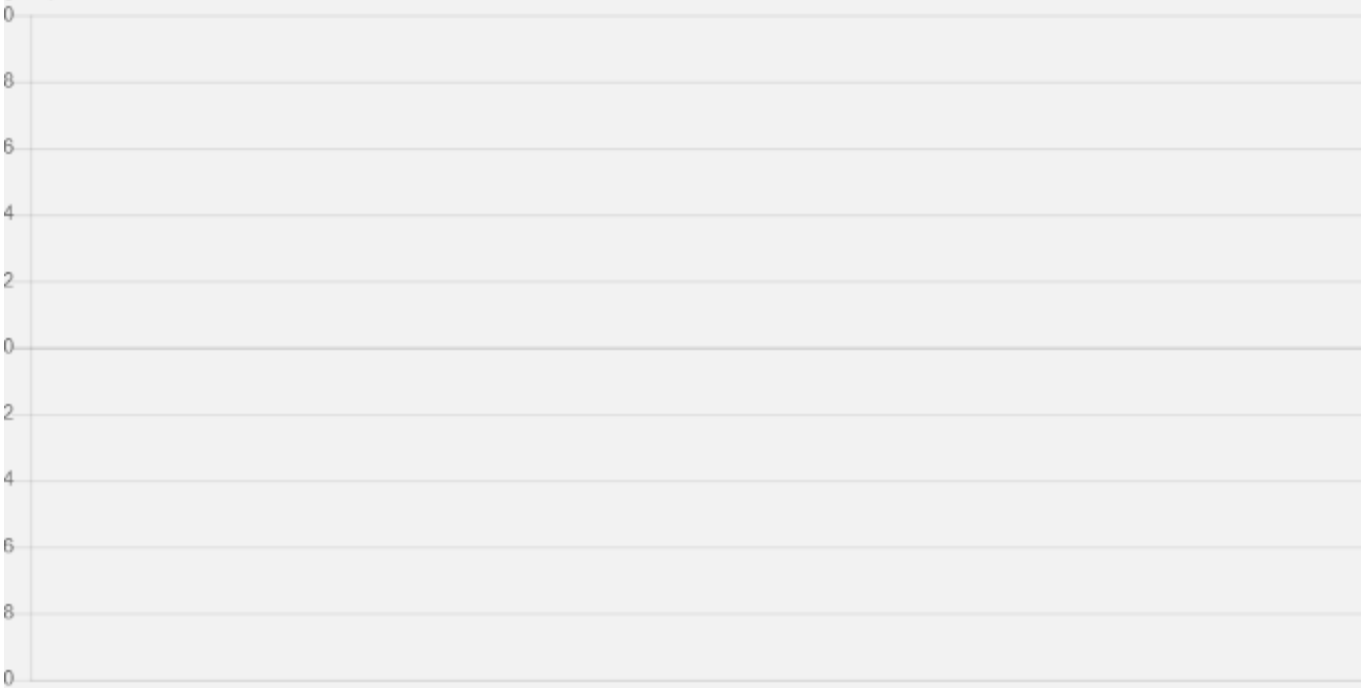
如图所示，文件主要分为移动端和PC端网页两部分，每部分都包含了Dialog组件，frame主体框架，page子页面模板等目录。值得一提的是，所有的子页面模板都不包含完整的HTML结构（只包含一个外部的DIV），外部资源的引入都在frame框架中进行统一管理

4.2.2



单

订单量



主体框架设计

图4-19 界面主体框架

页面框架主要由三部分组成：左侧抽屉导航栏，顶部状态栏以及标签栏，右下角子页面容器。下面将介绍其具体实现方法：

左侧抽屉导航：HTML的布局方式为嵌套的ul标签，并通过vue.js中的v-for标签遍历一个二维数组渲染而成，该二维数组通过后端API获取，在数据库中的具体数据结构为一个带自关联外键字段的表，可以存储树状结构。

```
function makeTab(tabId) {
    layui.use('element', function() {
        var element = layui.element;
        if (NAV_OPENED.indexOf(tabId) == -1) {
            var navTab = findTabInfoById(tabId);
            $.get(BASE_ROUTE_URL+navTab.url, function(data) {
                element.tabAdd('model', {
                    title: navTab.name,
                    content: data,
                    id: tabId
                });
                NAV_OPENED.push(tabId);
                element.tabChange('model', tabId);
            });
        } else {
            element.tabChange('model', tabId);
        }
        mdui.mutation();
    });
}
```

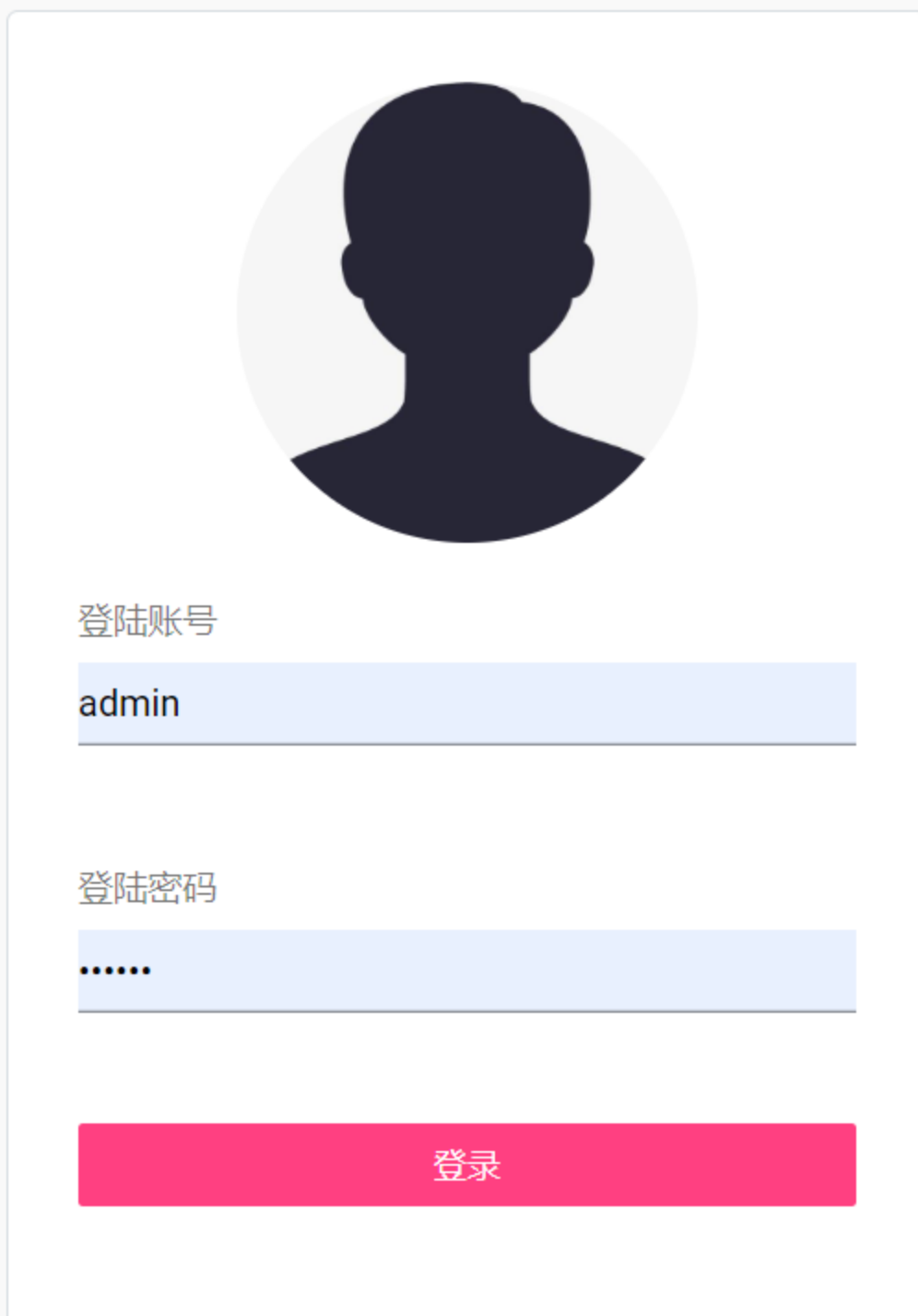
顶部标签栏：顶部标签的标签项为ul，每个li对应一个div容器，通过获取被触发click事件的li标签中的id来查找对应的div容器并改变其display属性来控制子页面的显示和隐藏。每个页面都有其唯一ID, 以此来实现左侧抽屉导航和标签栏之间的联动。具体实现如下图

图4-20 标签切换逻辑

每个左侧抽屉都关联了对应的页面初始化信息，当触发了点击事件时将从服务端申请模板进行渲染以及初始化工作，如页面在标签栏内未关闭，则切换到特定的页面。

子页面容器：子页面容器将填充剩下的屏幕空间，所有已经加载的子页面将存放在容器中，容器将自动适应用户的缩放比例变化，并且限制变量作用域。

广州铠信挖掘机租赁管理系统



The image shows a user login interface for the 'Guangzhou Kaixin Excavator Rental Management System'. It features a central white card with a light gray border. At the top of the card is a circular placeholder for a user profile picture, represented by a dark blue silhouette. Below this, the text '登陆账号' (Login Account) is followed by a light blue input field containing the text 'admin'. Further down, the text '登陆密码' (Login Password) is followed by a light blue input field filled with six black dots. At the bottom of the card is a prominent pink button with the white text '登录' (Login).

除了以上三点以外，页面框架还包括一个登陆界面

图4-21 用户登陆界面

登陆界面的逻辑并不复杂，为了缩减篇幅，在此不做过多描述。

4.2.3 级联选择弹窗注册器


```
<div id="machineChooser"></div>
<div id="supportChooser"></div>
<div id="driverChooser"></div>
<div id="customerChooser"></div>
<script>
    $("#machineChooser").load(DIALOG_MACHINE_CHOOSER);
    $("#supportChooser").load(DIALOG_SUPPORT_CHOOSER);
    $("#driverChooser").load(DIALOG_DRIVER_CHOOSER);
    $("#customerChooser").load(DIALOG_CUSTOMER_CHOOSER);
</script>
```

在dialogs目录下有一个名为dialogRegister.html的文件，文件内容如下图

图4-22 级联选择弹窗注册

在初始化框架时，该文件将被动态加载并执行其中的JS脚本来加载全局的选择弹窗，在需要进行级联选择的时候将弹出如下图的窗口

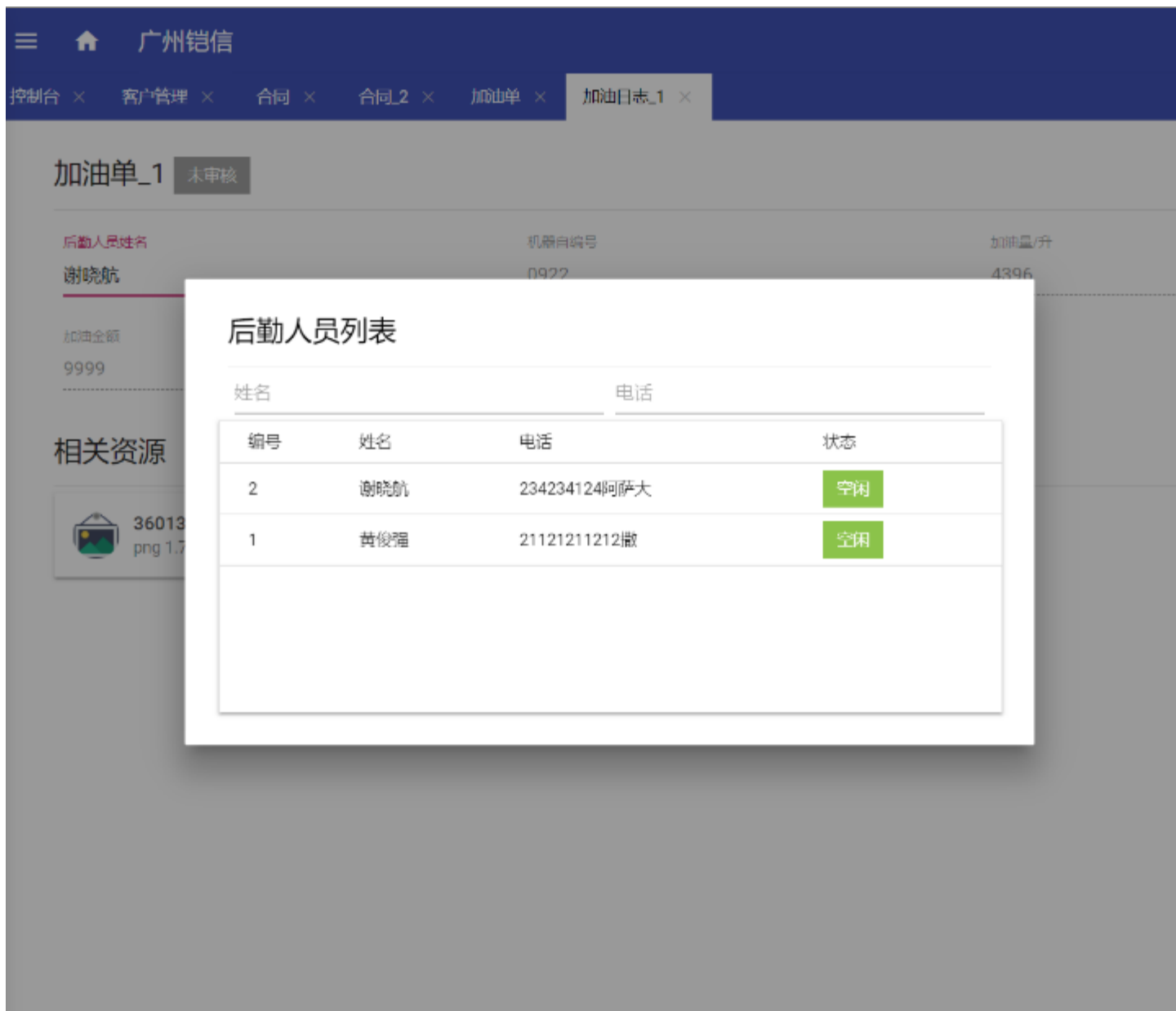


图4-23 弹窗界面

这个弹窗包含了一个快速的条件筛选，如后勤人员列表中可以通过后勤人员姓名，电话来对结果进行筛选，选择的项会填入输入框中来起到限制输入的作用。可以看到一共有四个级联选择弹窗，分别用于选择机器，后勤人员，机手，客户的快速选择，在此不多做展示。

4.2.4



文件管理组件

图4-24 文件管理器

在做需求分析时，我了解到客户当前依然需要各种照片，视频等来做凭证和业务支撑，如合同扫描件，对账单，发票信息等，因此需要一个公共的文件系统来实现资源共享，具体实现在后端核心中有描述。

如上图，每个模块的详情页面都会提供相关资源的下载，在编辑状态下提供删除，上传，重命名等管理功能，以此来实现一个完整的文件资源管理能力。

4.2.5 标准数据表格

广州铠信

制台 × 客户管理 × 机器管理 × 合同 × 机手管理 ×

筛选条件

客户名称

客户名称

甲方联系人姓名

甲方联系人姓名

租用地点

甲方联系人姓名

合同状态

合同状态

合同类型

所有合同类型

开始时间

1999-12-26 10:16:00

结束时间

2019-03-08 08:29:35

查询

ID	合同编号	客户名称	客户联系人姓名	合同种类	合同状态	创建时间
2	2019011401	Dustone科技有限公司	陈岩	临租合同	未审核	2019-0
1	201901101	Dustone科技有限公司	陈岩	月租合同	未审核	2019-0

新增

导出

共计2项

1

数据表格是整个系统最核心的组件，也是用户交互最密集的组件，下面将以合同模块的数据表格举例说明

图4-25 标准数据表格

```
Vue({
  el: '#driverQueryTable',
  data: {
    trs: [],
    pagination: {},
    requestModel: {
      start: 0,
      example: {
        type: "",
        status: ""
      }
    },
    pageNow: 0
  },
  mounted: function() {
    driverQueryVue = this;
    this.list(0);
  },
  methods: {
```

如上图所示，本系统的数据表格主要由三个部分组成：条件筛选（模糊查询，精确查询，时间范围查询等）、表格本体（数据，操作栏）、工具栏（分页，新增，导出，统计）。HTML格式在此不做展示，下面将介绍其JS逻辑，为了控制篇幅，我在此仅以机手管理为例展示Vue对象的组成成分。

图4-26 数据表格逻辑1

如上图所示，在VUE对象中 el对应的为指定数据表格的DOM的ID属性，data对应为vue对象的数据，其中 trs数组存储着表格中每一行的数据，pagination为后台获取到前端的数据，requestModel为前端和后端的标准请求模型，筛选区域的数据将直接填入这个模型并返回后台查询，其中包括了分页信息，筛选条件，返回码等，pageNow则为当前所在页数，为分页组件提供数据支撑。

Mounted是vue的回调方法之一，用于配置在DOM渲染完成后的操作，可以配置初始化流程等，接下来将介绍VUE的第二个部分 method

```
function(start) {
  ~ self = this;
  lf.requestModel.start = start;
  ~ ajaxRequestModel=DEEP_COPY(self.requestModel);
  ajaxRequestModel.example=checkEmpty(ajaxRequestModel.example);
  ios.post(PAGE_DRIVER_QUERY, ajaxRequestModel).then(function(response)
    self.pagination = response.data;
    self.trs = response.data.content;
    self.pageNow = response.data.number+1;
    mdui.mutation();
  );

function(page) {
  ~ self = this;
  mp(page, self); //定义在adminHeader.html 中

Number: function(start) {
  ~ self = this;
  mpByNumber(start, self);

  function() {
    ~ self = this;
    bleQuery(0, self);
```

图4-27 数据表格逻辑2

上图是methods的第一部分，这部分定义了该vue对象具体的业务逻辑，可以通过监听DOM的事件来调用，其中list为基本查询方法，该方法用于收集查询条件，并通过vue官方推荐的axios来完成ajax请求，并使用返回的数据对vue对象的数据模型进行更新，由于MVVM的架构的特性，vue将自动完成渲染工作。下面的三个方法jump、jmpByNumber\ query 为分页逻辑的实现。

```

showInfo: function(tr) {
    var tabMsg={
        id:"driverInfo"+tr.id,
        eId:"driverInfoAera",
        eNum:tr.id,
        name:'机手信息_'+tr.name,
        url:ROUTE_DRIVER_INFO,
        target:'DriverInfo',
        initData:{
            entity:tr
        }
    }
    makeInfoTab(tabMsg);
},
addEntity: function(){
    var tabMsg={
        id:"driverInfo",
        eId:"driverInfoAera",
        eNum:"ADD",
        name:'新增机手',
        url:ROUTE_DRIVER_INFO,
        target:'DriverAdd',
        initData:{}
    }
    makeInfoTab(tabMsg);
},
newSelected: function(num) {

```

图4-28 表格逻辑3

```

elected: function(num) {
  ar self = this;
  f (num == self.pageNow) {
    return {
      'mdui-color-theme-accent': true
    }
  }
  else {
    return {
      'mdui-color-theme-accent': false
    }
  }
}

tData:function () {
  ar self=this;
  ar ajaxRequestModel=DEEP_COPY(self.requestModel);
  ajaxRequestModel.example=checkEmpty(ajaxRequestModel.example);
  axios.post(PAGE_DRIVER_EXPORT, ajaxRequestModel).then(function(res) {
    if(response.data.code=="200"){
      var $form = $('<form method="GET"></form>');
      $form.attr('action', response.data.msg);
      $form.appendTo($('body'));
      $form.submit();
      $form.remove();
    }else{
      mdui.snackbar({
        message: "文件导出失败!",
        timeout: 2000,
        position: "right-bottom",
      });
    }
  });
};

```

这是methods的第二部分，主要实现两个逻辑。在本系统中，数据的编辑和新增都是同一个模板，因此需要不同的初始化参数来进行初始化，并调用frame中的makeInfoTab方法创建新的标签页，makeinfotab方法跟之前介绍的maketab方法有所不同，他会使用初始化参数的target参数来使用反射调用详情页面对应的初始化方法，进行详情/新增页面的初始化工作。

图4-29 数据表格逻辑4

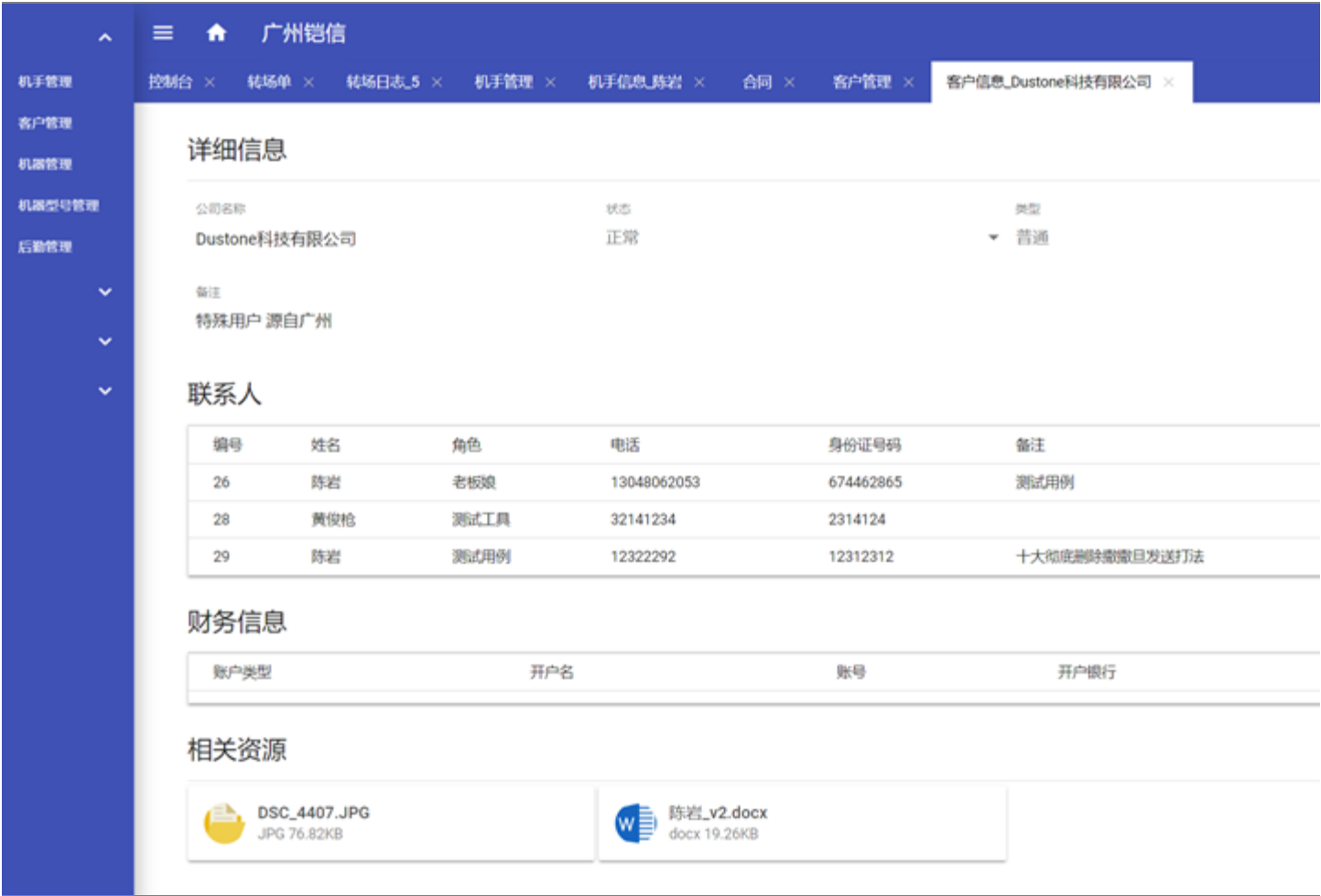
上图为methods的第三部分，其中pageSelected为一个过滤器，用来改变分页组件中当前页的CSS样式，exportData为导出按钮对应的方法，其将当前的筛选条件传入后端API中，后端生成一个XLS文件，收到返回结果后再新建一个表单下载xls文件到浏览器实现数据导出。

至此，一个数据表格基本完成，使用到的第三方组件为laydat-e，数据表格使用的CSS样式主题为 mdui 的默认风格。

4.2.6 查看/编辑/新增/审核 四合一组件

对于本系统来说，大致分为两种组件，日志类组件和普通组件，二者的区别主要在于日志组件需要有审核功能，这里就以客户信息管理和转场日志为例来对组件进行说明。

图4-30



详情页面

如上图所示，客户信息主要有详细信息，联系人，财务信息，相关资源等组成。值得一提的是，大部分数据的显示区域其实都是输入框，通过对disabled状态的输入框添加特殊的样式来实现浏览效果，并使用v-bind命令控制输入框的启用禁用，我们接下来对js逻辑进行进一步的解析。

图4-31

```
new Vue({
  el: '#customerInfoAera',
  data: {
    title: "",
    entity: {
      bankAccount: {},
    },
    contacts:[],
    editPanelModel: "none",
    viewPanelModel: "inline",
    inputDisabled: true,
    requestModel: {
      start: 0,
      example: {
        id: 0,
      }
    }
  },
  mounted: function () {
    customerInfoVue = this;
  },
}
```

详情页面逻辑1

```

        (tabMsg) {
            this;
            / = tabMsg.initData.entity;
            entity.bankAccount == null) {
                entity.bankAccount = {};

            contactList();
            title = '详细信息';

            on (tabMsg) {
                this;
                editPanelModel = "inline";
                viewPanelModel = "none";
                inputDisabled = false;

                function () {
                    this;
                    entity.resource == null) {
                        newResource = {
                            name: self.entity.name,
                            type: "trunk",

                        post(RESOURCE_UPDATE, newResource).then(function (resp) {
                            self.entity.resource = resp.data.t;
                            self.requestModel.example = self.entity;
                            $.post(PAGE_CUSTOMER_UPDATE, self.requestModel).then(function (r
                                self.entity = response.data.content[0];
                                openFileManagerDialog(self.entity, "客户信息");
                            );

                        FileManagerDialog(self.entity, "客户信息");

```

首先是vue对象的数据部分，title字段用于实现编辑/浏览下标题的更改，entity是实例数据（其中的bankaccount只是占位字段，用于视图预处理） contacts存储着客户名下的全部联系人信息（将通过一个额外的ajax请求获取

） editPanelModel, viewPanelModel, inputDisabled则用于实现编辑/查看的视图切换，requestModel为标准请求模型。

图4-32 详细页面逻辑2

接下来是methods部分，init和initAdd方法顾名思义，用于初始化新建模式和查看模式，初始化参数来自于上文数据表格中提到的初始化参数，将模块初始化成不同的状态有利于提高代码复用率和软件性能。

```

tion (mode) {
this;
1) {
tPanelModel = "inline";
wPanelModel = "none";
utDisabled = false;
ode == 2) {
tPanelModel = "none";
wPanelModel = "inline";
utDisabled = true;

questModel.example = self.entity;
st(PAGE_CUSTOMER_UPDATE, self.requestModel).then(function (respon
.entity = response.data.content[0];
self.entity.bankAccount == null) {
self.entity.bankAccount = {}};

l.snackbar({
message: response.data.responseMessage,
timeout: 2000,
position: "right-bottom",

firm("所有的更改无效!", "提示", function () {
.editPanelModel = "none";
.viewPanelModel = "inline";
.inputDisabled = true;

.requestModel.example = {};
.requestModel.example.id = self.entity.id;
.requestModel.start = 0;
s.post(PAGE_CUSTOMER_QUERY, self.requestModel).then(function (res

```

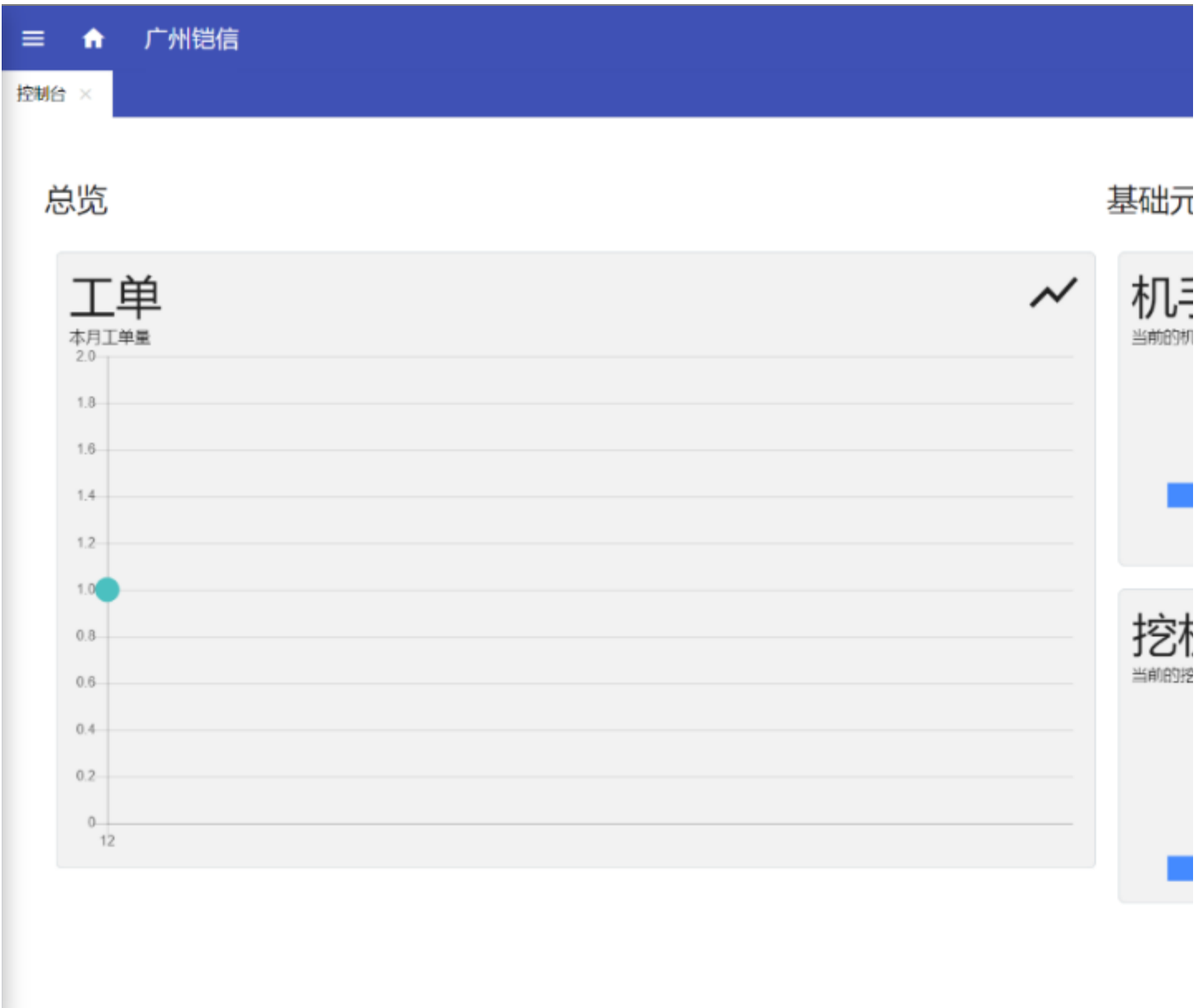
openFileManager方法则可将当前数据模型装载进文件管理组件进行管理操作，关于文件管理组件前文已经提到过，在此不再赘述。

图4-33 详情页面逻辑3

右上角编辑/保存/放弃的按钮逻辑，处理编辑按钮，使用模式标识符来管理切换，其中mode==1时为编辑模式 mode==2时为保存，当mode为其他值时视为放弃保存，此时会弹出确认框询问用户操作，以防止误操作。

带审核功能的组件与编辑的逻辑大致相同，在此省略。

4.2.7



概览页面

图4-34 概览页面原型

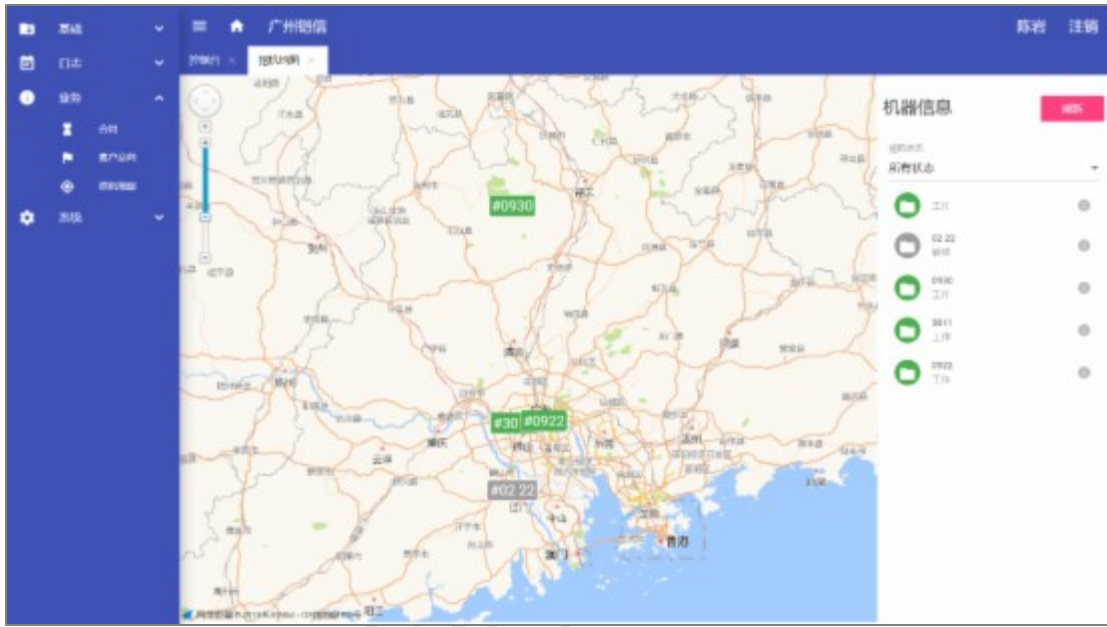
对于业务支撑系统来说，业务情况概览相当重要，快速的掌握企业运营状况是常见的需求之一。本系统给出了预选的解决方式，实际需求仍需要和客户进一步的对接。

如上图，这是由三个图标组成的页面，其中有两个甜甜圈图和一个折线图，分别反应当前的机手工作状态分布，挖掘机工作状态分布，每日的工单数量也可以反映当前公司的运营情况。完整的功能将在新一轮需求分析后继续开发。

本组件用到的第三方库为Chart.js图标库，通过后端API获得的数据在经过格式化后将用于绘制图表。

4.2.8 挖机地图（基于高德地图API）

资产定位对于设备租赁行业的企业来说至关重要，在经过多方



面的考虑后，我最终选择了高德地图API作为实现方式，其较高的免费额度足以满足小型系统的使用需求，并且在后期的开发中对货车（挂车）的路线规划支持更友好，由于目前暂未拿到客户原先购买的资产定位服务的数据接口，因此暂时使用虚拟坐标代替

图4-35 挖机地图

如上图，面板左侧为地图面板，上面的图示分别表示挖掘机的自编号，图示的颜色为挖机当前的状态，右侧为挖掘机列表，可以提供筛选操作，后续将添加更多功能。

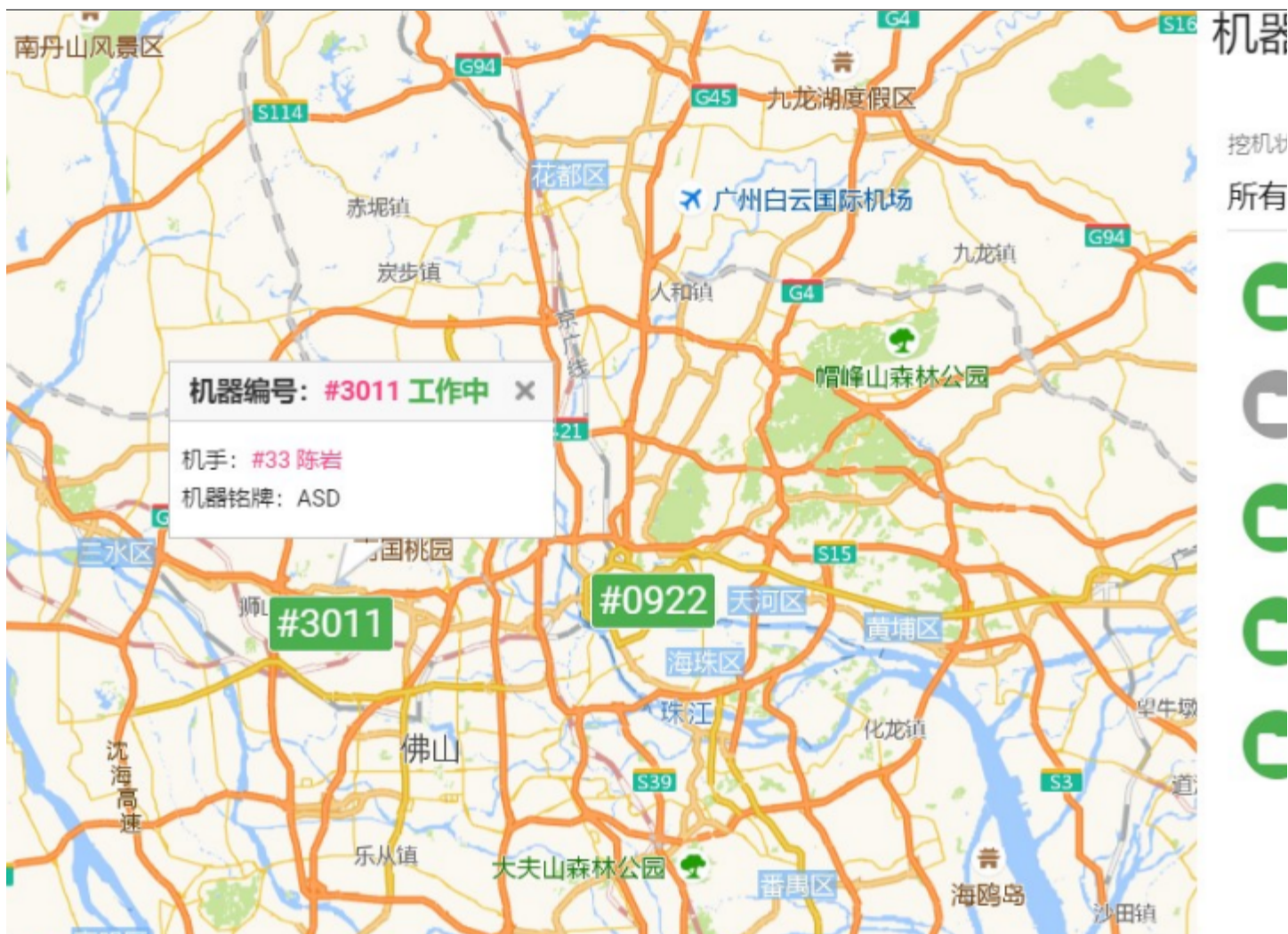


图4-36 挖机地图详情

单击图示后即可查看详细信息，如当前的驾驶员，订单号等，后续将增加更多信息和功能，如转场路径规划，机器调度等高级功能。

4.2.9 前端异常处理

```
on configAxiosInterceptors(){
ios.interceptors.response.use(function (response) {
  if(response.headers["content-length"]==253){
    if(response.headers["content-type"]=="text/html;charset=utf-8"){
      mdui.snackbar({
        message: "验证信息失效! 三秒后返回登陆界面!",
        timeout: 3000,
        position: "right-bottom",
      });
      function skipToLogin(){
        window.location.href = 'login';
      }
      var t=window.setTimeout(skipToLogin,3000);
    }
  }
  return response;
function (error) {
  // Do something with response error
  return Promise.reject(error)
}
```

和传统的web应用不同，异常处理一直是前后端分离系统的一大难点之一，大量的AJAX请求使得通常用于处理异常的客户端跳转命令不再有效，在前端实现拦截器机制成为了最完善的解决方案之一。本系统利用axios支持配置拦截器的特性来实现了对特殊response包的异常处理。如下图所示。

图4-37 异常处理逻辑

拦截器将对符合匹配length=253 且content-type=text/html的response进行处理，当功能API出现异常时，将弹出提示信息并进行跳转，当路由API出现异常时，将路由的页面替换为异常提示，并提供三秒后跳转至登陆界面的能力。

4.3 数据库优化

在进行技术选型时，我意识到了ORM框架选择Hibernate后由于hibernate本身设计思想带来的局限性，如有时不必要的全映射，查询语句合并产生的N+1问题等，接下来将阐述我当前对这个问题的解决思路。

4.3.1 将视图作为实体映射

code	compact_id	compact_type	compact_number	customer_id	customer_name
	(Null)	(Null)	(Null)	(Null)	(Null)
	(Null)	(Null)	(Null)	(Null)	(Null)
	2	临租合同	2019011401	34	Dustone科技有限公司
	(Null)	(Null)	(Null)	(Null)	(Null)
	(Null)	(Null)	(Null)	(Null)	(Null)

由于hibernate的全映射特性，一旦查询都是将所有字段全部查出，即使我们只需要一两个字段的內容，这无疑是对数据库性能资源的极大浪费，使用延迟加载又会对数据完整性产生威胁。对此，我决定将视图作为表进行映射。

图4-38 数据库表结构

DEFINED

line_status`

```
line_status`.`Id` AS 'id',
line_status`.`machine_id`,
line_status`.`status`,
e`.`nameplate` AS 'machine_nameplate',
e`.`code` AS 'machine_code',
line_status`.`compact_id`,
compact`.`type` AS 'compact_type',
compact`.`number` AS 'compact_number',
compact`.`customer_id`,
compact`.`customer_name`,
line_status`.`work_id`,
driver_id`,
driver_name`,
driver_tel`,
line_status`.`transport_id`,
ct`.`support_id` AS 'transport_support_id',
ct`.`support_name` AS 'transport_support_name',
ct`.`support_tel` AS 'transport_support_tel',
line_status`.`maintain_id`,
r`.`support_id` AS 'maintain_support_id',
r`.`support_name` AS 'maintain_support_name',
r`.`support_tel` AS 'maintain_support_tel',
line_status`.`wgs_x`,
line_status`.`wgs_y`,
line_status`.`gcj_x`,
line_status`.`gcj_y`,
line_status`.`update_time`

e_machine_status`
`bean_machine` ON ((`service_machine_status`.`machine_id` = `bean_machine
`service_compact` ON ((`service_machine_status`.`compact_id` = `service_c
`log_work` ON ((`service_machine_status`.`work_id` = `log_work`.`id`)))
`log_transport` ON ((`service_machine_status`.`transport_id` = `log_trans
`log_maintain` ON ((`service_machine_status`.`maintain_id` = `log_maintai
```

视图也被称为虚表，可以视为预定义的查询过程，在语句中不支持传入参数，但可以被视为一张表进行where条件查询，这里以挖机地图为例，他的数据源是一个视图：

图4-39 挖机状态视图

这个视图是对机器表 (bean_machine)、挖机状态表 (machine_status)、合同表 (bean_contract)、机手表 (bean_driver) 进行的多表连接查询，其定义如下：

然后将查询结果中拥有唯一约束的字段作为主键映射为entity实体交给Hibernate托管，通过这种方式既不破坏Hibernate的封装性，又能优化查询中使用到的字段数量，同时也避免了Hibernate合并SQL语句时产生的1+N问题使得数据库引擎花费大量的资源在开启关闭查询上。

4.3.2 使用触发器维护冗余字段

```
CREATE TRIGGER `maintain_support_trigger` BEFORE INSERT
ON `bean_support`
FOR EACH ROW
BEGIN
    IF NEW.support_name IS NULL THEN
        SET NEW.support_name = NEW.support_tel;
    END IF;
END
```

为了尽量减少级联查询所带来的开销，开发人员通常会对一些经常查询但不需要更改的字段进行冗余设计，这里我们以转场日志为例介绍触发器在系统中的应用

图4-40 冗余字段触发器

这里的后勤人员姓名 (support_name) 和后勤人员电话 (support_tel) 是经常需要用到的数据，用户在对专场日志进行追踪时也需要现场人员的联系方式，因此我们使用触发器在插入转场日志前获取后勤人员信息插入到冗余字段中。

4.4 用户权限管理系统 (User Authority Management System)

跟大多数Web应用一样，权限管理是必不可少的一部分，通过对不同用户的权限进行管理，来维持网站的稳定运行，防止数据被有意或无意的破坏。

由于需求和条件的差异性，大家实现权限管理的方式也是五花八门，也产生了大量优秀的权限验证框架 (SpringSecurity, Shiro 等) 展现了其设计思想。我们可以根据部署的方式将他们划分为两类，其一是作为一个子模块整合进系统中，其二是独立成为一个单独的授权系统。二者各有其特点，子模块简单易维护，独立系统可拓展性强。

在与客户进行需求对接后，我发现了一下几点需求：

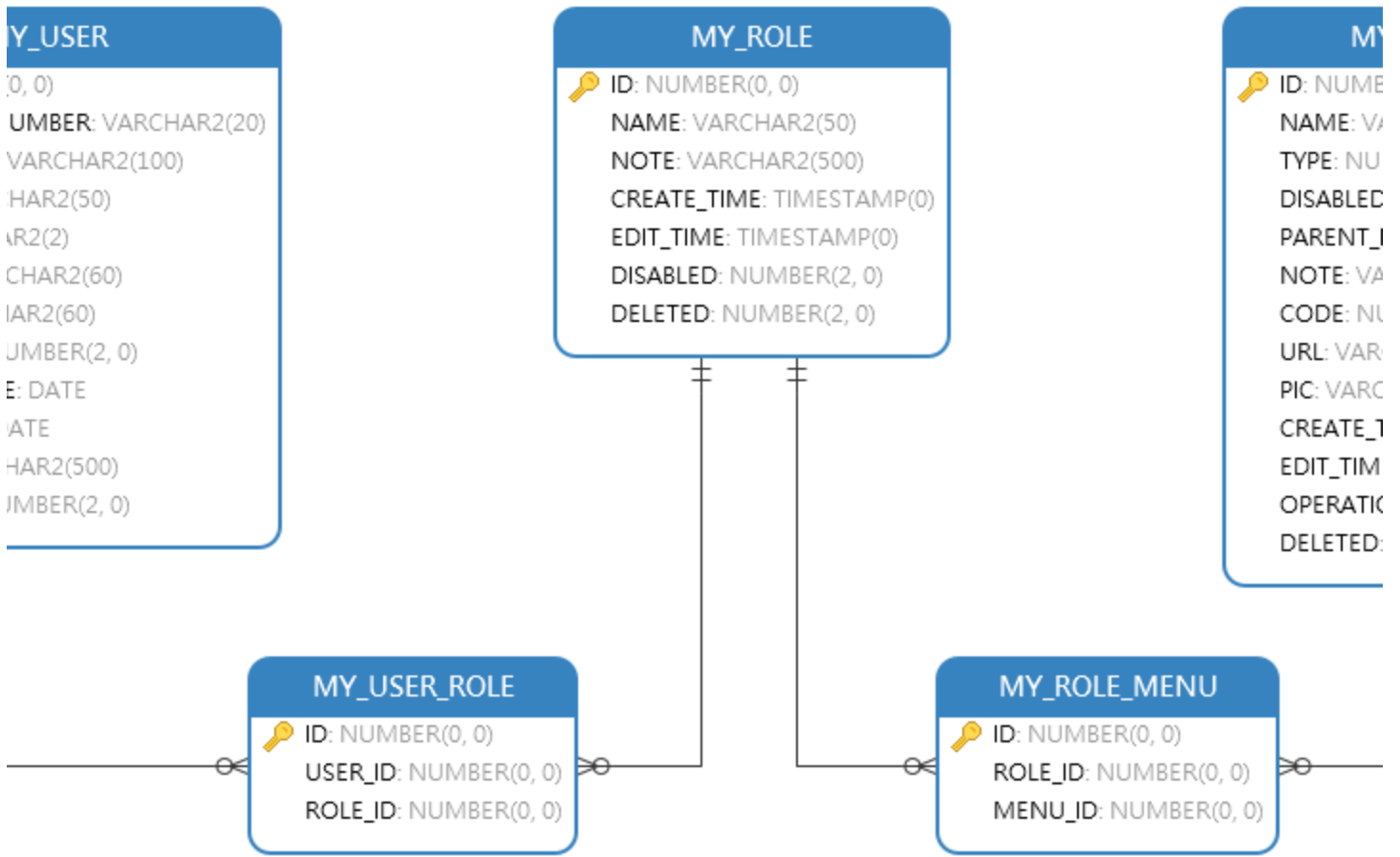
- 1) 支持使用相同的账户和密码登录到不同的系统中，账户集中式管理，并且可以在不同的系统中为不同的角色。
- 2) 支持跨域身份验证。
- 3) 支持账号登录日志，账号禁用等功能。

出于以上几点需求，我决定将权限验证作为一个独立的系统来实现，为了控制篇幅，论文中仅阐述系统的设计以及界面原型。

4.4.1 技术选型

本系统跟租赁管理系统稍有不同，数据库使用的是Oracle，而ORM层使用的是Mybatis，产生差异的主要原因是我希望完成毕设的同时通过实习单位的技术考核。在项目正式上线前我会将数据库迁移至Mysql以降低使用成本。

4.4.2



数据库设计

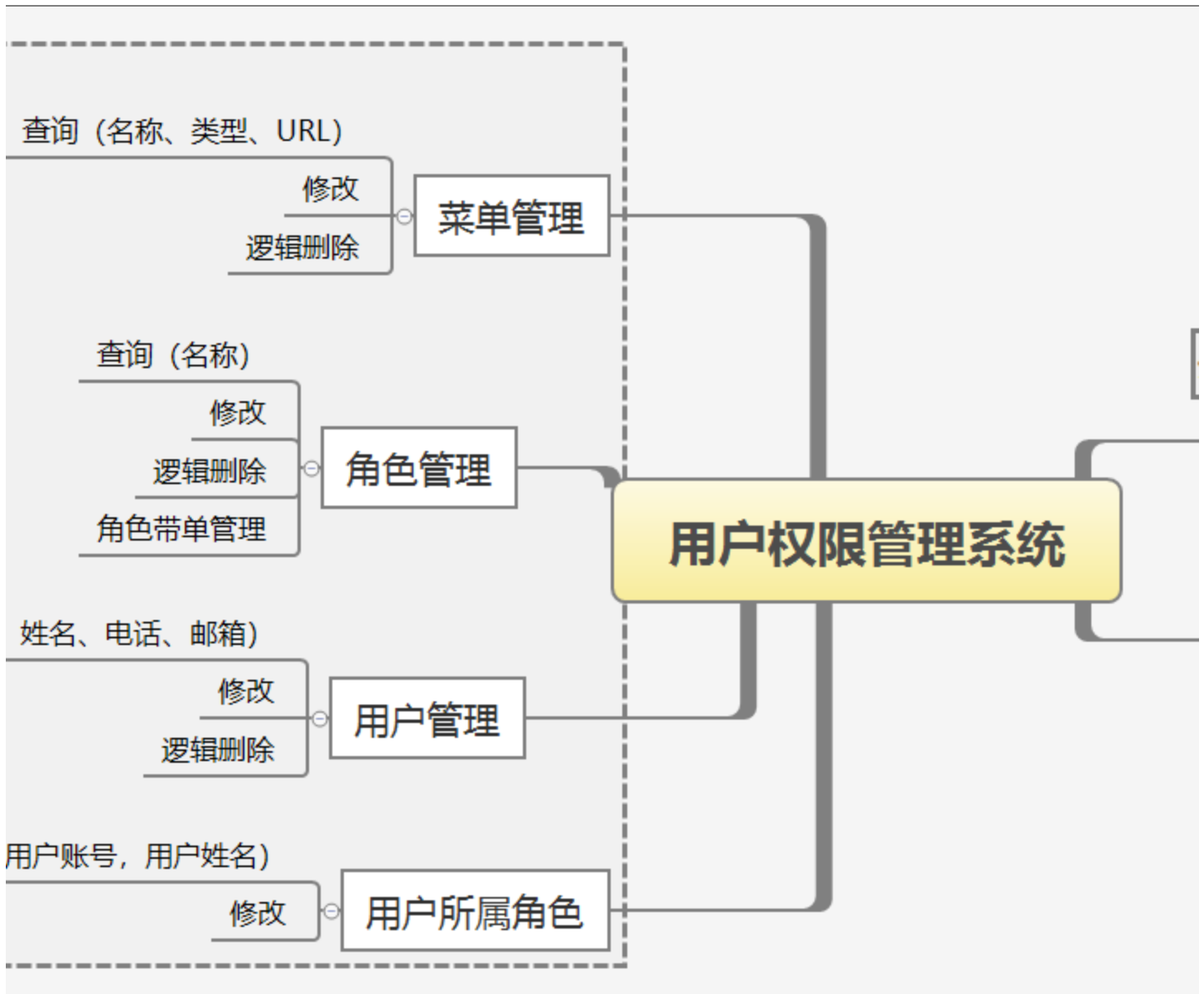
图4-41 用户权限管理系统UML图

如上图，MY_USER(用户表)和MY_ROLE(角色表)使用中间表进行多对多关联，即一个用户拥有一个或多个角色，MY_ROLE和MY_MENU(菜单表)使用中间表进行多对多关联，一个角色可以拥有一个或多个菜单的访问权限，菜单表中有PARENT_ID自关联字段，用于存储上级菜单的主键以存储菜单树状结构。

除连接表外，所有的表都有edit_time 和create_time来记录修改时间和创建时间，disabled字段来表明当前数据对外部系统是否可见，deleted字段用于标记逻辑删除（出于系统稳定性考虑，所有的元数据都使用逻辑删除）note字段用于保存备注信息。

4.4.3 系统架构

图4-42



用户权限管理系统架构

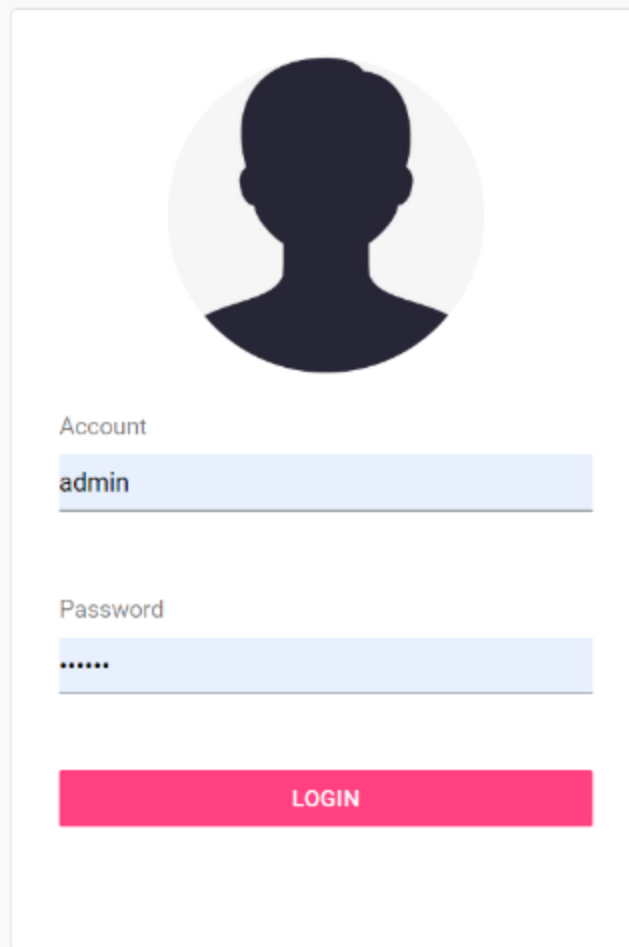
可以看到，系统分成了两部分，其中内部功能以Web应用的方式提供给系统管理员使用，外部接口为使用Hessian暴露给外部应用（这里主要为挖掘机租赁管理系统）调用的接口，以此实现用户授权以及租赁管理系统的菜单生成。

4.4.4 页面设计

用户权限管理系统的大部分组件都和租赁管理系统相同，在此只阐述不同的地方。

1)

User Authority Management System



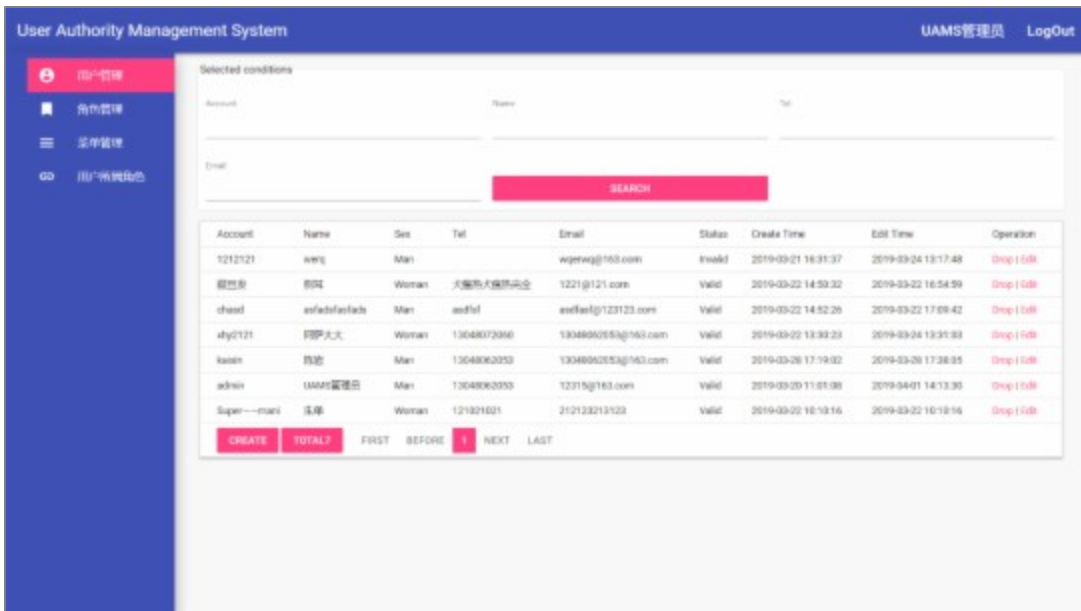
The image shows a login form for the 'User Authority Management System'. It features a dark blue silhouette of a person's head and shoulders inside a light gray circle. Below this, there are two input fields: 'Account' with the text 'admin' and 'Password' with masked characters '*****'. At the bottom is a red 'LOGIN' button.

登陆界面/国际化支持

图 4-43 用户登录界面/国际化

由于公司方面的技术要求，我使用vue-i18n插件配合自定义字典在前端页面实现了全局的中英文切换，用户可在登陆界面单击右上角的按钮来进行切换，在用户切换后将自动插入一条cookie以在用户下次访问系统时有更好的体验。

2)

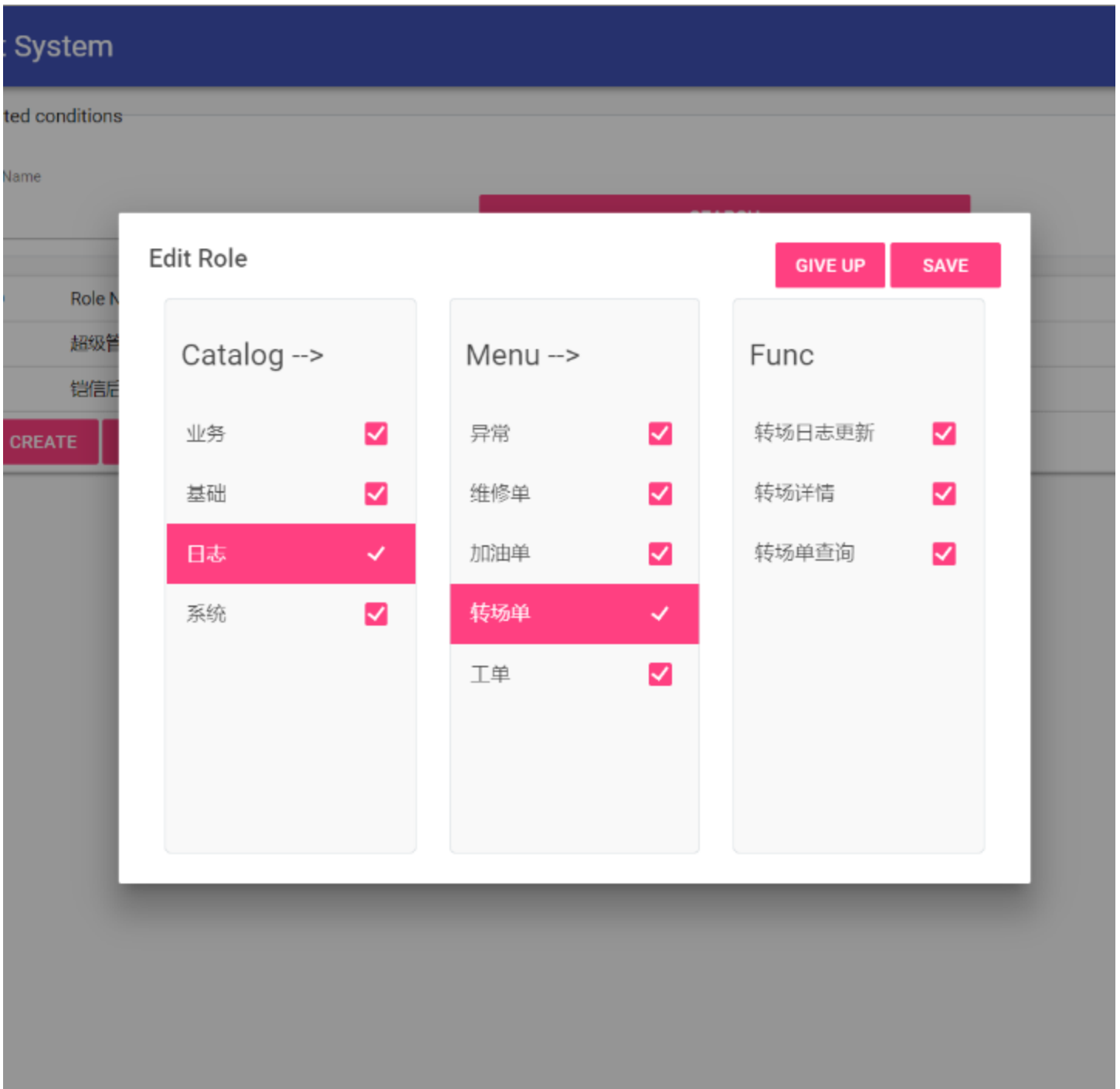


标准数据表格

图 4-44 标准数据表格

我对标准数据表格进行了重新布局，将表格控制相关的组件（新增，总数，分页）放在了表尾的固定位置，防止数据造成的布局异常，修改了背景样式使得表格由悬浮的效果，提升用户体验。

3)



角色菜单管理弹窗

图 4-45 角色-菜单管理弹窗

在角色-菜单管理弹窗中，我加入了一个三级级联多选列表，用于展现目录/菜单/功能的三级树状结构，其中 目录/菜单中选定的项将用于生成用户菜单，功能项则对应了角色的细分权限，如对机手信息的增删改查权限，这也是权限管理系统存在的意义，在进行这项设置后，所有拥有该角色的用户都将拥有这些权限。

4)



用户所属角色弹窗

图 4-46 用户所属角色

用户所属角色相对简单，为一个多选列表，勾选即代表用户拥有这个角色，一个用户可以拥有多个角色以登陆多个系统。

4.4.5 外部接口设计


```
ic interface UserAuthorityService {
public User userAuth(User user,String targetRoleName);
public List<SimpleMenuInfo> getMenuTreeByRoleId(int role
```

外部接口的作用在前文描述过，通过Hessian暴露给租赁管理系统调用，其定义了以下两个方法

图 4-47 用户授权服务接口定义

其中，userAuth方法接受一个包含了用户登陆时输入的登陆账号以及密码的User对象，以及登陆的目标角色，不同的系统调用该方法时都会指定特定的目标角色，以验证用户是否拥有该角色。getSimpleTreeByRoleId方法接受一个角色的ID，并返回角色所拥有的菜单树。

用户授权流程：

- 1) 校验用户登陆密码、账号，是否与数据库中的数据库匹配，不匹配则返回错误提示用户名或密码错误。
- 2) 校验当前用户是否拥有目标系统指定的角色，不匹配则返回用户没有登陆该系统的权限！。
- 3) 根据当前用户的角色生成角色所拥有的菜单访问白名单，包装后作为接口的返回值。

菜单树生成流程：

- 1) 根据传入的角色ID找出角色所拥有的一级和二级菜单（目录和菜单）。
- 2) 将查出来的菜单信息按照字段parentId(上级ID)组装成目录树，作为接口的返回值。

第5章 总结与期望

5.1 系统的缺陷以及后续升级计划

众所周知，完美的系统并不存在。每个系统在成长阶段都会遇到需求变更，前期架构不合理等问题。下面将对我在当前开发中遇到的问题进行总结，并给出了未来解决问题的方向。

5.1.1 业务逻辑支撑

由于经验不足，在项目设计的初期业务逻辑支撑的问题并没有引起我足够的关注。我的大部分精力都集中在实体属性的分析中，而轻视了对实体状态的控制，使得实体状态和属性经常被集中在一张表中，这也就造成了属性和状态的高度耦合。

在后续的开发中，我充分地体会到这种耦合带来的灾难性后果：业务控制逻辑和实体属性维护逻辑混淆在一起。业务逻辑只能依靠大量重复的判断来实现，大量的代码混杂在一起，这无疑为后续的升级维护带来了巨大障碍，更改业务逻辑和实现新的逻辑都需要修改大量的代码。

由此可见，逻辑支撑的重写已经成了当务之急。在我进行后续业务的开发前，我将对系统进行重构，主要是使之符合BPMN2.0（业务流程建模符号）规范所包含的设计思想，并引入目前流行的工作流框架Activiti进行流程控制，来为后续开发打下坚实的基础。

5.1.2 前端组件化

组件化是Vue.js的重要特性之一。我在本系统的开发中发现了大量可被组件化的模块，如标准数据表格、公共文件管理、级联选择器等。这些组件由于业务逻辑相似且在不同的功能中差异性不大，因此很适合组件化来实现代码和样式的高度复用，这也是开发人员常说的轮子。

处于时间限制，我是在学习vue的过程中开发系统的，为了降低难度暂时放弃了组件化，但为了支撑接下来更复杂的业务逻辑，组件化成了必由之路。

5.1.3 建立日志

对于一个健壮的、商业化运行的系统中，日志是最重要的组成成分，日志为业务逻辑、数据恢复、权限验证提供了重要依据，本系统当前并没有用以的日志系统来处理系统运行中产生的数据，这也是本系统当前尚未实现完全商业化原因。

当前我并没有能很好处理这个问题的能力，或许工作中会用到的甲骨文公司开发的电子商务套件（Oracle EBS）会为我提供解决方案

。

5.1.4 数据高速缓存

在本系统中存在需要频繁读写的数据，如挖机状态表，表中存储着挖掘机的坐标信息，这些坐标信息都是通过定位接口定时更新的，存储在普通的表中很显然并不合适，将会对数据库带来很大的负担。

目前主要有两种解决方案，其一是在挖机状态表使用Mysql提供memory而非普通的InnoDB数据库引擎，这种基于内存的存储的数据库引擎拥有强悍的读写能力，其二是使用当下流行的基于内存的Redis键值对数据库来缓存挖机的坐标信息。有关这方面的开发进程将在与客户沟通后的得到定位服务接口后进行。

5.2 总结

本系统的开发自2018年9月起，经历了需求对接，需求分析，原型设计，数据建模等几个阶段，到目前已是测试版本开发的末期，在独立实施整个项目的过程中，我自己也随着项目的成长而成长，在此期间收获颇多。

首先要感谢的是学校计算机系海颐特训班的老师和同学，以及广州海颐软件有限公司为我提供的优良的初期成长环境，在2018年7月至8月的暑期实习中我受益匪浅，学习到了大量数据建模，前后端通信模式的基本知识，为我的后期发展打下了坚实的基础。这也成为了我学习的动力之一。

在2018年9月份，我在母亲的帮助下与她所工作的公司进行了最初的需求对接，从而启动自己的开发计划。同时，我也进入到广州奔步电脑开始为期三个月的第二次实习，在这家公司中我担任运维开发助理的职位，接触到了项目的完整生命周期。本项目的开发工作也在休息时间内稳步进行，在这里感谢我的父母对我的支持以及公司对我的培养。

在整个项目的过程中，我的指导老师李检辉也给我提供了很多的帮助，从论文的选题，第一次项目演示，到论文审查，老师给我的帮助都是非常大的，也给我提供了很多宝贵的建议，我必须为每次毕业设计的审查做好充足的准备。师傅领进门，修行在个人，我很感谢我的老师不光领我入了门，还在成长期间提供了很多帮助和建议。

对于我自己来说，项目成长的过程也是我自己成长的过程，项目已经经历了四次重构，每一次都是脱胎换骨的更新，以此来弥补从前的先天不足，我一直在将自己工作中学到的知识应用到项目中，以此来修正自己对项目的理解，正如我在后续升级计划中提到的，项目即将迎来第五次重构，或许毕业设计会告一段落，但这个项目不会停止，因为它是根据工作所学实践的结晶，随着我的成长而成长。

参考文献

- [1] 梁灏. Vue.js 实战 [M], 北京: 清华大学出版社, 2017.
- [2] 唐汉明/翟振兴/关宝军/王洪权. 深入浅出MySQL [M], 北京: 人民邮电出版社, 2014.
- [3] 李刚. 轻量级Java EE企业应用实战(第4版) [M], 北京: 电子工业出版社, 2014.
- [4] 杨开振. Java EE互联网轻量级框架整合开发 [M], 北京: 电子工业出版社, 2017.
- [5] 陈晓孟. Mybatis技术内幕 [M], 北京: 电子工业出版社, 2017.
- [6] How2j.cn, <http://how2j.cn/> (2019/3/27)
- [7] Vue.js, <https://cn.vuejs.org/v2/api/> (2019/3/27)
- [8] MDUI, <https://www.mdui.org/> (2019/3/27)

• 说明:

相似片段中“综合”包括:

《中文主要报纸全文数据库》 《中国专利特色数据库》 《中国主要会议论文特色数据库》 《港澳台文献资源》
《图书资源》 《维普优先出版论文全文数据库》 《年鉴资源》 《古籍文献资源》 《IPUB原创作品》

- 声明:

报告编号系送检论文检测报告在本系统中的唯一编号。

本报告为维普论文检测系统算法自动生成, 仅对您所选择比对资源范围内检验结果负责, 仅供参考。

客服热线: 400-607-5550 | 客服QQ: 4006075550 | 客服邮箱: vpcs@cqvip.com

唯一官方网站: <http://vpcs.cqvip.com>



关注微信公众号