学校代码：10184

学　　号：2194242478

延 边 大 学

本科毕业设计说明书

题　　目：小型汽车租赁系统的设计与实现

学生姓名：朱赫

学　　院：工学院

专　　业：计算机科学与技术（辅修）

指导教师：王金祥　讲师

班　　级：计算机辅修班

二〇二三年三月

摘　要

我国目前是汽车大国，国人对于汽车的需求日益增加，有的人为了能开不同的车型和体验不同车量的驾驶感觉会选择租车，在汽车租赁方面各大企业更是层出不穷，例如一嗨租车、联动云租车等等，一个好的小型的汽车租赁系统对于企业来说是必不可少的，因为可以保障企业车辆的安全以及用户的信息安全性，记录租赁车型号以及用户电话等等信息，还可以最大程度上保证用户的隐私安全、提高工作效率，一个汽车租赁系统的设计与实现，对一个小型汽车租赁企业来说是有一定帮助的。本系统是有关汽车租赁的一个小型系统，可以供小型企业用于汽车租赁，经过测试，该系统可以达到对单条数据的增删查改，对多条数据的删除，对用户姓名的模糊查询等等，在实际应用过程中，可存放约 1000 条数据。

关键词：汽车租赁、java、servlet、javaweb

Abstract

At present, China is a big country of automobiles, and the demand for automobiles is increasing. Some people choose to rent cars in order to drive different models and experience the driving feeling of different vehicles. In the aspect of car rental, various large enterprises emerge in endlessly, such as Yihi car rental, linkage cloud car rental, etc. A good small car rental system is essential for enterprises, Because it can guarantee the safety of enterprise vehicles and the information security of users, record the rental model number, user phone number and other information, and can also maximize the privacy security of users and improve work efficiency, the design and implementation of a car rental system is helpful for a small car rental enterprise.

Keywords: car rental, java, servlet, javaweb

目　录

引　言

我国是一个汽车使用很多的国家，汽车一直都是新中国成立以来人们越来越刚需的工具，由于大家生活都变得富裕了，有的人就会有多辆汽车闲置，在这种情况下，汽车租赁就应运而生了，由于汽车是一个价值不低的工具，所以在对数据的储存以及信息管理要多加谨慎，所以一个安全一点的汽车租赁系统尤为重要，本文是一个小型汽车租赁系统的设计与实现的说明书，意在让使用者能够看懂本系统的构造以及了解本系统的开发过程。本系统是基于 java 语言进行开发的，数据库使用的是 mysql，连接数据库的技术为当下比较广泛使用的 mybits 技术，整体项目基于 maven 管理，让程序可移植性更高一点，在登录页面使用的是 jsp 技术，页面效果基于 css 实现的、而在用户信息录入查询页面是用的 html 和 vue。在以上情况下设计以及实现了此汽车租赁系统，以供大家参考使用。

第一章 系统的分析与设计

系统的设计对于开发一个系统来说尤为重要，它是将客户的需求从具体到抽象的过程，为后续的开发以及设计改动指明具体的方向，本车辆管理系统大体分为用户登录界面，用户注册界面，以及租赁管理模块，在租赁管理模块内容包括记录用户信息、车辆品牌以及展示车辆是否可用、根据用户姓名以及车辆信息模糊查询信息，还包括批量删除功能以及单条数据删除修改的功能。租车是用户发起的，首先用户到达门店提供租车需求，管理员输入账户密码登录到界面，记录用户输入的信息，在备注里面记录电话以及起始租借日期等等，记录的信息会被记录到数据库中，整体项目是部署在 tomcat 服务器上进行开发的，每次进入网址要占用 8080 端口号，其中整体程序包层次为有：

1. mapper 层：里面有两个接口，分别为 BrandMapper 和 UserMapper，分别存放各自连接数据库方法的接口，sql 语句是基于注解写入的，占位符通过 json 配入，稍微复发一点的 sql 是配置在 resource 下的同名目录下的 Mapper.xml。
2. pojo 层：里面放的是 Brand 以及 User 的实体类以及 PageBean，实体类主要的作用就是充当临时存储数据的类，而 PageBean 的主要作用是用于分页查询的,由于为了方便日后程序的扩展，所以 PageBean 的类型为泛型的，我们本次只有一个实体类使用到了它（Brand），设计它的逻辑是：在调用后端数据库的时候返回这样一个类型的值，它里面有两个常量，分别记录页面的总记录数和页面数据。

```
 */
public class PageBean<T> {

    //总记录数
    private int totalCount;
    //页面数据
    private List<T> rowsBrands;


    public int getTotalCount() {
        return totalCount;
    }
    public void setTotalCount(int totalCount) {
        this.totalCount = totalCount;
    }
    public List<T> getRowsBrands() {
        return rowsBrands;
    }
    public void setRowsBrands(List<T> rowsBrands) {
```

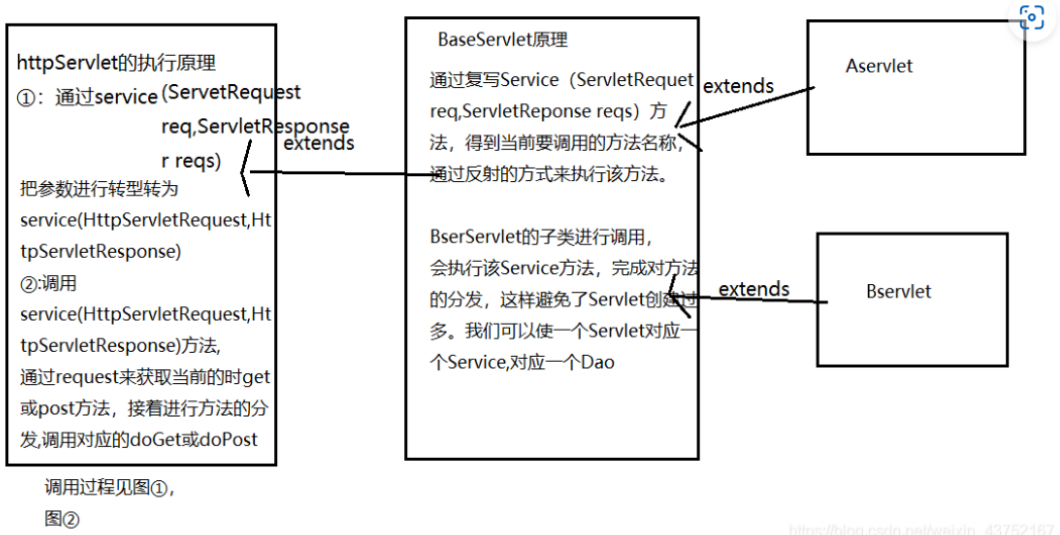
图表 1-1 代码演示

3. service 层：存放 service 接口的层级，其中有两个类，分别为 BrandService 和 UserService，各自存放实现方法的接口
4. impl：在 service 层下面，存放这 service 实现类的包，命名为 impl，其中存放两个类，BrandServiceImpl 和 UserServiceImpl。里面主要定义了几个方法，包括增删改查等功能的方法
5. util 层：这一层为工具类，目前程序只有一个 SqlSessionFactoryUtils 的类，建造这个类的目的是获取 sqlsession，类中存在静态代码块，随着程序的加载，执行数据库连接操作，后续操作可以根据类来实现 sqlsession 的获取。其中静态代码块的代码是这样的：

```
static {
        //静态代码块会随着类的加载而自动执行，且只执行一次
        try {
            String resource = "mybatis-config.xml";
            InputStream inputStream =
Resources.getResourceAsStream(resource);
            sqlSessionFactory = new
SqlSessionFactoryBuilder().build(inputStream);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

这样写的目的是，方便日后程序的维护，当连接有变更，只需要找到当前类来修改就可以，如果不这么写，每次连接都要手写这几行来获取 session，不但会产生代码冗余，还会使程序的可维护性极大的降低。

6. servlet 层：这一层主要存放几个 servlet，Baservlet 主要是各个 servlet 的基类，它继承于 httpservlet，它主要的功能是获取请求路径，然后根据反射来获取相应的类并通过反射获得执行方法。在其他 servlet 中，都要继承与 BaseServlet，在子 servlet 中，调用 service 中的操纵数据库方法来实现具体的数据库操作。BaseServlet 的子类进行调用，会执行 Service 方法，完成对方法的分发，这样能够有效避免 Serlet 创建过多。其中 BaseServlet 的原理如下图：



图表 1-2 BaseServlet **原理**

BaseServlet 代码：

```
    @Override
        protected void service(HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException {
            //获取请求路径
            String uri = req.getRequestURI();
            //获取最后一段路径，方法名
            int index = uri.lastIndexOf('/');
            String methodName = uri.substring(index + 1);
            //执行方法
            //获取 BrandServlet /UserServlet 字节码对象 Class
```

```
            Class<? extends BaseServlet> cls = this.getClass();
            //获取方法 Method 对象
            try {
              Method method = cls.getMethod(methodName,
HttpServletRequest.class, HttpServletResponse.class);
                //执行方法
                method.invoke(this,req,resp);
            } catch (NoSuchMethodException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }


        }
```

1.1 登录界面

所有的操作人员一定要根据登录界面才能进入系统，在用户名或者密码输入错误的时候要相应的提示错误信息，还要考虑到用户输入空字符串以及 sql 注入问题，在后端代码要进行相应的判断，在 sql 注入方面也要尤其注意，还要为了方便清空输入错误的信息编写相应的重置信息按钮，在登录界面要尽量做到美观，表单对齐还要考虑到前端请求用 get 还是 post，因为 get 请求可能会暴露管理员密码，对于不知道用户名以及密码的管理员还要设置能跳转到注册页面的连接。

1.2 注册页面

在注册页面，要提示相应的注册成功以及失败信息，字符串为空的情况下也要作出后端的相应判断，还要检测新注册的用户名是否和数据库已经存在的用户名重复，重复的情况下要提示管理员重新设置用户名，还要阻止数据库进行插入操作，避免数据库有空值，这样会导致用户名为空也能进入系统内部。

1.3 租赁管理模块

在租赁管理模块要进行大量的数据库操作，所以对于设计者来说尤为重要，不能使程序崩掉，影响用户的操作。首先是展示页面，因为数据上百条，所以一定要进行分页，而写为了满足操作自由度，在分页的基础上，还要满足管理员能够选择每页显示条数和对相应的页面进行输入快速跳转，每条数据上要有删除以及修改按钮，点击修改数据能实现看到备注信息，因为备注信息为隐私数据，所以一定要点击更新按钮的时候才能看见，还要有汽车使用状态的滑动按钮。在每条数据前要有复选框这样能满足管理员的批量杉树数据，由于有时存在误删操作，所以在每次点击删除之前，要提示用户是否要删除数据，每次删除或更新成功后要作出相应的提示，告诉管理员操作成功，在页面的最上方要有能对数据进行查询的功能，姓名以及汽车名称可以实现模糊查询。

第二章 数据库设计

本系统使用的数据库是 MySQL，相比于市面上的其他主流数据库，MySQL 的优点很突出，首先是他的运行速度足够快，其次是对于个人来说是完全免费的，相比于其他主流数据库 MySQL 也具有复杂程度低的主要特点，MySQL 还可以利用标准 SQL 语法和支持 ODBC（开放式数据库连接）的应用程序。在数据库设计层面上，一共设置了两个表，首先是管理员用户登录表，该表主要存储管理员账号密码等信息，另一个表是存储客户信息数据的表，主要存储客户的姓名、车辆的名称、车辆的可用性、以及车辆用户信息的备注以及用户的电话号码等等信息，其中 id 是不暴露在展示界面的，id 只供后期或取 json 字符串，所以并没有设置为连续。由于本项目是基于 maven 管理开发的，所以连接后端数据库框架使用的是 mybits，框架底层执行的业务是 jdbc 技术支持的，连接数据库的字符串为：name="url"
value="jdbc:mysql:///zhulun?useSSL=false&allowPublicKeyRetrieval=true"

name="username" value="root"

name="password" value="123456"

用户要根据自己的数据库连接名以及密码来进行更改，具体的位置在 work\src\main\java\resource\mybatis-config.xml

在操作数据库增删改查操作是基于注解以及配置文件进行的，具体会在后面提到。两个表的建表语句分别为（values 为虚拟信息）：

```
create database zhulun;

use zhulun;

drop table if exists tb_brand;
```

```
create table tb_brand
(
    id          int primary key auto_increment,
    brand_name      varchar(20),
    custom_name varchar(20),
    ordered          int,
    tel    varchar(100),
    status           int
);
insert into zhulun.tb_brand (brand_name, custom_name, ordered, tel, status)
values
        ('宝马', '张三', 100, '111111111', 1),
        ('特斯拉', '李四', 50, '555555555', 1),
        ('红旗', '王五', 30, '6666666666', 1),
        ('一汽大众', '李培林', 10, '333333333', 1),
        ('玛莎拉蒂', '王立业', 50, '555555555', 0),
        ('蔚来', '郭林权', 5, '7777777777', 0),
        ('桑塔纳', '赵哈', 40, '55656464', 1),
        ('特斯拉', '孙飒飒', 50, '884946465', 1),
```

```
('奥迪', '李成功', 5, '4546465', 0);


use zhulun;
CREATE TABLE tb_user(
    id int primary key auto_increment,
    username varchar(20) unique,
    password varchar(32)
);
INSERT INTO tb_user(username,password) values('延边','123'),('大学','234');
SELECT * FROM tb_user;
```

在执行两个建表语句之后，数据库的搭建就完成了。

第三章 前后端交互设计与实现

3.1 登录页面的前后端交互

图表 3-1 登录界面

在前端页面中，登录页面是基于jsp完成的，jsp技术可以支持多种网页格式，JSP技术既可以支持HTML的网页技术格式,还可以支持应用于WML文件格式,还可以支持其他一些XML的文件格式。Jsp使用的脚本语言是java语言，它具有java的大多数好处，它还具有强大的扩充能力，大大降低了开发者的落实难度。在页面设计中，username以及password是根据用户在输入文本框中输入的数据进行获取的，当用户电锯登入按钮时，<form action="/work/user/login" id="form">，框架会去后端UserServlet里面找login方法然后执行，其中login方法是这样的：传入两个参数：HttpServletRequest和HttpServletResponse对象。HttpServletRequest对象表示接收到的HTTP请求，它包含了客户端通过GET或POST请求传递过来的参数以及其他的HTTP头部信息。HttpServletResponse表示要发送给客户端的HTTP响应，它可以设置HTTP响应头、重定向、设置Cookie

等。

在方法中，首先使用request.getParameter()方法获取客户端传递过来的用户名和密码参数，并将它们存储到相应的字符串变量中。然后调用UserService类的login方法来验证用户的身份。如果验证通过，就使用response.sendRedirect()方法进行重定向，跳转到brand.html页面，显示用户已经登录成功。

如果验证不通过，那么该方法会将错误信息存储到request对象中的login_msg属性中。之后，使用request.getRequestDispatcher()方法将请求转发到login.jsp页面，以便用户重新输入用户名和密码。这里使用请求转发而不是重定向的原因是，需要将错误信息传递给login.jsp页面，而请求转发可以将这些信息直接添加到request对象中，在login.jsp页面中可以轻松地获取到并显示给用户。

首先，方法会获取用户输入的 username 和 password，然后调用 service 的 login 方法，service 层执行 mapper 层的数据库查询语句

```
@Override
public User login(String username, String password) {
    // 获取SqlSession
    SqlSession sqlSession = factory.openSession();
    //获取UserMapper
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);
    //调用方法
    User user = mapper.select(username, password);

    //释放资源
    sqlSession.close();

    return user;
}
```

图表 3-2 代码演示

在mapper层的代码是这样的：

@Select("select * from tb_user where username = #{username} and password = #{password}")

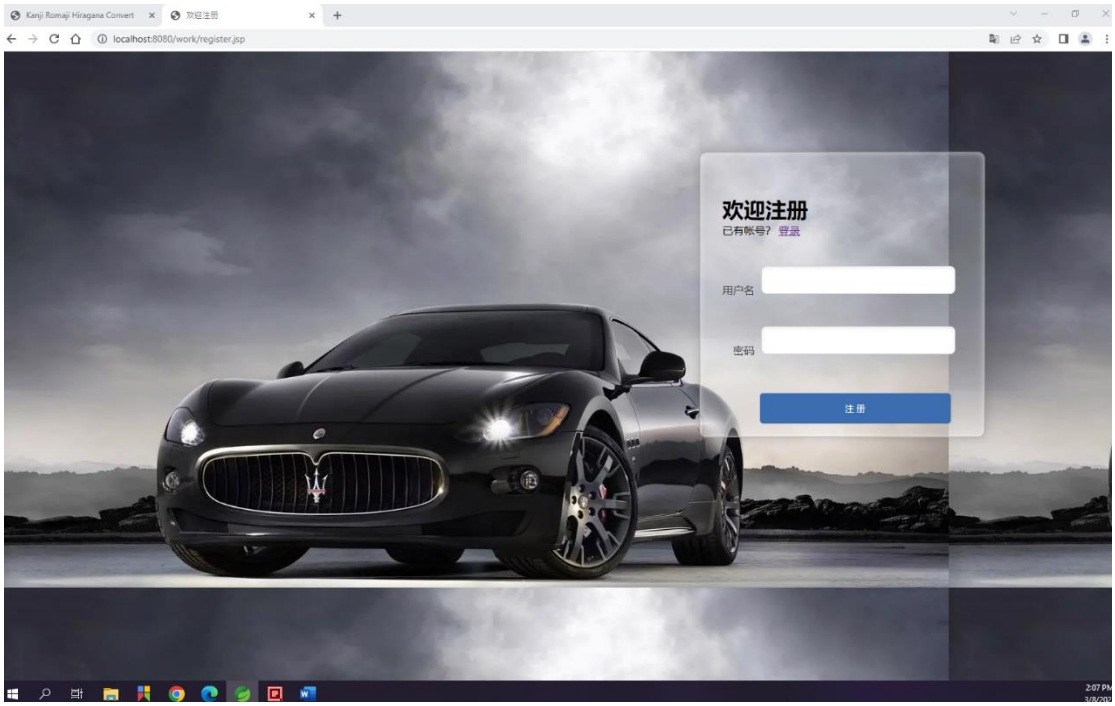
User select(@Param("username") String username,@Param("password") String password);

使用注解执行 sql 语句，然后返回的类型为 pojo 中的 User 实体类，login 向下执行时，如果得到的 User 为空，那么就证明没有在数据库找到相应的数据，那么就是用户名或者密码输入错误，后端会向 request 域当中存储错误信息字符串，并携带参数重新跳转到 login.jsp。 ${login_msg} ${register_msg} 。这一步重新跳转的目的是能让前端获取错误信息并展示。如果得到的 User 存在结果，那么证明数据库存在信息，那么后端代码会重定向到 brand.html，也就是用户信息管理页面，证明着登录成功。这里要提一下转发和重定向的具体区别：转发相当于服务器跳转，相当于方法调用，在执行当前文件的过程中转向执行目标文件，两个文件(当前文件和目标文件)属于同一次请求，前后页 共用一个 request，可以通过此来传递一些数据或者 session 信息，request.setAttribute()和 request.getAttribute()。而重定向会产生一个新的 request，不能共享 request 域信息与请求参数。

然后 login 页面还有一个没有账号功能：没有账号？

这个功能可以跳转到注册账号页面，也就是/register.jsp。

3.2 注册页面的前后端交互



图表 3-3 注册界面

这个页面的前端构造与登录页面大同小异，首先是想反回登录页面，用户可以选择点击登录连接，就会跳转到上一个登陆页面，已有帐号？ 登录。这个点击事件的调用的是servlet中的register方法<form id="reg-form" action="/work/user/register" method="get">，其中register方法是这样的：

接收客户端通过HTTP请求传递过来的用户名和密码参数，首先对参数进行非空校验，如果存在空值或空字符串，则将错误信息存储到request对象中，并将请求转发到register.jsp页面，以便用户重新填写注册信息。如果参数校验通过，则创建一个User对象并通过UserService类的register方法进行注册，如果注册成功，则将成功信息存储到request对象中并跳转到login.jsp页面，提示用户注册成功，请登录。如果注册失败，将错误信息存储到request对象中并将请求转发到register.jsp页面，以便用户重新填写注册信息。

该方法的具体实现如下：

首先，使用request.getParameter()方法获取客户端传递过来的用户名和密码参数，并将它们存储到相应的字符串变量中。然后，判断这些参数是否为空或

空字符串，如果有任何一个为空或空字符串，则将错误信息存储到request对象中，并将请求转发到register.jsp页面，提醒用户填写必要的注册信息。如果用户名和密码参数都不为空，则创建一个User对象，并将用户名和密码设置为用户输入的值。接着，调用UserService类的register方法来检查是否可以注册。如果可以注册，则将提示信息存储到request对象中，并将请求转发到login.jsp页面，提示用户注册成功，请登录。如果无法注册，则将错误信息存储到request对象中，并将请求转发到register.jsp页面，提示用户该用户名已经被占用，请重新填写用户名。

```
@Override
public boolean register(User user) {
    // 获取SqlSession
    SqlSession sqlSession = factory.openSession();
    //获取UserMapper
    UserMapper mapper = sqlSession.getMapper(UserMapper.class);

    //判断用户是否存在
    User userTemp=mapper.selectByUsername(user.getUsername());

    if(userTemp==null) {
        mapper.add(user);
        sqlSession.commit();

    }
    return userTemp==null;

}
```

图表 3-4 判断代码

首先会判断username和password是否为空，如果为空的话，写入错误信息并转发到register.jsp。如果不为空，那么就对User实体类进行赋值，传入service层的register方法，这个方法主要的功能是判断用户是否存在，如果存在返回false否则返回true，判断使得否存在会调用mapper层的selectByUserName方法，返回的为User，如果User有数据，那么证明用户输入的username与数据库中的重复返回的数据为false。

当servlet层结果接收到true数据时，证明数据库数据没有重复，那么在request域中存储提示成功的信息并跳转到login.jsp中，并显示注册成功的信息给用户，反之，证明数据库中存在用户名，将错误信息提示为用户名已经存在并添加到

request域中转发到register中并提示用户名已存在，servlet整体的代码逻辑是这样的：

```
if(b) {
            request.setAttribute("register_msg", "注册成功，请登录");
            request.getRequestDispatcher("/login.jsp").forward(request, response);

        }else {
            request.setAttribute("register_msg", "用户名已存在");
            request.getRequestDispatcher("/register.jsp").forward(request, response);
        }
```

3.3 数据主页面前后端交互

3.3.1 页面展示的设计与实现

在用户跳转到程序主页面时，首先要进行的就是页面展示，在前端，先定义了以及虚假的展示页面，也就是显示程序正在加载中的页面，由于加载后端数据库需要几毫秒，在数据过多的情况下需要甚至几秒，所以刚开始不能展现空白页面，所以要展现一个加载中的页面以供展示，其中前端代码是这样实现的：

```
tableData: [{
        brandName: '加载中',
        customName: '加载中',
        ordered: '100',
        status: "1"
    }, {
        brandName: '加载中',
        customName: '加载中',
        ordered: '100',
        status: "1"
    }, {
```

```
                brandName: '加载中',
                customName: '加载中',
                ordered: '100',
                status: "1"
            }, {
                brandName: '加载中',
                customName: '加载中',
                ordered: '100',
                status: "1"
            }]
```

当页面加载完成后，就要调用后端代码了，前端调用是基于Vue实现的，在前端Vue中定义一个钩子，里面写查找全部的方法：

```
new Vue({
    el: "#app",
    mounted(){
        //当页面加载完成后，发送异步请求，获取分页数据
        this.selectAll();
    },
```

因为页面刚开始就是分页的所以在定义后端方法时要根据分页查询，前端获取用户页面设置的现在页数以及每页的条数，然后通过post请求和在地址上传入现在页数和每页条数的参数传给后端：

```
axios({
                method:"post",

url:"http://localhost:8080/work/brand/selectByPagesAndCondition?curPage="+this.curPage+"&pageSize="+this.pageSize,
                data:this.brand
            })
```

后端获取现在页数以及每页条数（注意一下的代码及操作均是基于Brand、BrandService、BrandServlet进行操作，与User无任何关系，因为在登录页面后就不用User了！），在Servlet里面寻找并调用selectByPagesAndCondition方法，执行

相关操作，其中后端代码是这样的：

```
public void selectByPagesAndCondition(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
                request.setCharacterEncoding("UTF-8");
                String _curPage= request.getParameter("curPage");
                 String _pageSize= request.getParameter("pageSize");
                 int curPage=Integer.parseInt(_curPage);
                 int pageSize=Integer.parseInt(_pageSize);

                 //获取查询条件的对象
                 BufferedReader br = request.getReader();
                 String params = br.readLine();//json字符串
                 //转为Brand
                 Brand brand = JSON.parseObject(params,Brand.class);



                 PageBean<Brand> pageBean
=brandService.selectByPagesAndCondition(curPage, pageSize,brand);



                 //转为JSON
                 String jsonString = JSON.toJSONString(pageBean);
                 //写数据
                 response.setContentType("text/json;charset=utf-8");
                 response.getWriter().write(jsonString);
            }
```

首先这个方法的定义是基于页面上的两个功能的，第一是页面的初始化，第二是单击查询按钮，首先获取前传过来的两个参数（现有页数以及页面显示条数），并将其转换为integet类型，然后获取request域中的json字符串，这里大家会有疑问，那就是，万一是页面加载的时候怎么办，因为页面加载的时候是没

有json字符串的，其实在BrandMapper端，已经将后台查询语句定义好：

```
<select id="selectByPagesAndCondition" resultMap="brandResultMap">
        select *
         from tb_brand
        <where>
            <if test="brand.brandName != null and brand.brandName != ''">
                and brand_name like #{brand.brandName}
            </if>
            <if test="brand.customName != null and brand.customName != ''">
                and custom_name like #{brand.customName}
            </if>
            <if test="brand.status != null">
                and status = #{brand.status}
            </if>
        </where>
        limit #{begin}, #{size}
    </select>
```

还有一个查询符合条件的数量的语句，这个方法的作用就是查询符合条件的条数为多少：

```
<select id="selectTotalCountByCondition" resultType="java.lang.Integer">
        select COUNT(*)
         from tb_brand
        <where>
            <if test="brandName != null and    brandName != ''">
                and brand_name like #{brandName}
            </if>
            <if test="customName != null and    customName != ''">
                and custom_name like #{customName}
            </if>
            <if test="status != null">
```

```
            and status = #{status}
        </if>
    </where>
</select>
```

这里面是基于一个if条件进行的，如果json字符串没有获取到，那么自然brandName和customName自然是获取不到的，那么就不会进入两个if条件，那么sql语句就是select * from tb_brand和select count(*) from tb_brand，后者返回一个int类型的数据，代表符合条件的条数，当然如果没有获取到json字符串就会全部符合条件，回到selectByPagesAndCondition方法，下一步是调用service的selectByPagesAndCondition：

```
@Override
public PageBean<Brand> selectByPagesAndCondition(int curPage, int pageSize, Brand brand) {
    // 获取对象
    SqlSession sqlsession =factory.openSession();
    //获取BrandMapper对象
    BrandMapper brandMapper=sqlsession.getMapper(BrandMapper.class);

    //计算开始索引
    int begin =(curPage-1)*pageSize;
    //计算查询条数
    int size =pageSize;

    String brandNameString = brand.getBrandName();
    if(brandNameString!=null && brandNameString.length()>0) {
        brand.setBrandName("%"+brandNameString+"%");
    }
    String customNameString = brand.getCustomName();
    if(customNameString!=null && customNameString.length()>0) {
        brand.setCustomName("%"+customNameString+"%");
```

```
        }

        List<Brand>
rows=brandMapper.selectByPagesAndCondition(begin,size,brand);
        //获取总符合条件数量
        int count =brandMapper.selectTotalCountByCondition(brand);

        PageBean<Brand> pageBean=new PageBean<>();
        pageBean.setRowsBrands(rows);
        pageBean.setTotalCount(count);
        sqlsession.close();
        return pageBean;
    }
```

其中运用了一个小算法，就是计算开始的索引“int begin =(curPage-1)*pageSize;”，就是开始的索引等于当前的页数减一后乘每页显示的条数，那么假如当前是在第2页并且用户选择每页显示5条的话，那么开始的索引就是5，然后对传进来的Brand的Name进行判断，如果不为空且长度大于0的时候，那么证明用户执行的是查询按钮，那么就要进行相应的模糊查询，在进行模糊查询时，BrandMapper是没有百分号的，因为sql模糊查询需要like加上%，所以我们在传变量字符串的时候要传入两个%，然后执行BrandMapper定义好的两个方法，一个是查询符合条件的Brand和查询符合条件的总数量，然后返回符合条件的Brand和数量，封装进PangBean类里面，然后在Servlet端把数据写入request域当中，在前端得到响应之后，把数据放进表单里面，前端的总体代码是这样的：

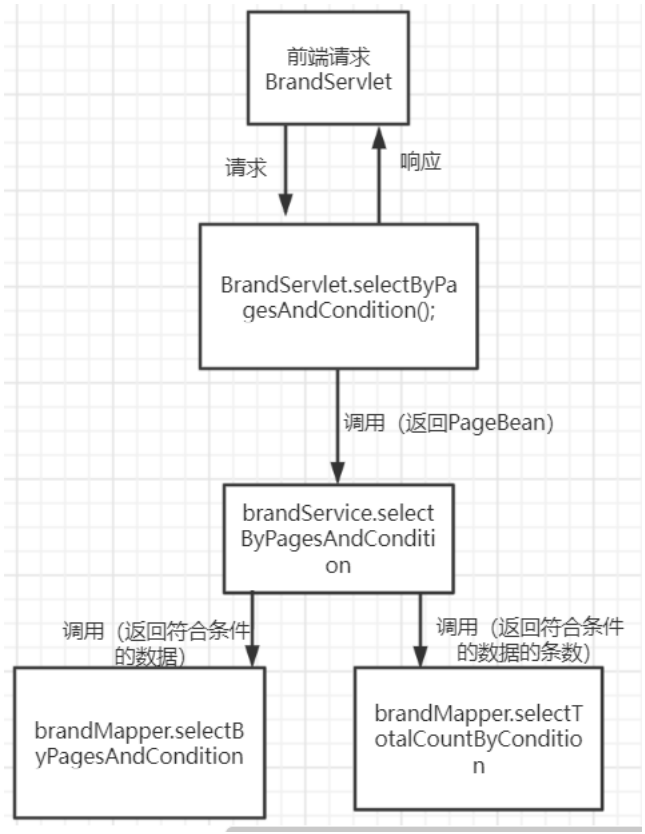
```
axios({
                        method:"post",

url:"http://localhost:8080/work/brand/selectByPagesAndCondition?curPage="+this.curPage+"&pageSize="+this.pageSize,
                        data:this.brand
```

```
        }).then(resp => {
            this.tableData = resp.data.rowsBrands;
            //设置总记录数
            this.totalCount = resp.data.totalCount;


        })

    },
```

还有一个使用前端SelectAll的点，就是当用户进入主页后，点击下方跳转页面时，这个跳转是基于前端SelectAll实现的，只需要将主方法这样定义就可以：

```
//分页
    handleSizeChange(val) {
       this.pageSize=val;
        this.selectAll();

    },
    handleCurrentChange(val) {
        this.curPage=val;
        this.selectAll();
    },
```

在这里关于后端我画了一个流程图，以供参考使用：



图表 3-5 流程图

3.3.2 新增按钮跳转页面的设计与实现

当用户点击新增按钮时，页面会跳出新增选项的表单，可以新增用户的所有信息，但是ID是不可以用户新增的，因为ID只是对应数据库修改查询数据用的，不用体现在前端展示页面，以下是新增的表单的前端代码：

```
<!--添加数据对话框表单-->
    <el-dialog
            title="新增品牌"
            :visible.sync="dialogVisible"
            width="30%"
            >
```

```
<el-form ref="form" :model="brand" label-width="80px">
    <el-form-item label="汽车品牌">
        <el-input v-model="brand.brandName"></el-input>
    </el-form-item>

    <el-form-item label="用户姓名">
        <el-input v-model="brand.customName"></el-input>
    </el-form-item>

    <el-form-item label="排序">
        <el-input v-model="brand.ordered"></el-input>
    </el-form-item>

    <el-form-item label="备注">
        <el-input type="textarea" v-model="brand.tel"></el-input>
    </el-form-item>

    <el-form-item label="状态">
        <el-switch v-model="brand.status"
                   active-value="1"
                   inactive-value="0"
        ></el-switch>
    </el-form-item>


    <el-form-item>
        <el-button type="primary" @click="addBrand">提交</el-
button>
        <el-button @click="dialogVisible = false">取消</el-button>
    </el-form-item>
```

```
        </el-form>

    </el-dialog>
```

当用户输入汽车品牌、用户姓名、排序、备注、状态信息，然后点击提交按钮，然后会调用前端得addBrand方法，前端的addbrand方法是这样编写的：

```
        // 添加数据
            addBrand(){
                //console.log(this.brand);
                var _this = this;

                // 发送ajax请求，添加数据
                axios({
                    method:"post",
                    url:"http://localhost:8080/work/brand/add",
                    data:_this.brand
                }).then(function (resp) {
                    if(resp.data == "success"){
                        //添加成功

                        //关闭窗口
                        _this.dialogVisible = false;

                        // 重新查询数据
                        _this.selectAll();
                        // 弹出消息提示
                        _this.$message({
                            message: '添加成功!!',
                            type: 'success'
                        });
```

```
            }
        })

    },
```

根据代码可以看出，首先会发送ajax请求，然后执行post请求，此时页面会跳转到add方法，在找到servlet端的add方法之后，进入add方法，因为与之前的代码相似，所以这里就不作过多展示，首先获取前端穿穿过来的json字符串，然后将字符串转换为Brand实体类，然后调用service端的add方法，service端获取session后会调用mapper的add方法：

```
//添加数据
    @Insert("insert into tb_brand
values(null,#{brandName},#{customName},#{ordered},#{tel},#{status})")
    void add(Brand brand);
```

获取的相应字符串会替代sql中的占位符，然后执行sql语句，例如用户这样输入：

图表 3-6 新增品牌

那么当用户提交之后，前端就会传过来这样的json字符串：

{status: "1", brandName: "保时捷鲨鱼版", customName: "磊", id: "", ordered: "200", tel: "这个车子很酷炫，是鲨鱼版的哦，可以游泳"}

brandName: "保时捷鲨鱼版"

customName: "磊"

id: ""

ordered: "200"

status: "1"

tel: "这个车子很酷炫，是鲨鱼版的哦，可以游泳"

然后执行mapper定义好的sql：

insert into tb_brand values(null, 保时捷鲨鱼版 , 磊,200, 这个车子很酷炫，是鲨鱼版的哦，可以游泳,1)")

然后插入进数据库，在执行完插入数据之后，后端给出的相应为“success”，当前端获取到相应的相应之后，会先重新查询数据，也就是刷新页面，然后会提示用户添加数据成功，如果用户最并没有输入数据，用户点击了取消按钮，那么前端会执行语句：@click="dialogVisible = false，让当前新增数据页面消失。

3.3.3 批量删除按钮的设计与实现

当用户点击批量删除按钮时，前端会执行一下代码：

```
deleteByIds(){
                    //如果没有选择数据
                    if(this.multipleSelection.length==0){
                        this.$message({
                            type: 'info',
                                message: '您没有选择数据'
                        });
```

```
  return;
  }
// 弹出确认提示框
this.$confirm('此操作将删除该数据, 是否继续?', '提示', {
    confirmButtonText: '确定',
    cancelButtonText: '取消',
    type: 'warning'
}).then(() => {
    //用户点击确认按
    //创建 id 数组 [1,2,3], 从 this.multipleSelection 获取
    for (let i = 0; i < this.multipleSelection.length; i++) {
        let selectionElement = this.multipleSelection[i];
        this.selectedIds[i] = selectionElement.id;
    }
    //发送 AJAX 请求
    var _this = this;
    // 发送 ajax 请求
    axios({
        method:"post",
```

url:"http://localhost:8080/work/brand/deleteByIds",

data:_this.selectedIds

首先会判断复选框是否选中数据，如果什么都没有选，那么系统会提示，您没有选中数据，然后return掉，如果用户选中了复选框，那么页面会提示用户，此操作将删除数据，师傅选择要继续，因为只有这样才能确保用不会误删数据造成严重的后果，当用户选择了确定删除按钮时，前端首先会获取用户选择的数据的id，并把id数据放入提前定义好的变量里，然后会在servlet里面找到deleteByIds()方法，然后在servlet层会调用service层的方法，然后在调用mapper层的方法：

```
//批量删除
void deleteByIds(@Param("ids") int[] ids);
```

在后在brandMapper.xml层调用相应的sql：

```
<delete id="deleteByIds">
        delete from tb_brand where id in
        <foreach collection="ids" item="id" separator="," open="(" close=")">
            #{id}
        </foreach>
    </delete>
```

然后执行完成之后，servlet会相应“success”成功的标识，当前端接受到标识之后提示删除成功！

3.3.4 更新按钮的设计与实现

更新和删除按钮前端是这样编写的：

```
<template>
        <el-table
                :data="tableData"
                style="width: 100%"
```

```
            :row-class-name="tableRowClassName"
            @selection-change="handleSelectionChange"
    >
        <el-table-column
                type="selection"
                width="55">
        </el-table-column>
        <el-table-column
                type="index"
                width="50">
        </el-table-column>

        <el-table-column
                prop="brandName"
                label="汽车品牌"
                align="center"
        >
        </el-table-column>
        <el-table-column
                prop="customName"
                label="用户姓名"
                align="center"
        >
        </el-table-column>
        <el-table-column
                prop="ordered"
                align="center"
                label="排序">
        </el-table-column>
        <el-table-column
```

```
                    prop="status"
                    align="center"
                    label="当前状态">
                </el-table-column>

                <el-table-column
                    align="center"
                    label="操作">

                    <template slot-scope="scope">
                        <el-row>
                            <el-button type="primary"
@click="startEdit(scope.row)">更新</el-button>
                            <el-button type="danger"
@click="deleteById(scope.row)">删除</el-button>
                        </el-row>
                    </template>

                </el-table-column>

            </el-table>
        </template>
```

当点击更新按钮时，前端页面会获取当前数据的一行，然后将当前行的json字符串数据转化并赋值给定义好的变量里面，然后将数据填入表单：

```
        startEdit(row) {
            // 获取改行已经有的数据，以供填入修改框
            this.brandEdition = JSON.parse(JSON.stringify(row));
            // 弹出修改框
            this.editDialogVisible = true;
        },
```

然后页面出现编辑菜单，由于编辑菜单和新增菜单的前端代码大体相同，这里就不做过多展示，当用户修改完数据后，点击提交按钮时会执行以下方法：

```
onfirmEdit() {
    // var _this = this
    //axios transit ajax request
    axios({
        method: "post",
        url: "http://localhost:8080/work/brand/update",
        contentType: "application/json;charset=utf-8",
        data: this.brandEdition
    }).then(resp =>  {
        if (resp.data === "success") {
            this.editDialogVisible = false
            this.selectAll()
            this.$message({
                message: '编辑成功！！',
                type: 'success'
            });
        }
    })
},
```

去调用servlet中的update方法，然后向下调用service中的update方法，然后调用mapper中的update方法：

```
//修改
@Update("update tb_brand set brand_name=#{brandName},custom_name=#{customName},ordered=#{ordered},tel=#{tel},status=#{status} where id=#{id}")
void update(Brand brand);
```

当修改成功后前端会提示编辑成功并重新刷新页面。

3.3.5 单个删除按钮的设计与实现

当用户选择单条数据进行删除时,首先会和用户确认是否要删除这条数据，避免误删，当用户选择确定时，再进行删除，前端代码是这样编写的：

```
deleteById(row){
                this.brand.id=row.id;
                this.$confirm('此操作将删除该数据, 是否继续?', '提示', {
                    confirmButtonText: '确定',
                    cancelButtonText: '取消',
                    type: 'warning'
                }).then(() => {
                  axios({
                      method: "post",
                          url: "http://localhost:8080/work/brand/deleteById",
                          contentType: "application/json;charset=utf-8",
                          data: this.brand

                  }).then(resp=>{
                      if (resp.data === "success") {

                              this.selectAll();
                              //pop up OK msg
                              this.$message({
                                  message: '删除成功',
                                  type: 'success'
                              });
                      }

                  })
```

```
        }).catch(() => {
            //when cancel clicked
            this.$message({
                type: 'info',
                message: 'Deletion stop'
            });
        })

    },
```

首先会将这一行的id值赋值给brand对象：

```
    // 品牌模型数据
    brand: {
        status: '',
        brandName: '',
        customName: '',
        id:"",
        ordered:"",
        tel:""
    },
```

当用户点击确定按钮之后，后端回去servlet中寻找deleteById()方法：

```
public void deleteById(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        request.setCharacterEncoding("UTF-8");
        //接收品牌数据
        BufferedReader br = request.getReader();
        String params = br.readLine();//json字符串

        //转为Brand对象
        Brand brand = JSON.parseObject(params, Brand.class);
```

```
            //调用service
            brandService.deleteById(brand);



            //响应成功的标识
            response.getWriter().write("success");
        }
```

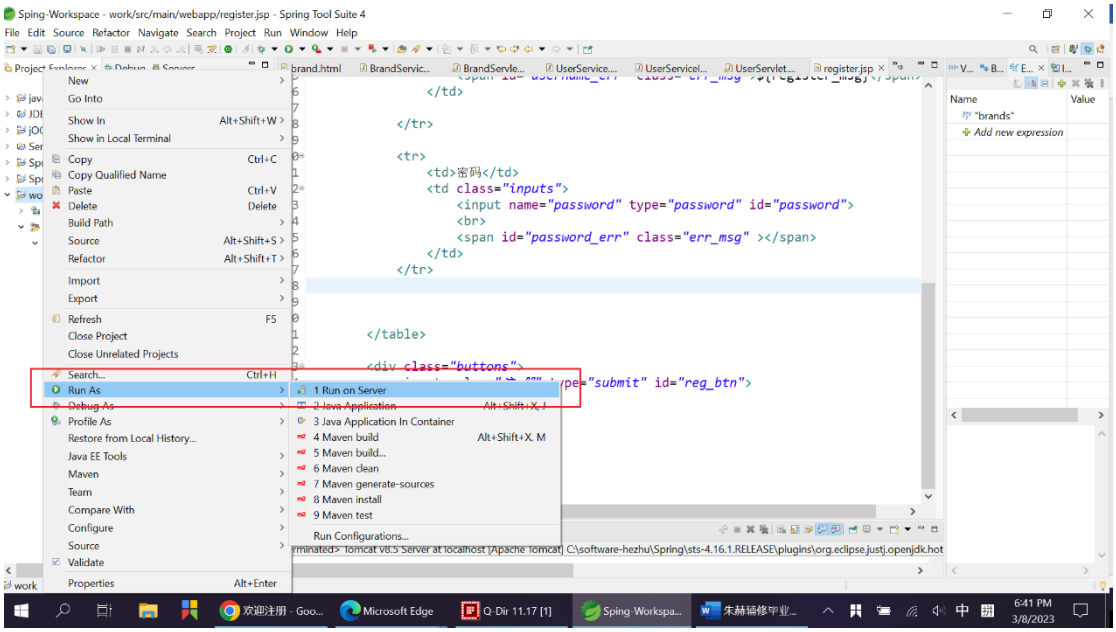
会调用service端的deleteById()，并将brand对象传入，在mapper端，执行数据库语句的操作，最终执行的语句为：

```
//根据id删除
    @Delete("delete from tb_brand where id=#{id}")
    void deleteById(Brand brand);
```

如果删除成功，前端会给出“success”的相应，当后端获取到响应之后，会提示用户删除成功，并重新执行sellectall（）方法刷新页面

第四章 页面操作说明书

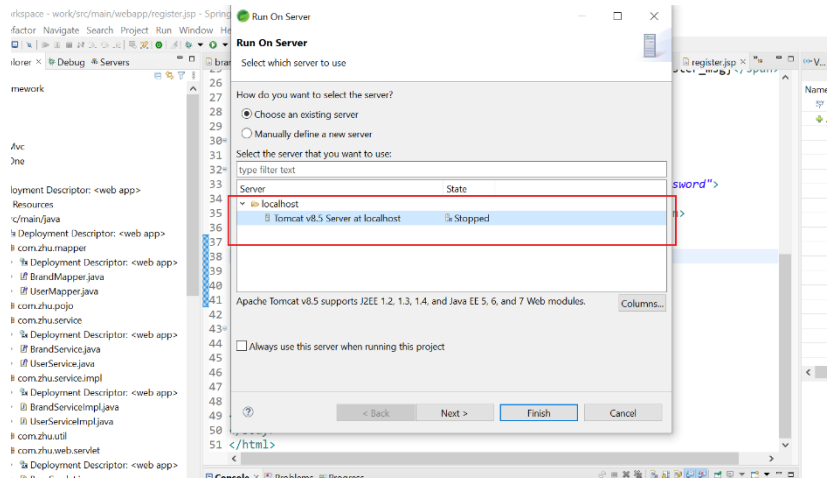
4.1 程序运行



图表 4-1 运行演示

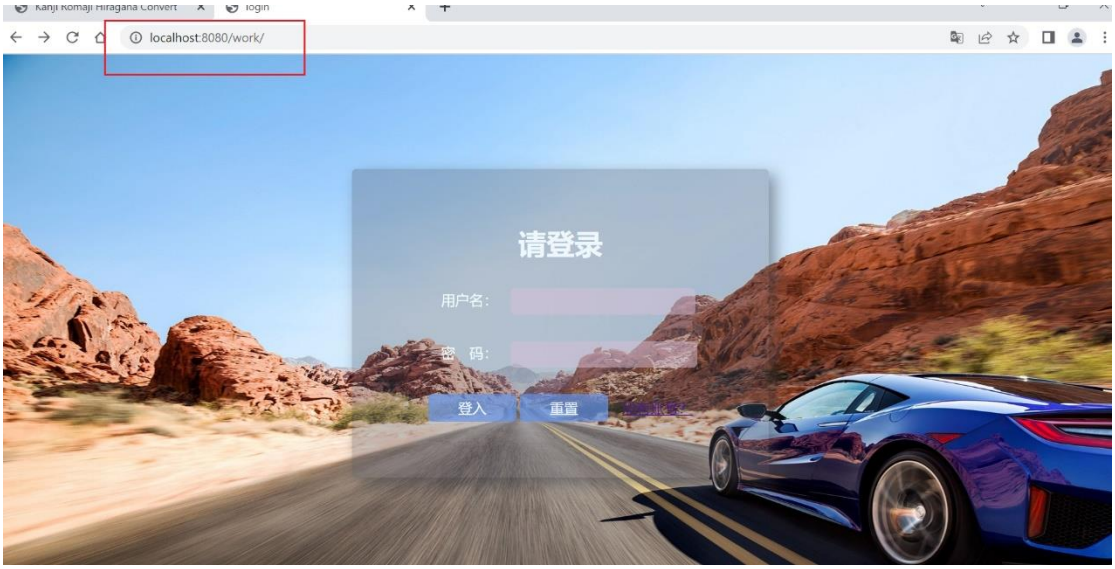
首先右键程序，run as 然后选择 run on server

然后选择在 tomcat 服务器上运行



图表 4-2 运行方法

这样我们就进入了网页程序的主入口，因为是部署在 tomcat 服务器上的，所以占用的端口号为 8080，在程序使用期间，切勿关闭服务器，否则可能会造成程序数据的丢失。

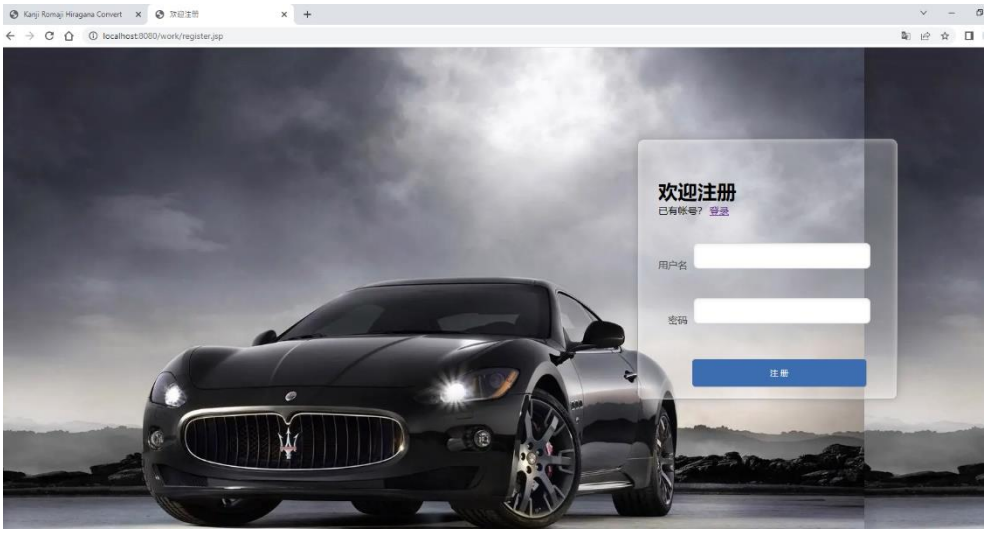


图表 4-3 端口演示

当我们输入错误的账号或者密码的时候，程序会提示错误信息“用户名或密码错误!”

图表 4-4 错误提示

当点击重置按钮会清空输入的内容，更加方便客户操作。点击没有账号按钮时会跳转到注册页面



图表 4-5 注册界面

图表 4-6 数据界面

当用户名或者和密码全部输入正确时会跳转到数据操作页面：

4.2 注册页面说明

当输入已经存在的用户名时，页面会报错让用户重新输入

图表 4-7 注册错误提示

当全部输入正确时，会跳转回登录页面，并提示注册成功信息：

图表 4-8 注册成功提示

4.3 数据操作页面的说明

在页面最下方，有跳转页面、点击页面以及选择每页条数的功能：

图表 4-9 每页条数

在页面最上方，可以根据租用状态、用户姓名、汽车品牌，检索，查看符合条件的数据，还可以根据字段中的关键字进行模糊查询，检索出符合条件的信息并展现出来：

图表 4-10 模糊查询

上图可以看出，在下方的每页条数等数据，也是随着检索状态而改变的，也具有跳转页面、点击页面以及选择每页条数的功能。在检索条件中，状态条件是下拉框来展示的，这样会防止用户输入错误信息：

整体页面支持复选框功能，用户可以一次选择多条数据，这个功能主要是配合批量删除时使用的，最上面的复选框是一个减号，主要的功能是取消全选，这样能极大程度的减少复杂性：

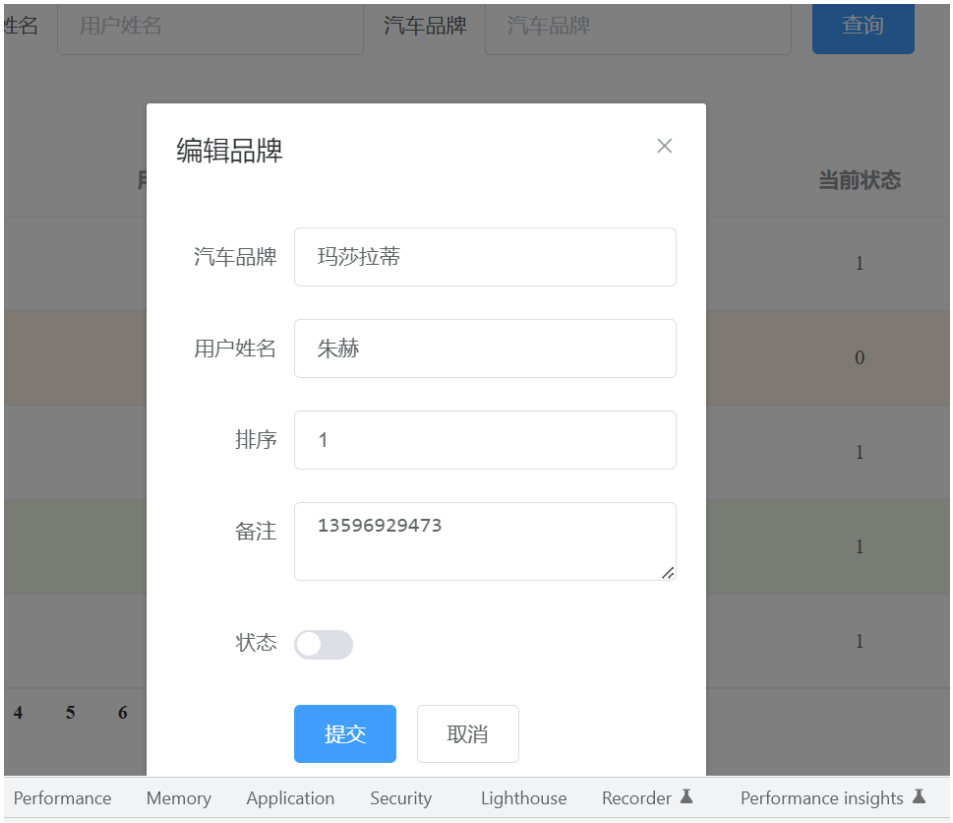
图表 4-11 复选框

图表 4-12 状态选择

点击批量删除时，程序会给与提示，方式用户误操作造成难以挽回的后果，点击确定时，数据将会被删除，点击取消，将会回到上一级：点击新增按钮时，程序会跳出新增的界面，用户可以输入，用户的 ID 是不会暴露在外面的，ID 只提供查询索引,当用户输入好信息后，点击提交按钮，数据会被提交到数据库中，当用户点击取消时，程序会退回到原有的页面并等待用户下一步操作。当用户点击提交时，系统会提示用户“添加成功!”

图表 4-13 提示

在每条数据右侧，都会有更新按钮，以便于用户更新单条数据，在点击更新时，输入框会自动获取程序所点击的信息，以便于用户的修改当用户点击提交按钮后，数据会提交到数据库并将成功信息响应给用户，当用户点击取消按钮时，页面会自动取消并返回到数据页面以提用户下一步骤的操作：



图表 4-12 **更新**

结　论

本次的汽车租赁系统的体量并不大，但涉及的细节较多，在本次设计中，整体架构是基于 servlet 进行，学习了关于 Servlet 的很多知识，在代码编写期间，出现了很多问题，但后来都得以解决，整体程序能容纳上千条数据，因为考虑到可能以后程序体量会扩大，所以在编写的时候，做了很多的解耦操作，避免后期更改代码很繁琐，例如在编写整体架构时，写了三层，servlet 层、service 层以及 mapper 层，这样写能够避免耦合，例如在需要更改数据库语句，那么只需要在 mapper 层更改即可，如果正常的都写在一起，那么会增加程序的耦合度，使修改难度大大提升，类似的解耦操作还有很多，在编写代码的过程中还存在代码冗余的问题，因为最开始写的前端 SellectAll()方法只是查询了所有数据，并没有分页查询，后来根据 id 分页查询时，和之前的 SellectAll()方法，代码有很大一部分相似，所以后来在编写过程总将两个方法整合到了一起，这样会降低其他人的阅读代码难度。

参考文献

1 国晓桐.汽车尾气与环境污染问题[J].无线互联科技,2014(02):210-210.

2 王冰,张晓莲.武汉市交通拥堵治理政策:有效与否,让公共性评价指标衡量说话[J].广西质量监督导报,2014(7):28-29.

3 吴丹.汽车“以租代购”模式浅谈[J].财经界:学术版,2016(19):136-136.

4 公正.现行经济背景下我国汽车租赁之发展研究[J].知识经济,2019(16):90-91.

5 干佳林.基于 J2EE 的知识产权服务电商平台的研究与实现[D]:[硕士学位论文].武汉:长江大学,2019.

6 张晨.国内外汽车租赁行业发展情况与趋势研究[J].现代经济信息,2016(12):346-347.

7 Paulo Dinis,Fernando Moreira da Silva. The Passenger Car in Portugal: Features of Functional Modules[J]. Elsevier B.V.,2015,3:6369-6375.

8 马良.融资租赁在汽车租赁企业车辆购置中的应用研究——以粤驰旅汽融资租赁项目为例[D]:[硕士学位论文].广州:中山大学,2011.

9 孙小兵.二手车业务对主机厂的作用和实施策略[D]:[硕士学位论文].上海:上海交通大学,2012.

10 田震.企业营销渠道变革研究——以汽车行业为例[D]:[硕士学位论文].北

京:首都经济贸易大学,2011.

11 郑施.我国汽车产业发展现状及未来发展趋势研究[J].才智,2015(07):351.

12 梁东.上海汽车租赁行业发展战略研究[D]:[硕士学位论文].上海:复旦大学，2007.

13 Jimenez, Jesus, Ruiz de Miras, Juan. Box-counting algorithm on GPU and multi-core CPU:an OpenCL cross-platform study[J].Journal of Supercomputing, 2013,65(3):1327-1352.

14 Xu Ye,Yao Lihua,Ouyang Tao,Li Jinfeng,Wang Tianfeng,Fan Zhaoqing,LuYouyong,Xie Yuntao. p53 Codon 72 polymorphism predicts the pathologic response to neoadjuvant chemotherapy in patients with breast cancer.[J]. Pubmed,2005,11(20).

15 Platforms T.The java.util.concurrent Synchronizer Framework[J].Science of Computer Programming, 2005, 58(3):293-309.

谢　辞

大学四年匆匆即逝，这四年学到了很多东西，无论学业与做人，感谢延边大学四年来对我的悉心栽培，感谢各位老师将知识倾囊相授。尤其要感谢我的导师王金祥老师对我毕业论文的指导与审核，还有金元赫老师等等老师对我们的悉心培养，老师，您们辛苦了！各位老师在传道受业解惑的同时，也培养了我们严谨的学习态度，在我的学业中给予了很多的支持与帮助，谢谢各位老师的无私奉献，“借得大江千斛水，研为翰墨颂师恩”。在此还要感谢我的朋友余思睿，感谢你一直以来的陪伴。马上要迈入人生下一个阶段，大学的时光永远都是我最美好的回忆，希望我的朋友们，都能学业有成、工作顺利！行文至此，忽然觉得很舍不得大学的同学老师们，四年大学时光，与老师同学们朝夕相处，真的是一段很难忘的回忆。和老师同学们所经历的每一分钟都清晰可见，在四年时间里老师们授予了我很多收益无穷的知识以及处理事情的方法，感谢你们出现在我人生中的四年里。在这里还要感谢我的父母，感谢你们从小到大对我的付出还有你们这么多年的支持和帮助，二十年的培养，让我每一秒都觉得很安全，由衷感谢你们的养育之恩。最后祝愿我的朋友、老师以及父母身体健康，都能开心的度过每一天。