第二章 相关技术基础

2.1 Javascript的特性与应用场景

Javascript是一种高级解释型编程语言。它基于原型，是一门多范式的语言。它支持函数式编程，面向过程编程以及面向对象编程等多种编程风格。同时，Javascript还提供了操纵文本、日期、正则表达式等多项功能的接口。与C++，java等语言不同的是，javascript不支持图形、网络与存储的I/O需求，但可以通过宿主浏览器来实现。传统意义上来讲，Javascript是一种解释型语言，但随着技术的不断发展，它也可以被即使执行。随着HTML5技术的普及，Javascript还可以被用于·游戏、移动App的开发以及后端服务，如Node.js。

一般来讲，完整的Javascript语言包括以下几部分：1.ECMAScript，它描述了语言的语法以及对象等基本信息。2.文档对象模型，它描述了如何交互处理网页内容。3.浏览器对象模型，它描述了如何与浏览器进行交互。Javascript是一种解释型语言，运行时无需对网页进行预编译，主要用来与HTML页面进行交互，工程上可以将其直接嵌入HTML，但讲Javascript脚本分离并单独导入HTML文件是更好的选项。Javascript主要被用来完成操纵HTML网页元素，响应浏览器事件以及控制cookie等操作。

与php、asp等服务器脚本语言不同，Javascript主要在客户端浏览器中执行而不需要服务器的支持，所以早期的设计中倾向于将更多的功能放在javascript中来减轻服务器的负担。但Javascript的低安全性会带来严重的危害。随着服务器性能的不断提升，现在工程中更倾向于将重要的信息放在服务器脚本中执行，而将页面交互等操作放在Javascript中执行。由于一些特殊的功能需要Javascript在客户端的支持，当前Javascript已经成为前端语言中的主流。近年来，随着Node.Js的不断发展，Javascript也被用于编写服务器脚本。

Javascript是结构化、指令式的，具有结构化语言的语句，如for，while等。Javascript是一种弱类型语言，在运行时会将一种类型的变量转换为另一种类型，如加号“+”可能会将两个操作符转化为字符串。Javascript是一种动态类型的语言，数据类型与数据关联而与变量无关，也可以使用鸭子类型等原则来实现变量类型检测。与C++的面向对象不同，Javascript的面向对象是基于原型的面向对象，这也使得Javascript能够提供函数式编程的特性。Javascript支持在函数中定义其他函数，并将这个函数作为返回值，外部函数作用域中的元素也会成为内部函数状态的一部分，同时，在Javascript中还能定义匿名函数。

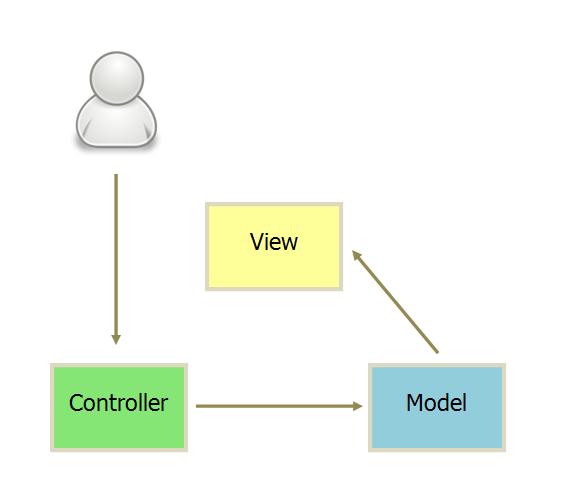
Javascript往往依赖于浏览器等运行时环境来提供方法与对象等要素，脚本通过浏览器提供的要素与网页进行交互。通常来讲，Javascript是单线程的，为了能够高效地处理并发事件，当Javascript处理到网络请求、输入输出等内容时，会异步执行，并加入回调函数来处理结果，【加一张图】这种功能使得Javascript能够在等待返回网络请求的同时处理键盘输入。ES6标准引入了Promise特性来优雅地进行异步，可以时回调函数的显示方式变得更简单。

2.2 MVC架构与MVVM架构

MVC架构时软件工程中十分常用的一种结构模式。在MVC架构中，软件由三个部分构成，模型(Model)，视图(View)，控制器(Controller)。MVC模型首先由Reenskage[见mvc维基]提出，是上世纪八十年代为程序需要smalltalk发明的架构。MVC模型的目的在于实现一种动态的程序设计，以便于后续对程序的修改，优化以及扩展，同时使代码复用成为可能。此外，MVC模型还能将复杂的模型简化，使得程序的架构更为直观。在将软件系统拆分的同时也能够将不同的部分交给不同的开发人员负责。

模型部分中封装了与业务相关的数据以及处理数据的算法。模型部分拥有直接访问数据的权力·，如访问数据库等。模型部分不依赖于视图部分以及控制器部分，即模型部分不关心如何显示数据或者如何操作数据，但模型中的变化必须被视图部分显示出来，即视图部分需要在模型中注册，如使用观察者模式等。视图部分用来进行数据的显示，一般不包含额外的处理逻辑。为了实现试图部分的实时更新，视图部分需要在模型部分注册以便于监控数据变化。控制器则用来组织两个模块以及控制软件的运行流程。它能够对用户事件等进行处理并作出回应。

在一些早期网页，如JSP网页中，HTML代码与用于数据库查询的代码往往混杂在一起，不便于修改与泛化，而MVC模型可以将其强制分开，尽管会付出一些额外的处理代价，但这能够明显地提高程序的可读性。首先，多个视图可以共享一个模型，例如用户在使用PC浏览器发送邮件的同时使用手机来查看邮件，这就需要Web与移动App同时提供视图。在MVC架构中，模型用于修改数据并将其提供给视图，视图负责将数据显示给用户，同一个模型可以被多个视图重用，大大增加了代码的重用性。其次，控制器具有自包含的性质，与视图和模型保持相对独立，所以可以方便地修改业务层逻辑。例如，当我们对数据库或数据源进行切换时，只需要对改变控制器即可。一旦控制器的更正完成，无需视图以及模型进行更改，视图就能对数据进行正确的显示。同时，控制器还提高了程序的可配置性与灵活性。控制层可以通过连接不同的模型与视图，为不同的用户提供不同的显示效果。



MVVM架构是对MVC架构的改进，有助于将用户图形界面的开发与后端业务逻辑的开发隔离开来，分为模型、视图、视图模型以及绑定器四个部分。MVVM架构是马丁弗勒的PM设计架构【】的变体，由微软工程师Peters与Cooper共同完成。MVVM使用与PM架构类似的方式抽离出视图的状态与行为。

MVVM中的模型指的是代表真实数据情况的领域模型，如数据库，或者数据访问层，如数据库的访问接口。视图指的是用户在屏幕上看到的信息。视图模型是对公共属性以及命令的抽象，而绑定器用来提供视图与视图模型之间的同步。

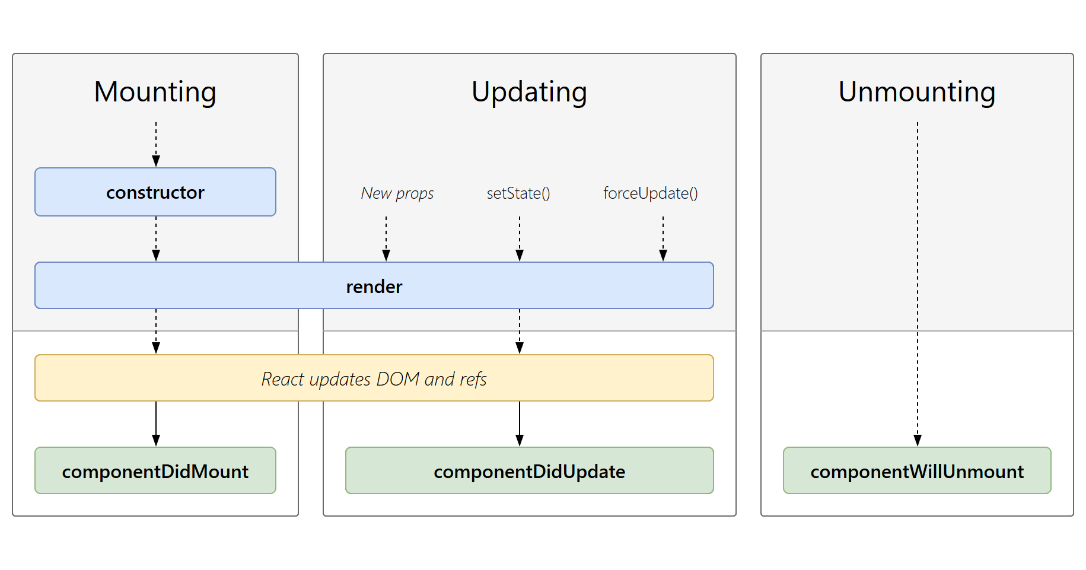
MVVM利用数据绑定函数，将所有的GUI代码从视图中移除，更好地促进视图与其他模块的分离，在MVVM架构中，往往不需要开发人员编写GUI代码，而可以使用XML等框架标记语言来代替，并将其与视图模型绑定。完成功能的分离后，就可以将XML的编写交给交互设计师而不是软件工程师。这样，我们就可以将应用程序分为多个工作流并放到多个流水线中开发以提高效率。【还能接着写】



2.3 基于MVVM架构的React

React.js是一款开源免费的Javascript工具库，它由Facebook以及由公司以及开发者组成的社区维护。React框架专注于状态管理以及将状态渲染到网页DOM的工作，充当着MVVM框架中视图模型的工作。

React具有诸多新颖的设计特性。React使用宣告式语法，开发人员只需要为每个组件设计状态视图，随后React组件会在状态变化时自动刷新，与命令式程序有着显著的区别。React代码由被称为组件的实体构成，开发人员可以对组件进行重复利用。组件可以是使用函数声明，也可以是使用类声明。【图片】。React还使用了虚拟DOM的概念，即使用某些暂存数据结构来存储页面组件，每当组件更新时不重新渲染，只对修改改过的元素进行更新，以此来提高显示效率。【wiki文献】。这一项改进显著减少了CSS更新以及重新排版的工作量。React使用Hook的方式来管理组件生命周期，允许开发人员编写对应的函数，在组件创建、销毁时执行。React还定义了JSX语法来描述组建的渲染。JSX对Javascript的功能进行了扩展，类似于HTML与Javascript的融合，允许开发人员以快捷的方式创建组件并定义DOM。



2.4 Python

Python是一种当下被各行各业广泛应用的解释类语言。相较于C++等传统编译型语言来说，Python具有动态类型系统以及自动垃圾回收机制。Python支持面向过程编程、面向对象编程，函数式编程等多种编程范式。同时，Python还具有强大的标准库以及海量的扩展库，适用于各类应用场景，能够帮助开发人员快速开发出逻辑清晰的软件系统。

Python首次由范罗苏姆在1991年发布【wiki】。Python在设计时强调代码的简洁性以及程序的可读性，能够使开发人员用更少的代码完成更复杂的任务。Python的解释器可以在多种操作系统下运行，具有跨平台开发的特点。

2.5 Flask

Flask是一个以Python为基础的轻量级Web后端服务框架，使用“微框架“的设计理念，使核心代码保持简洁，同时由用户来扩展其他功能。诸多常用功能，如表单检验工具以及资料库等都不设置在核心中，而是交由用户扩展完成。相较于其他Web框架，Flask具有诸多独特的优势，如Flask内建有开发测试用的服务器，使用RESTful技术进行请求分流，使用Jinja2完成模板引擎，能够确保cookie的安全性等。

第三章 软件需求分析

3.1 问题描述

构建一个网上求租、出租发布系统，普通网民与中介机构注册后均可以发布出租、求租（包括期望的地区等）信息，但是不同网民每人每天发布量不可以超过2条，所有用户需要填写必要信息进行注册，管理员对注册信息进行审核认证，中介机构经过审核认证后，每天可以无限制的发布出租、求租信息，网民可查询出租信息，管理员对用户信息，出租信息等进行综合管理。

3.2 功能需求

3.2.1 基本数据维护功能

提供不同用户输入，修改，发布基本数据的途径，提供管理员修改基本数据的途径。基本数据包括：用户的个人信息（用户id，用户名，密码等），用户发布的租房求租消息（地区，价位，发布时间），管理员可以对这些数据进行修改更新。

3.2.2 基本业务功能

用户注册、登录、发布租房或求租信息，随时根据数据的变化更新数据库。机构注册后要经过管理员的审核。限制普通用户的消息数量。管理员具有对消息进行操作的权限。

3.2.3 数据库管理功能

对所有租房求租信息与用户信息进行统一管理维护的功能，同时也要记录每个普通用户单日的信息发布数量。

3.3 数据流图

【我回头加上】

第四章 后端及数据库架构设计

4.1 数据库架构设计

本文采用mysql数据库进行数据服务。Mysql是一款流行的关系型数据库，将信息存储在多个关系表中，为用户提供了一种通用的接口语言，用户可以根据自身需求同时查询多个表中的不同数据。Mysql数据库由于高可靠性以及高性能，被广泛的应用于众多公司中。

本文使用两个关系表分别存储系统所用的数据。

com\_user表中存储用户信息，表中字段、类型以及意义如下表所示

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 类型 | 主键 | 非空 | 说明 | 备注 |
| User\_name | Varchar(45) | True | True | 用户名 |  |
| Password | Varchar(45) | False | True | 密码 |  |
| Phone\_number | Varchar(45) | False | True | 电话 |  |
| User\_type | bool | False | True | 用户类型 | 0：普通用户  1：中介 |
| User\_state | bool | False | Ture | 用户是否通过审核 | 0：未通过  1：通过 |

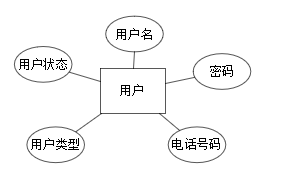
rent\_info表中存储用户发布的租房信息，表中字段、类型以及意义如下表所示

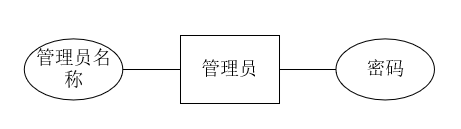
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 类型 | 主键 | 非空 | 说明 | 备注 |
| User\_name | Varchar(45) | True | True | 用户名 |  |
| Phone\_number | Varchar(45) | True | True | 电话 |  |
| Area | Varchar(45) | True | True | 租房地段 |  |
| Price | Int | True | Ture | 价位 |  |

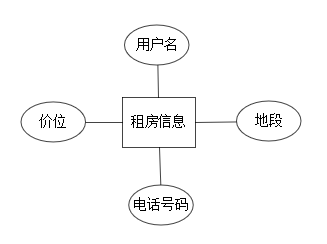
admin表中存储用户发布的租房信息，表中字段、类型以及意义如下表所示

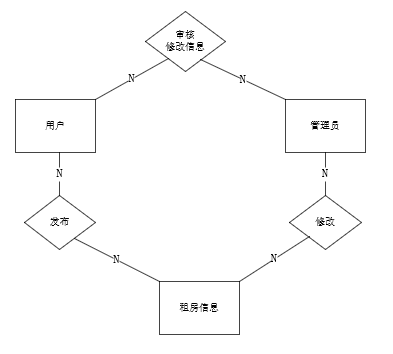
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 属性名 | 类型 | 主键 | 非空 | 说明 | 备注 |
| admin\_name | Varchar(45) | True | True | 用户名 |  |
| Password | Varchar(45) | True | True | 密码 |  |

4.2 数据库E-R图



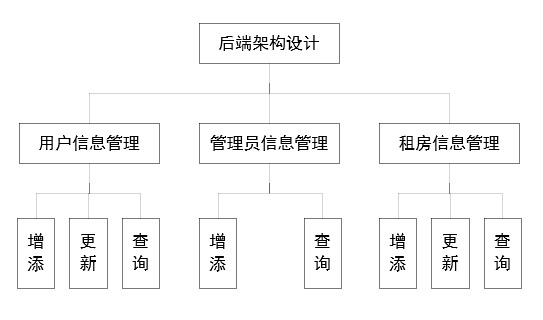






4.3 后端架构设计

本文所用的后端架构分为三个模块，分别负责对com\_user、admin以及rent\_info表进行增删改查。模块图示意如下



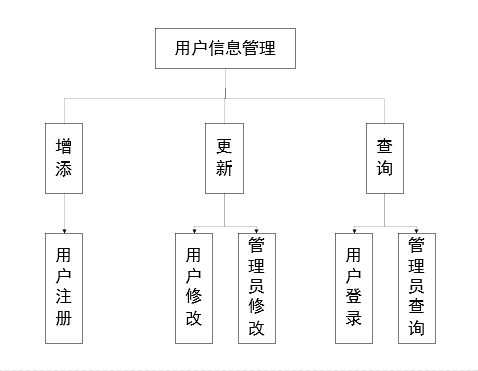
4.2.1 用户信息管理

用户信息管理模块及其子模块的模块图如下图所示。

用户注册时调用增添模块，其中包括用户名, 密码，电话号码, 用户类型等词条，注册时用户状态固定为0，即未审核状态，管理员审核后改为1。

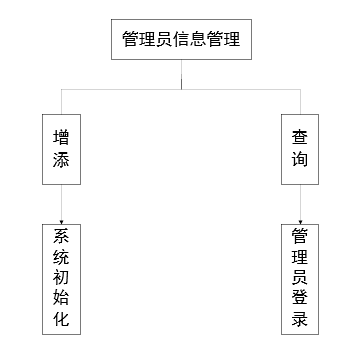
用户修改个人信息时调用更新模块，可以对密码以及电话号码进行修改，无法修改用户名以及用户类型，用户状态。管理员管理用户状态时可以对用户密码，电话号码，用户类型，用户状态等属性进行修改。

用户登录时需要对用户信息进行查询，查找是否存在满足用户名、密码以及用户类型要求的用户，存在则登陆成功，否则登陆失败。管理员在进入管理界面是时也需要对用户信息进行查询，显示所有的用户名以及密码，用户状态等，允许管理员根据用户名对用户进行查询。



4.2.2 管理员信息管理

管理员信息管理模块及其子模块的模块图如下图所示。

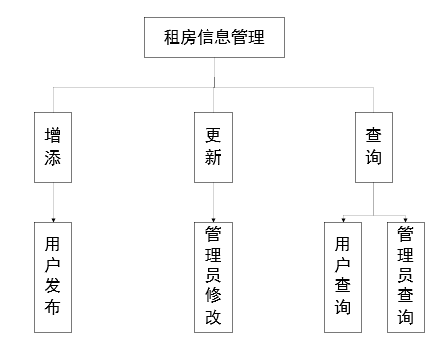


系统初始化时调用增添模块，将管理员名称以及密码加入管理员信息。

管理员登录时需要对管理员信息进行查询，查找是否存在满足管理员名称、密码要求的管理员，存在则登陆成功，否则登陆失败。

4.2.3 租房信息管理

租房信息管理模块及其子模块的模块图如下图所示。



用户发布租房信息时调用增添模块，其中包括用户名, 电话号码, 租房地段，价位等词条。

管理员求改求租信息时调用更新模块，可以对地段以及价位进行修改。

用户浏览信息时需要对租房信息进行查询，查找是否存在满足地段、价位要求的租房信息，后端返回查询的结果。管理员在进入管理界面时也需要对租房信息进行查询，显示所有租房信息的用户名，电话，地段以及价位，允许管理员与用户根据价位以及地段对租房信息进行查询。

4.4 后端与数据库的交互库

本文使用pymysql库建立后端与数据库之间的联系。Pymysql库能够根据给出的用户名与密码连接对应的数据库，执行代码中给出的sql语句并返回结果。但python所用的数据结构与sql语句的结构差距过大，需要一些函数将python中的对象转化为sql语句。

针对上述需求，本文设计了包MyDB来简化后端与数据库交互的效率。MyDB的代码架构图如图所示

4.4.1 Insert接口

MyDB的insert接口接受两个参数，table\_name代表表名，data\_list代表要插入的属性值。Insert函数内部会自行将输入参数链接为sql语句的形式，代码如下：

def insert(self, table\_name, data\_list):

status = False

err = None

cursor = self.db.cursor()

parse = "insert into %s values (%s)" % (

table\_name, DBUtils.listToString(data\_list))

try:

print(parse)

cursor.execute(parse)

self.db.commit()

status = True

err = None

except Exception as e:

status = False

err = e

return (status, err)

4.4.2 Delete接口

MyDB的delete接口接受两个参数，table\_name代表表名，attr\_list代表要删除的属性名称，data\_list代表要删除的属性值。delete函数内部会自行将输入参数链接为sql语句的形式，代码如下：

def delete(self, table\_name, attr\_list, data\_list):

status = False

err = None

cursor = self.db.cursor()

parse = "delete from %s where %s" % (

table\_name, DBUtils.twoListToCond(attr\_list, data\_list))

try:

print(parse)

cursor.execute(parse)

self.db.commit()

status = True

err = None

except Exception as e:

status = False

err = e

return (status, err)

4.4.3 Update接口

MyDB的update接口接受两个参数，table\_name代表表名，sel\_attr代表要选中的属性名称，sel\_value代表要选中的属性值，set\_attr代表要更新的属性名称，set\_value代表要更新的属性值。update函数内部会自行将输入参数链接为sql语句的形式，代码如下：

def update(self, table\_name, sel\_attr, sel\_value, set\_attr, set\_value):

status = False

err = None

res = None

cursor = self.db.cursor()

parse = "update %s set " % (table\_name,)

for i in range(len(set\_attr)-1):

parse += "%s = '%s' ," % (set\_attr[i], set\_value[i])

parse += "%s = '%s'" % (set\_attr[-1], set\_value[-1])

parse += " where "

for i in range(len(sel\_attr)-1):

parse += "%s = '%s AND ' ," % (sel\_attr[i], sel\_value[i])

parse += "%s = '%s'" % (sel\_attr[-1], sel\_value[-1])

try:

print(parse)

cursor.execute(parse)

self.db.commit()

status = True

err = None

res = cursor.fetchall()

except Exception as e:

status = False

err = e

res = None

return (status, err, res)

4.4.4 query接口

MyDB的query接口接受三个参数，table\_name代表表名，attr\_list代表要查询的属性名称，data\_list代表要查询的属性值。query函数内部会自行将输入参数链接为sql语句的形式，代码如下：

def query(self, table\_name, attr\_list, data\_list):

status = False

err = None

res = None

cursor = self.db.cursor()

parse = "SELECT \* FROM %s" % (table\_name,)

if(len(attr\_list) != 0):

parse = "SELECT \* FROM %s where %s" % (

table\_name, DBUtils.twoListToCond(attr\_list, data\_list))

try:

print(parse)

cursor.execute(parse)

self.db.commit()

status = True

err = None

res = cursor.fetchall()

except Exception as e:

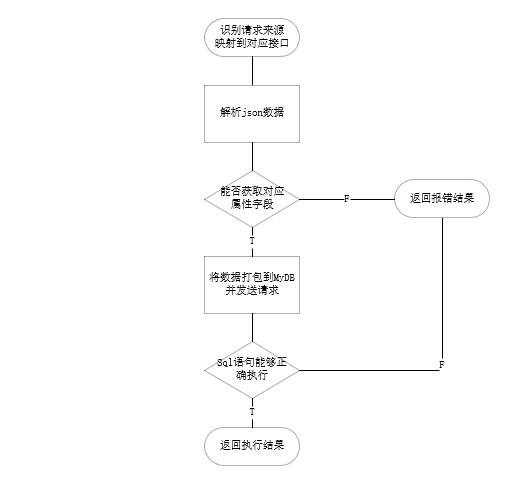
status = False

err = e

res = None

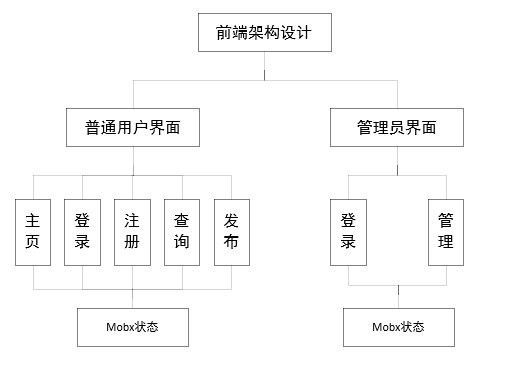
return (status, err, res)

在包MyDB的帮助下，后端处理请求的流程变得清晰简单。接收到一个数据请求后，后端首先识别请求来源，并将其根据不同来源映射到不同处理接口。处理接口接收到请求后，首先对请求体中的json数据进行解析，随后检测数据中是否存在接口要用到的属性，若存在则将属性打包后使用MyDB向数据库发送请求，得到回应后将结果返回，否则将错误信息返回，程序流程图如下。



第五章 前端架构设计

本文中使用React框架进行前端开发，并使用Mobx进行状态一致性管理，确保不同界面的显示内容能够同步更新。前端的模块图如下图所示



普通用户界面之间以及管理员界面之间分别使用Mbox进行状态同步，确保更新更够被及时反应在页面上。

5.1 用户界面

用户界面分为主页、登陆界面、注册界面、查询界面以及发布界面五个界面，本节将分别介绍其中的逻辑以及功能。

Mobx状态中保存着用户的用户名、密码、电话号码以及用户类型这几个基本信息，并会随用户的操作而不断更新。

5.1.1 用户主页

用户主页中展示了用户的用户名、密码、电话号码以及用户类型，用户可以修改密码或电话，并提交请求，系统判断密码正确后会将数据更新到数据库并同时修改mobx状态。程序流程图如下。