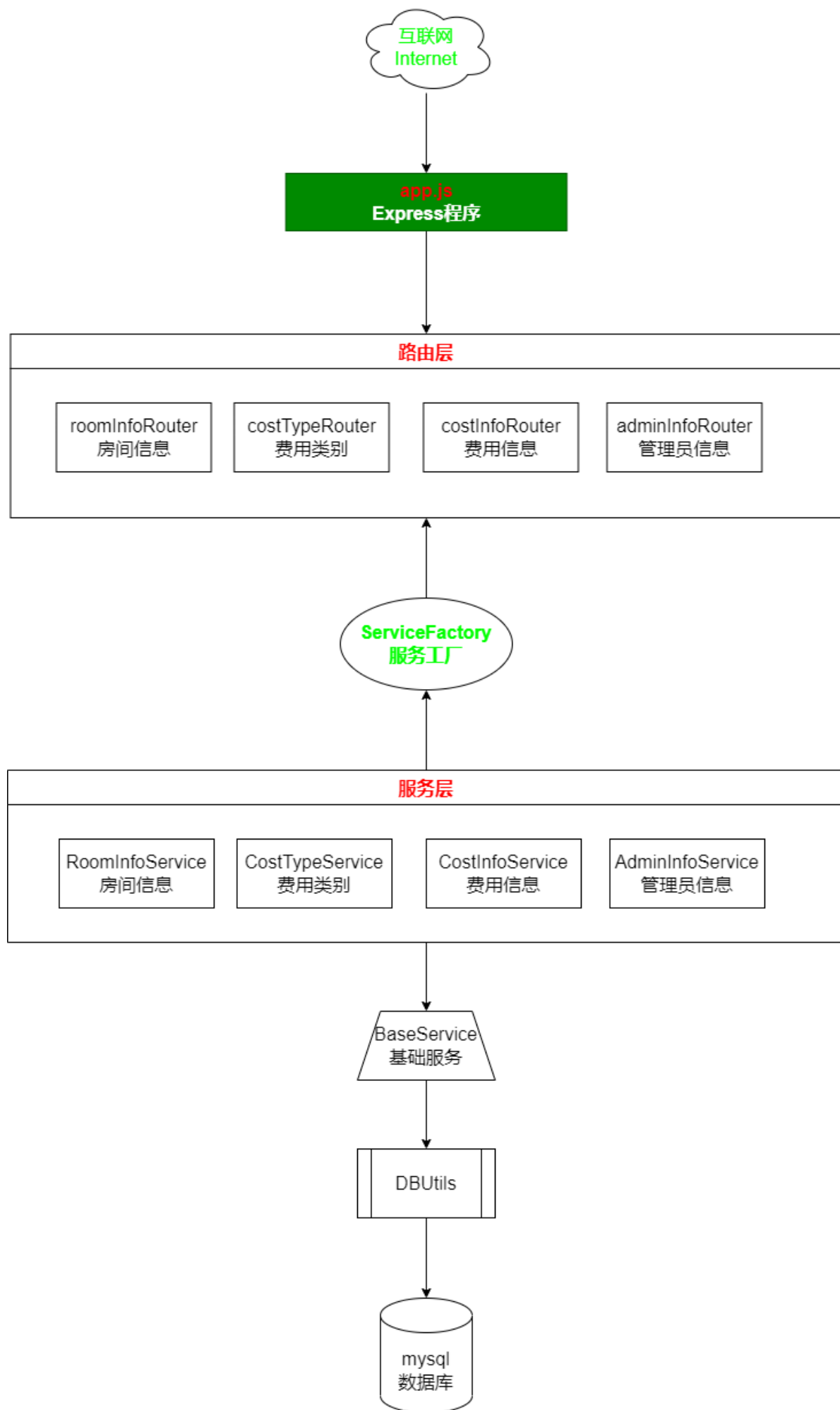


# Express与数据库项目整合

---

- 项目名称：社区物业管理系统
  - 项目平台：nodejs+mysql+bootstrap+layer+jquery
  - 项目技术：express
  - 开发人员：杨标
- 

## 项目架构图



## 一、项目初始初始化

```
1 $ npm init --yes
```

## 二、安装项目的依赖信息

```
1 $ npm install mysql2 express --save
```

## 三、创建express 程序

```
1 const http = require("http");
2 const express = require("express");
3 const app = express();
4
5
6 const server = http.createServer(app);
7 server.listen(16888, "0.0.0.0", () => {
8     console.log(`服务器启动成功`);
9 })
```

## 四、根据项目的模块创建路由

在上面的项里面创建 `routes` 目录，然后创建路由文件

- `roomInfoRouter.js`
- `costTypeRouter.js`
- `costInfoRouter.js`
- `adminInfoRouter.js`

上面的四个路由文件分别对了四个路由模块

```
1 /**
2  * @author 杨标
3  * @description roomInfo路由模块
4  */
5 const express = require("express");
6 const router = express.Router();
7
8 module.exports = router;
```

## 五、连接路由文件

每一个路由文件都要与app.js进行连接

```
1 //连接路由文件
2 app.use("/roomInfo", require("../routes/roomInfoRouter"));
3 app.use("/costType", require("../routes/costTypeRouter"));
4 app.use("/costInfo", require("../routes/costInfoRouter"));
5 app.use("/adminInfo", require("../routes/adminInfoRouter"));
```

## 六、创建mysql数据库连接的配置文件

首先在项目下面创建一个 `DBConfig.js` 的文件。这个文件放在 `config` 的目录下面

```
1  /**
2   * 本地数据库连接的配置信息
3   */
4   const localDBConfig = {
5     host: "127.0.0.1",
6     port: 3306,
7     user: "sg",
8     password: "xxxxxxx",
9     database: "community"
10  }
11  /**
12   * 远程数据库的连接
13   */
14  const remoteDBConfig = {
15    host: "www.softeem.xin",
16    port: 3306,
17    user: "dev",
18    password: "xxxxxxx",
19    database: "community"
20  }
21  module.exports = {
22    localDBConfig,
23    remoteDBConfig
24  }
```

后期把数据库的地址，用户名及密码换成自己的密码

## 七、根据数据表文件创建数据库

正常的开发应该是根据功能图来实现数据库的建模操作，再根据建模来生成数据表，目前的主流建模软件有很多， `PD/EA` 等

房间表roominfo

列名	类型	说明
id	int	主键,自增
roomname	varchar(20)	房间名称
roomarea	double	房间面积
ownername	varchar(20)	业主姓名
ownersex	varchar(2)	业主性别
IDCard	varchar(18)	身份证号
telephone	varchar(20)	手机号码
email	varchar(255)	邮箱
roomstatus	int	房间状态「自住 出租 未售 其它」

roomstatus	int	支付状态[0:未支付,1:已付,2:未付,3:未付]
------------	-----	----------------------------

### 费用类别costtype

列名	类别	说明
id	int	主键, 自增
costname	varchar(255)	费用类别的名称
price	decima(10,2)	费用的单价
desc	text	费用的说明

### 费用信息costinfo

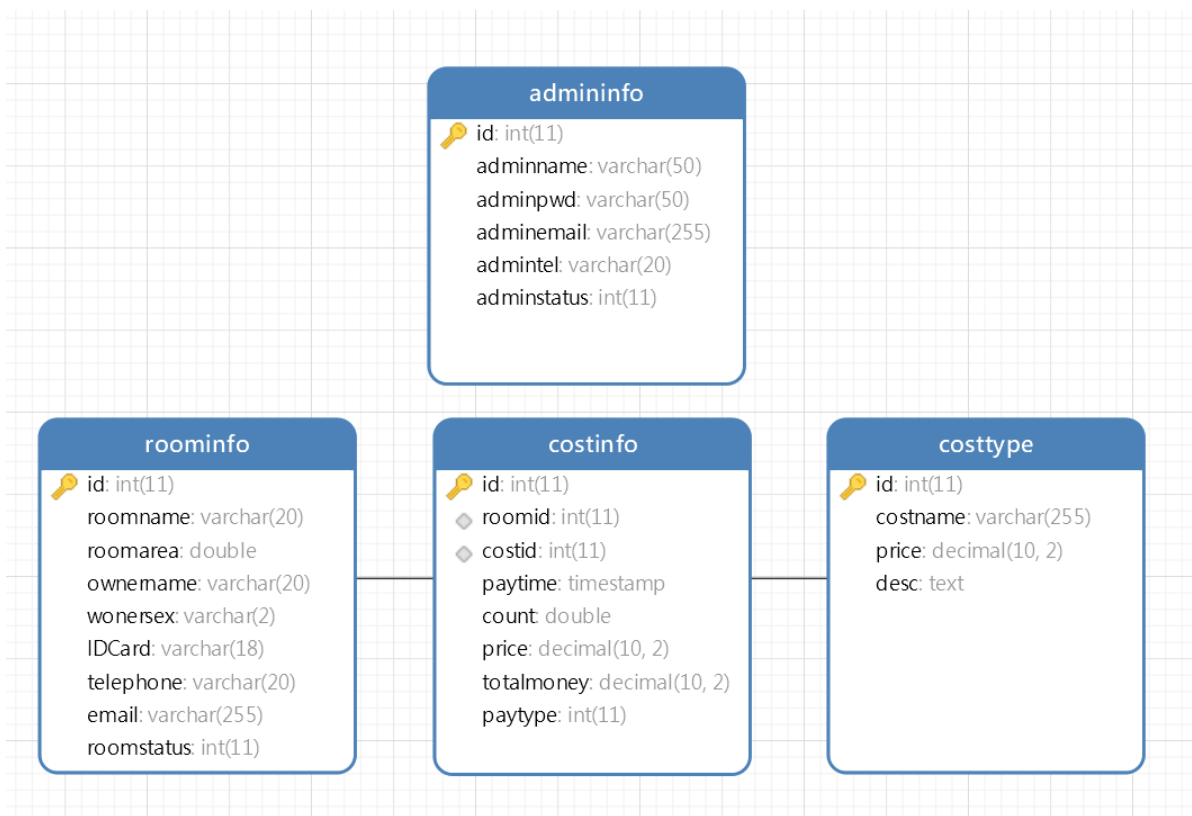
这个表就是收费信息表，也是当前系统的核心表

列名	类别	说明
id	int	主键, 自增
roomid	int	外键, 房间的编号, 来源于roominfo表
costid	int	外键, 费用类别编号, 来源于costtype表
paytime	timestamp	缴费时间, 默认为当前时间
count	double	缴费的数量, 有可能有小数
price	decima	费用类别里面的单价
totalmoney	decima	总价, 单价*数量
paytype	int	缴费方式[支付宝,微信,现金,转账]

### 管理员表admininfo

列名	类型	说明
id	int	主键自增
adminname	varchar(50)	管理员账号
adminpwd	varchar(50)	管理员密码, md5加密存储
adminemail	varchar(255)	管理员邮箱
adminitel	varchar(20)	管理员手机号
adminstatus	int	管理员状态[正常,禁用]

当我们的数据库设计完成了以后，我们就要开始在mysql里面创建数据库了



当数据库构建完成以后，我们一定要在数据表上面构建主外键的约束关系【一定是外键找主键】

这里要注意，把数据库建好了以后，一定要导出一个SQL有脚本文件

```
1  /*
2    Navicat Premium Data Transfer
3
4    Source Server      : 杨标
5    Source Server Type : MySQL
6    Source Server Version : 50540
7    Source Host        : 127.0.0.1:3306
8    Source Schema      : community
9
10   Target Server Type  : MySQL
11   Target Server Version : 50540
12   File Encoding       : 65001
13
14   Date: 17/10/2022 09:12:47
15  */
16
17  SET NAMES utf8mb4;
18  SET FOREIGN_KEY_CHECKS = 0;
19
20  -- -----
21  -- Table structure for admininfo
22  -- -----
23  DROP TABLE IF EXISTS `admininfo`;
24  CREATE TABLE `admininfo` (
25    `id` int(11) NOT NULL AUTO_INCREMENT,
26    `adminname` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
27    `adminpwd` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
```

```

28     `adminemail` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
29     `admintel` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
30     `adminstatus` int(11) NOT NULL,
31     PRIMARY KEY (`id`) USING BTREE
32 ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci ROW_FORMAT = Compact;
33
34  -- -----
35  -- Records of admininfo
36  -- -----
37
38  -- -----
39  -- Table structure for costinfo
40  -- -----
41  DROP TABLE IF EXISTS `costinfo`;
42  CREATE TABLE `costinfo` (
43      `id` int(11) NOT NULL AUTO_INCREMENT,
44      `roomid` int(11) NOT NULL,
45      `costid` int(11) NOT NULL,
46      `paytime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
47      `count` double NOT NULL,
48      `price` decimal(10, 2) NOT NULL,
49      `totalmoney` decimal(10, 2) NOT NULL,
50      `paytype` int(11) NOT NULL,
51      PRIMARY KEY (`id`) USING BTREE,
52      INDEX `roomid`(`roomid`) USING BTREE,
53      INDEX `costid`(`costid`) USING BTREE,
54      CONSTRAINT `costinfo_ibfk_1` FOREIGN KEY (`roomid`) REFERENCES `roominfo`
(`id`) ON DELETE RESTRICT ON UPDATE RESTRICT,
55      CONSTRAINT `costinfo_ibfk_2` FOREIGN KEY (`costid`) REFERENCES `costtype`
(`id`) ON DELETE RESTRICT ON UPDATE RESTRICT
56 ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci ROW_FORMAT = Compact;
57
58  -- -----
59  -- Records of costinfo
60  -- -----
61
62  -- -----
63  -- Table structure for costtype
64  -- -----
65  DROP TABLE IF EXISTS `costtype`;
66  CREATE TABLE `costtype` (
67      `id` int(11) NOT NULL AUTO_INCREMENT,
68      `costname` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
69      `price` decimal(10, 2) NOT NULL,
70      `desc` text CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
71      PRIMARY KEY (`id`) USING BTREE
72 ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci ROW_FORMAT = Compact;
73
74  -- -----

```

```

75  -- Records of costtype
76  -- -----
77
78  -- -----
79  -- Table structure for roominfo
80  -- -----
81  DROP TABLE IF EXISTS `roominfo`;
82  CREATE TABLE `roominfo` (
83    `id` int(11) NOT NULL AUTO_INCREMENT,
84    `roomname` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
85    `roomarea` double NOT NULL,
86    `ownername` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
87    `onersex` varchar(2) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
88    `IDCard` varchar(18) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
89    `telephone` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT
NULL,
90    `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
91    `roomstatus` int(11) NOT NULL,
92    PRIMARY KEY (`id`) USING BTREE
93  ) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE =
utf8mb4_general_ci ROW_FORMAT = Compact;
94
95  -- -----
96  -- Records of roominfo
97  -- -----
98
99  SET FOREIGN_KEY_CHECKS = 1;

```

## 八、封装mysql操作方法文件DBUtils

在当前项目下面新建一个 `DBUtils.js` 的文件，代码如下

```

1  /**
2   * @author 杨标
3   * @Date 2022-10-19
4   * @desc mysql数据为操作的相关内容
5   */
6  const mysql = require("mysql2");
7
8  const { localDBConfig, remoteDBConfig } = require("../config/DBConfig.js");
9
10 class DBUtils {
11   /**
12    * 获取数据库连接
13    * @returns {mysql.Connection} 获取的数据库连接
14    */
15   getConn() {
16     let conn = mysql.createConnection(remoteDBConfig);
17     conn.connect();
18     return conn;
19   }
20   /**
21    *
22    * @param {string} strSql 要执行的SQL语句

```



```

23     * @param {Array} params SQL语句里面的参数
24     * @returns {Promise<Array|mysql.ResultSetHeader>} 返回承诺携带的结果
25     */
26     executeSql(strSql, params = []) {
27         return new Promise((resolve, reject) => {
28             let conn = this.getConn();
29             conn.query(strSql, params, (error, result) => {
30                 if (error) {
31                     reject(error);
32                 }
33                 else {
34                     resolve(result);
35                 }
36                 conn.end();
37             });
38         });
39     }
40     /**
41     * 初始化参数的方法
42     */
43     paramsInit() {
44         let obj = {
45             strWhere: "",
46             ps: [],
47             /**
48             * 精确查询
49             * @param {string|number|boolean} value
50             * @param {string} name
51             * @returns {obj}
52             */
53             equal(value, name) {
54                 if (value) {
55                     this.strWhere += ` and ${name} = ? `;
56                     this.ps.push(value);
57                 }
58                 return this;
59             },
60             /**
61             * 模糊查询
62             * @param {string|number|any} value
63             * @param {string} name
64             * @returns {obj}
65             */
66             like(value, name) {
67                 if (value) {
68                     this.strWhere += ` and ${name} like ? `;
69                     this.ps.push(`%${value}%`);
70                 }
71                 return this;
72             }
73         }
74
75         return obj;
76     }
77
78 }

```

```
79
80 module.exports = DBUtils;
```

## 九、根据模块来完成Service的操作及BaseService的提取

之前我们都知道这里有4个数据表，所以应该至少有4个模块，这个Service的命名如下

1. RoomInfoService.js
2. CostTypeService.js
3. CostInfoService.js
4. AdminInfoService.js

### BaseService.js

```
1  /**
2   * @author 杨标
3   * @description 所有Service的基础类
4   * @date 2022-10-19
5   */
6  const DBUtils = require("../utils/DBUtils");
7  class BaseService extends DBUtils {
8    constructor(currentTableName) {
9      super();
10     this.currentTableName = currentTableName;
11     this.tableMap = {
12       roominfo: "roominfo",
13       costinfo: "costinfo",
14       costtype: "costtype",
15       admininfo: "admininfo"
16     }
17   }
18
19   /**
20    * 根据id删除一项
21    * @warn 后期这里要改成软删除【逻辑删除】
22    * @param {number} id
23    * @returns {Promise<boolean>} true删除成功, false删除失败
24    */
25   async deleteById(id) {
26     let strSql = `delete from ${this.currentTableName} where id = ? `;
27     let result = await this.executeSql(strSql, [id]);
28     return result.affectedRows > 0;
29   }
30
31   /**
32    * 获取所有数据
33    * @returns {Promise<Array>}
34    */
35   async getAllList() {
36     let strSql = `select * from ${this.currentTableName} `;
37     let result = await this.executeSql(strSql);
38     return result;
39   }
40 }
41
42 module.exports = BaseService;
```

RoomInfoService.js

```
1  const BaseService = require("../BaseService");
2
3  class RoomInfoService extends BaseService{
4      constructor(){
5          super("roominfo");
6      }
7  }
8
9
10 module.exports = RoomInfoService;
```

十、创建服务层工厂

在当前的项目的目录下面创建一个 `factory` 的文件夹，然后在这个文件夹下面创建 `ServiceFactory.js` 的文件

```
1  /**
2   * @author 杨标
3   * @description 服务层工厂
4   */
5
6  class ServiceFactory {
7      static createAdminInfoService() {
8          const AdminInfoService = require("../services/AdminInfoService");
9          return new AdminInfoService();
10     }
11
12     static createRoomInfoService() {
13         const RoomInfoService = require("../services/RoomInfoService");
14         return new RoomInfoService();
15     }
16
17     static createCostTypeService() {
18         const CostTypeService = require("../services/CostTypeService");
19         return new CostTypeService();
20     }
21
22     static createCostInfoService() {
23         const CostInfoService = require("../services/CostInfoService");
24         return new CostInfoService();
25     }
26 }
27 module.exports = ServiceFactory;
```

十一、完成下面页面的数据请求操作

房间信息列表

房主姓名

输入姓名

房主编号

输入编号

手机号码

手机号码

查询

新增

编辑

删除

导出为excel

全选	业主姓名	性别	房间编号	房间面积	房间状态	业主账号	身份证号	手机号码	业主邮箱
----	------	----	------	------	------	------	------	------	------

在上面的页面里面，我们可以看到，首先就会有一个查询界面，同时还会有新增，修改，删除操作，这是一个房间的模块，所以我们要针对性对性的对房间来进行功能

## 1. 完成查询的操作

这个查询的操作肯定是查询房间的信息，所以我们要针对性去找房间的模块 `roomInfo`，与房间模块相关的联的主要有2个

- `roomInfoRouter.js` 用于处理房间的请求的
- `RoomInfoService.js` 用于操作房间表的数据库操作

`roomInfoRouter.js`

```
1
2  router.get("/getList", async (req, resp) => {
3      //我希望请求这个方法的时候，能够返回房间信息列表
4      //数据在哪里，数据库
5      //操作roominfo的表，roominfo表对应RoomInfoService
6      try {
7          let roomInfoService = ServiceFactory.createRoomInfoService();
8          let roomInfoList = await roomInfoService.getAllList();
9          resp.json(roomInfoList);
10     } catch (error) {
11
12     }
13 });
14
```

当我们请求数据的时候，这个时候就可以得以json数据，如下所示

```
[
  {
    "id": 6,
    "roomname": "1-1-106",
    "roomarea": 6777,
    "ownername": "0",
    "ownersex": "男",
    "IDCard": "420312199505018726",
    "telephone": "18520217515",
    "email": "hhhhh@qq.com",
    "roomstatus": 1
  },
  {

```

但是这个做并不好，有一些问题

这样的格式其实并不好，我们希望返回一个固定的格式

## 2. 返回固定格式的json

成功的时候

```
1  {
2      status: "success",
3      msg: "获取数据成功",
4      data: [
5          //真正的数据
6      ]
7  }
```

## 失败的时候

```
1  {
2      status:"fail",
3      msg:"服务器错误",
4      data:null
5  }
```

我们希望每次返回数据的时候都是一个固定的格式，这样前端在接收数据的时候就会非常方便，直接判断某一个字段就知道这次的请求是否成功了

在当前的项目下面创建了一个 `model` 的文件夹，在下面创建一个 `ResultJson.js` 的文件，代码如下

model有模型的意思，在java里面一般叫DTO，在MVC里面，指的就是M，它一般用于对象的统一格式传输

```
1  /**
2   * 定义统一的JSON返回格式
3   * @author 杨标
4   * @date 2022-10-20
5   */
6
7  class ResultJson {
8      /**
9       *
10      * @param {boolean} status
11      * @param {*} msg
12      * @param {*} data
13      */
14      constructor(flag, msg, data = []) {
15          this.status = flag ? "success" : "fail";
16          this.msg = msg;
17          this.data = data;
18      }
19  }
20  module.exports = ResultJson;
```

现在将代码更改如下

```
1
2  router.get("/getList", async (req, resp) => {
3      //我希望请求这个方法的时候，能够返回房间信息列表
4      //数据在哪里，数据库
5      //操作roominfo的表，roominfo表对应RoomInfoService
6      try {
7          let roomInfoService = ServiceFactory.createRoomInfoService();
8          let roomInfoList = await roomInfoService.getAllList();
9          let resultJson = new ResultJson(true, "房间信息获取成功", roomInfoList);
10         resp.json(resultJson);
11     } catch (error) {
12         let resultJson = new ResultJson(false, "服务器错误");
13         resp.json(resultJson);
14     }
15 });
```

这个时候返回的数据格式就是固定的格式了，方便后期前端进行相应的操作

## 请求失败的时候

```
{
  "status": "fail",
  "msg": "服务器错误",
  "data": []
}
```

## 请求成功的时候

```
{
  "status": "success",
  "msg": "房间信息获取成功",
  "data": [
    {
      "id": 6,
      "roomname": "1-1-106",
      "roomarea": 6777,
      "ownername": "0",
      "ownersex": "男",
      "IDCard": "420312199505018726",
      "telephone": "18520217515",
      "email": "hhhhh@qq.com",
      "roomstatus": 1
    }
  ],
}
```

通过上面的对比，我们可以发现，无论是成功了，还是失败了，我们都可以看到一个固定格式的 `json` 字符串

## 3.根据查询条件完成查询操作

### 房间信息列表

房主姓名  房主编号  手机号码

<input type="checkbox"/> 全选	业主姓名	性别	房间编号	房间面积	房间状态	业主账号	身份证号	手机号码	业主邮箱
-----------------------------	------	----	------	------	------	------	------	------	------

我们可以看到，我们真要做的是根据查询条件来查询内容，不是获取所有数据，而我们之前写的代码是调用了 `let roomInfoList = await roomInfoService.getAllList();` 获取了所有的数据，这是不对的，所以我们应该要重新写个方法，根据查询条件来获取结果

而数据库的操作应该在 `service` 里面，现在我们操作 `roominfo` 的数据表，那么应该找 `RoomInfoService`

但是 `RoomInfoService` 没有根据查询条件获取数据的方法，所以我们要手动的写一个

```
1  /**
2   * @author 杨标
3   * @description roomInfo模块的操作
4   */
5
6  const BaseService = require("../BaseService");
7
8  class RoomInfoService extends BaseService {
9    constructor() {
10      super("roominfo");
```

```

11     }
12
13     /**
14      * 根据条件查询
15      * @param {{ roomname, ownername, telephone }} param0 查询参数
16      * @returns {Promise<Array>} 查询结果
17      */
18     async getList({ roomname, ownername, telephone }) {
19         let strSql = `select * from ${this.currentTableName} where 1=1 `;
20         let { strWhere, ps } = this.paramsInit()
21             .like(roomname, "roomname")
22             .like(ownername, "ownername")
23             .equal(telephone, "telephone");
24         strSql += strWhere;
25         let result = await this.executeSql(strSql, ps);
26         return result;
27     }
28 }
29
30
31 module.exports = RoomInfoService;

```

回到前端页面

```

1 <script>
2     $(function() {
3         let url = "http://127.0.0.1:16888/roomInfo/getList";
4         const getRoomInfoList = () => {
5             //ajax向后台发请求
6             $.get(url, result => {
7                 console.log(result);
8             });
9         }
10        getRoomInfoList();
11    })
12 </script>

```



当我们看到一个像这样的错误的时候，这个错误就是典型的Ajax跨域错误

## 十二、解决Ajax跨域的问题

要解决这个问题，我们先要弄清楚另一个问题，为什么会有Ajax跨域的错误？什么是ajax跨域

Ajax不能跨域是因为所有的BOM不能跨域，而BOM不能跨域的原因是因为浏览器不能跨域，浏览器不能跨域的原因是因为有一个**同源策略**

## 一种约定

同源策略（Same origin policy）是一种约定，它是浏览器最核心也最基本的安全功能，如果缺少了同源策略，则浏览器的正常功能可能都会受到影响。可以说Web是构建在同源策略基础之上的，浏览器只是针对同源策略的一种实现。

## 同源策略也叫同域策略

同源策略，它是由Netscape提出的一个著名的安全策略。

当一个浏览器的两个tab页中分别打开来 百度和谷歌的页面

当浏览器的百度tab页执行一个脚本的时候会检查这个脚本是属于哪个页面的，

即检查是否同源，只有和百度同源的脚本才会被执行。<sup>[1]</sup>

如果非同源，那么在请求数据时，浏览器会在控制台中报一个异常，提示拒绝访问。

同源策略是浏览器的行为，是为了保护本地数据不被JavaScript代码获取回来的数据污染，因此拦截的是客户端发出的请求回来的数据接收，即请求发送了，服务器响应了，但是无法被浏览器接收。

现在我们发现，当前网页的址是 `http://127.0.0.1:5007/roomInfoList.html`，而 ajax 要请求的页面则是 `http://127.0.0.1:16888/roomInfo/getList`，这个时候两个地址的域 `origin` 不是相同的，这样就存在一个跨域现象，跨域以后的访问是受到了浏览器同源策略的限制的，所以ajax的请求返回不了结果

要解决这个Ajax跨域的问题有很多办法

### 1. CORS 跨域资源共享

CORS 全称 `corss origin resource share` 跨域资源共享

```
1 resp.setHeader("Access-Control-Allow-Origin", "*");
2 resp.setHeader("Access-Control-Allow-Methods",
  "GET,POST,PUT,DELETE,OPTIONS");
3 resp.setHeader("Access-Control-Aloow-Headers", "Content-Type");
```

### 2. http反向代理【在vue里面讲】

### 3. jsonp 【面试宝典了解就可以了，基本不用】

按照上面的方式，我们可以在返回数据之前，添加请求头来解决问题

```
1
2 //http://127.0.0.1:16888/roomInfo/getList?roomname=1-1-201&ownername=0&telephone=18520217515
3 router.get("/getList", async (req, resp) => {
4   //我希望请求这个方法的时候，能够返回房间信息列表
5   //数据在哪里，数据库
6   //操作roominfo的表，roominfo表对应RoomInfoServcie
7   // 所有get请求的参数都在req.query
8   // query就是?后面所有的参数形成的对象
9   /**
10    * {
11    *   roomname:"1-1-201",
12    *   ownername:"0",
13    *   telephone:"18520217515"
14    * }
```



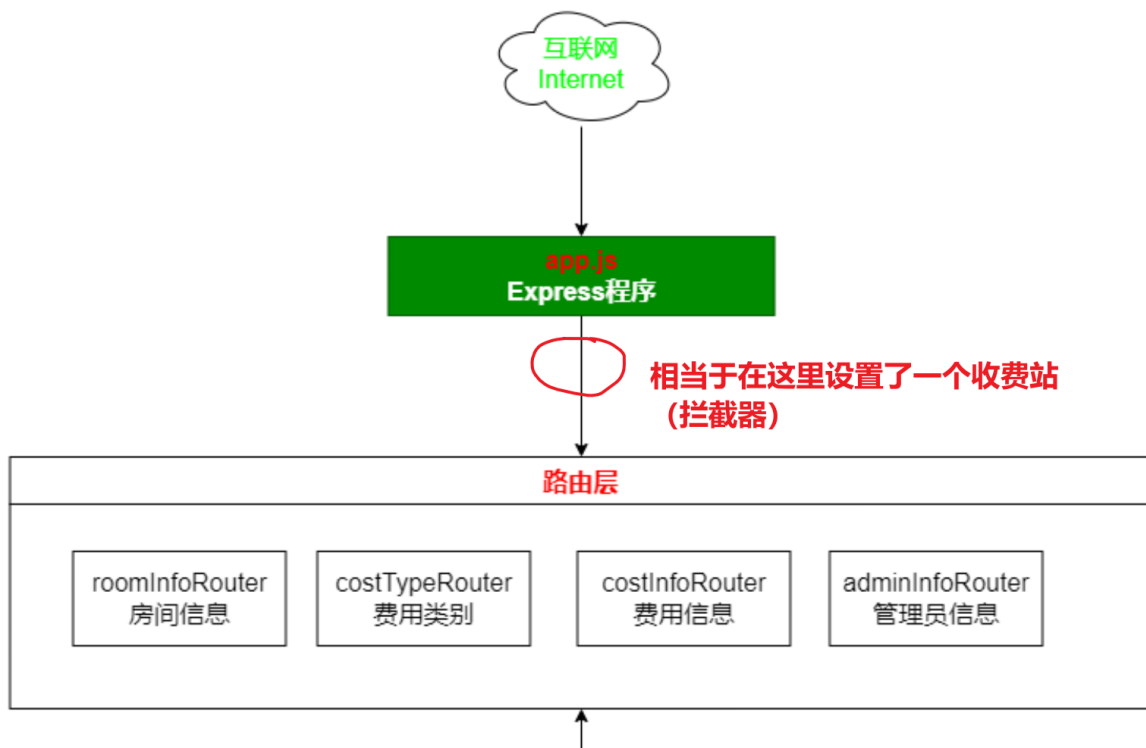
```

15     */
16     try {
17         let roomInfoService = ServiceFactory.createRoomInfoService();
18         let roomInfoList = await roomInfoService.getList(req.query);
19         let resultJson = new ResultJson(true, "房间信息获取成功", roomInfoList);
20         //颁发跨域能行证
21         resp.setHeader("Access-Control-Allow-Origin", "*");
22         resp.setHeader("Access-Control-Allow-Methods",
23             "GET,POST,PUT,DELETE,OPTIONS");
24         resp.setHeader("Access-Control-Allow-Headers", "Content-Type");
25         resp.json(resultJson);
26     } catch (error) {
27         let resultJson = new ResultJson(false, "服务器错误");
28         resp.json(resultJson);
29     });

```

但是这会存在一个现象就是每次在每回数据的时候，我都要添加响应头，怎么办呢？

## 十三、拦截器



所有的请求最终都会经过 `app.js` 的，所以我们只要在 `app.js` 这个地方设置一个拦截，那第每次的请求都会被拦截

`app.js`

```
1 //添加一个拦截器
2 app.use((req, resp, next) => {
3     //req: 浏览器到服务器的请求
4     //resp:服务器返回浏览器的
5     //next:是否放行
6     console.log("我是拦截器，我在拦你， 你走不动了");
7     //颁发跨域能行证
8     resp.setHeader("Access-Control-Allow-Origin", "*");
9     resp.setHeader("Access-Control-Allow-Methods",
10     "GET,POST,PUT,DELETE,OPTIONS");
11     resp.setHeader("Access-Control-Allow-Headers", "Content-Type");
12     next();
13 });
```

上图的代码就是通过器来实现添加请求头，以期达到 `ajax` 跨域访问