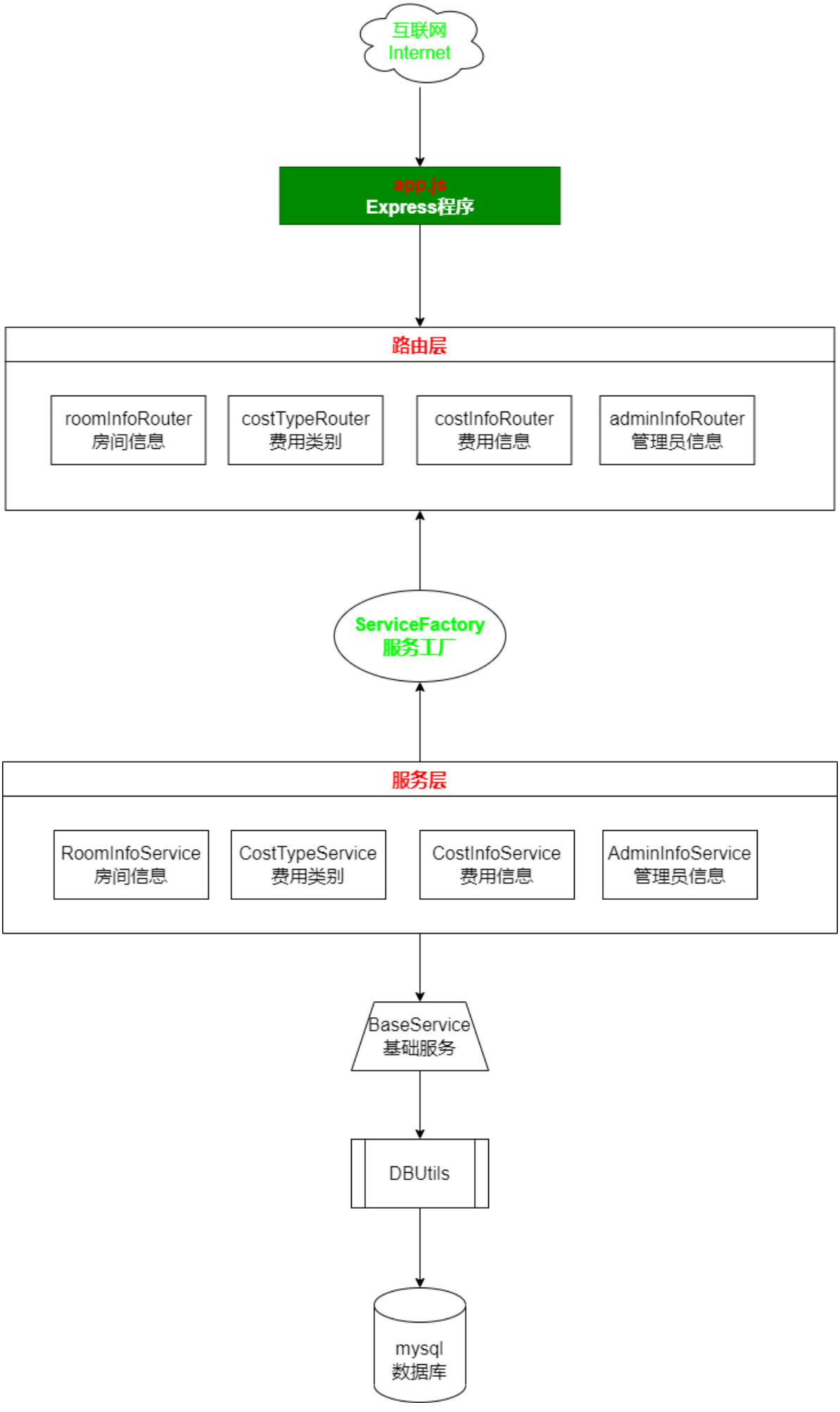
# Express与数据库项目整合

- 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


互联网
Internet
app.js
Express程序
路由层
roomInfoRouter
房间信息
costTypeRouter
费用类别
costInfoRouter
费用信息
adminInfoRouter
管理员信息
ServiceFactory
服务工厂
服务层
RoomInfoService
房间信息
CostTypeService
费用类别
CostInfoService
费用信息
AdminInfoService
管理员信息
BaseService
基础服务
DBUtils
mysql
数据库

## 一、项目初始始化

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

## 二、安装项目的依赖信息

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

## 三、创建express 程序

```
const http = require("http");
const express = require("express");
const app = express();


const server = http.createServer(app);
server.listen(16888,"0.0.0.0",()=>{
    console.log(`服务器启动成功`);
})
```

## 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

```
/**
 * @author 杨标
 * @description roomInfo路由模块
 */
const express = require("express");
const router = express.Router();

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

```
//连接路由文件
app.use("/roomInfo", require("./routes/roomInfoRouter"));
app.use("/costType", require("./routes/costTypeRouter"));
app.use("/costInfo", require("./routes/costInfoRouter"));
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

```
/**
 * 本地数据库连接的配置信息
 */
 const localDBConfig = {
    host: "127.0.0.1",
    port: 3306,
    user: "sg",
    password: "xxxxxxx",
    database: "community"
}
/**
 * 远程数据库的连接
 */
const remoteDBConfig = {
    host: "www.softeem.xin",
    port: 3306,
    user: "dev",
    password: "xxxxxx",
    database: "community"
}
module.exports = {
    localDBConfig,
    remoteDBConfig
}
```

> 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，目前的主流的建模软件有很多，`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 | 房间状态[占住,出租,维修,其它] |

**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**

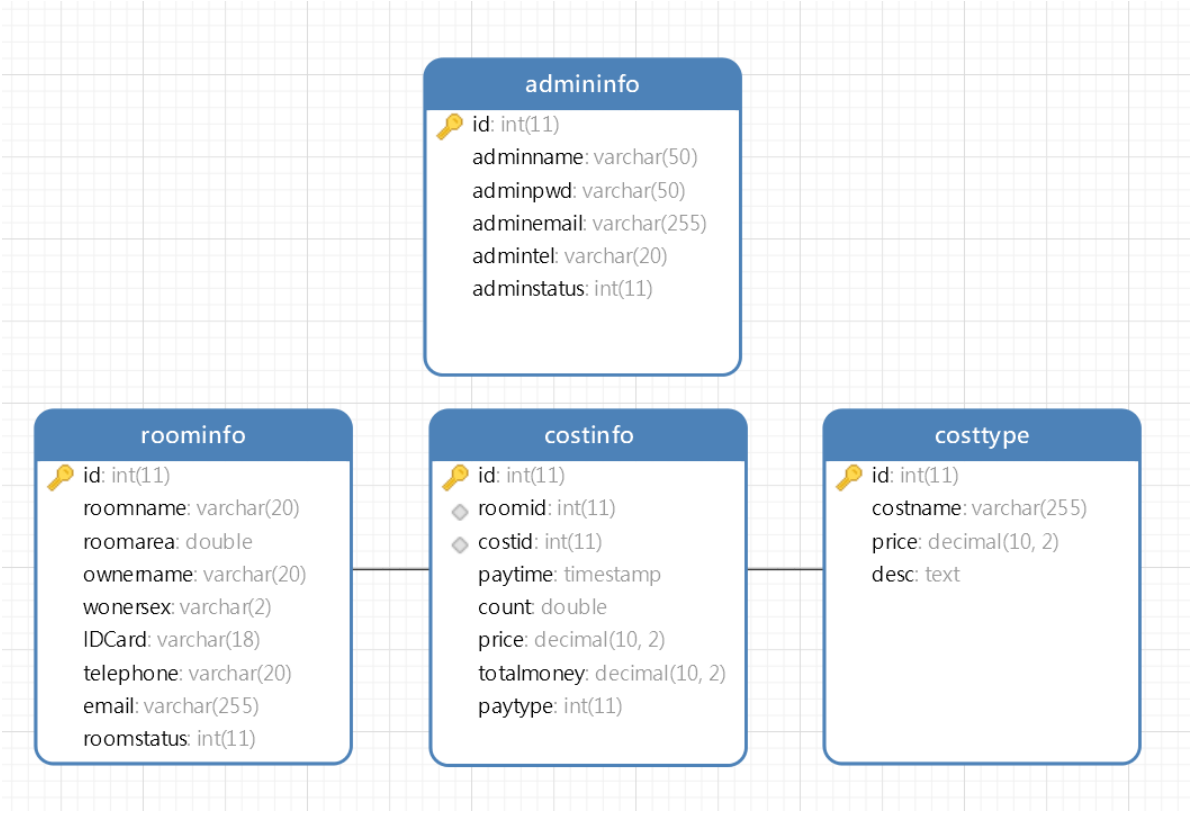
> 这个表就是收费信息表，也是当前系统的核心表

| 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 |
| admintel | 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

```
/*
 Navicat Premium Data Transfer

 Source Server         : 杨标
 Source Server Type    : MySQL
 Source Server Version : 50540
 Source Host           : 127.0.0.1:3306
 Source Schema         : community

 Target Server Type    : MySQL
 Target Server Version : 50540
 File Encoding         : 65001

 Date: 17/10/2022 09:12:47
*/

SET NAMES utf8mb4;
SET FOREIGN_KEY_CHECKS = 0;

-- ----------------------------
-- Table structure for admininfo
-- ----------------------------
DROP TABLE IF EXISTS `admininfo`;
CREATE TABLE `admininfo`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `adminname` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `adminpwd` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
```

```
  `adminemail` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `admintel` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `adminstatus` int(11) NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Compact;

-- ----------------------------
-- Records of admininfo
-- ----------------------------

-- ----------------------------
-- Table structure for costinfo
-- ----------------------------
DROP TABLE IF EXISTS `costinfo`;
CREATE TABLE `costinfo`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `roomid` int(11) NOT NULL,
  `costid` int(11) NOT NULL,
  `paytime` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `count` double NOT NULL,
  `price` decimal(10, 2) NOT NULL,
  `totalmoney` decimal(10, 2) NOT NULL,
  `paytype` int(11) NOT NULL,
  PRIMARY KEY (`id`) USING BTREE,
  INDEX `roomid`(`roomid`) USING BTREE,
  INDEX `costid`(`costid`) USING BTREE,
  CONSTRAINT `costinfo_ibfk_1` FOREIGN KEY (`roomid`) REFERENCES `roominfo` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT,
  CONSTRAINT `costinfo_ibfk_2` FOREIGN KEY (`costid`) REFERENCES `costtype` (`id`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Compact;

-- ----------------------------
-- Records of costinfo
-- ----------------------------

-- ----------------------------
-- Table structure for costtype
-- ----------------------------
DROP TABLE IF EXISTS `costtype`;
CREATE TABLE `costtype`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `costname` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `price` decimal(10, 2) NOT NULL,
  `desc` text CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Compact;

-- ----------------------------
```

```
-- Records of costtype
-- ----------------------------

-- ----------------------------
-- Table structure for roominfo
-- ----------------------------
DROP TABLE IF EXISTS `roominfo`;
CREATE TABLE `roominfo`  (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `roomname` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `roomarea` double NOT NULL,
  `ownername` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `onersex` varchar(2) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `IDCard` varchar(18) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `telephone` varchar(20) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `email` varchar(255) CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci NOT NULL,
  `roomstatus` int(11) NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB AUTO_INCREMENT = 1 CHARACTER SET = utf8mb4 COLLATE = utf8mb4_general_ci ROW_FORMAT = Compact;

-- ----------------------------
-- Records of roominfo
-- ----------------------------

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

```
/**
 * @author 杨标
 * @Date 2022-10-19
 * @desc mysql数据为操作的相关内容
 */
const mysql = require("mysql2");

const { localDBConfig, remoteDBConfig } = require("../config/DBConfig.js");

class DBUtils {
    /**
     * 获取数据库连接
     * @returns {mysql.Connection} 获取的数据库连接
     */
    getConn() {
        let conn = mysql.createConnection(remoteDBConfig);
        conn.connect();
        return conn;
    }
    /**
     *
     * @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Servcie的命名如下

1. `RoomInfoService.js`
2. `CostTypeService.js`
3. `CostInlfoService.js`
4. `AdminInfoService.js`

**BaseService.js**

```
1   /**
2    * @author 杨标
3    * @description 所有Servcie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

```
const BaseService = require("./BaseService");

class RoomInfoService extends BaseService{
    constructor(){
        super("roominfo");
    }
}


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

```
/**
 * @author 杨标
 * @description 服务层工厂
 */

class ServiceFactory {
    static createAdminInfoService() {
        const AdminInfoServcie = require("../services/AdminInfoService");
        return new AdminInfoServcie();
    }

    static createRoomInfoServcie() {
        const RoomInfoService = require("../services/RoomInfoService");
        return new RoomInfoService();
    }

    static createCostTypeService() {
        const CostTypeService = require("../services/CostTypeServcie");
        return new CostTypeService();
    }

    static createCostInfoService() {
        const CostInfoService = require("../services/CostInfoServcie");
        return new CostInfoService();
    }
}
module.exports = ServiceFactory;
```

## 十一、完成下面页面的数据请求操作

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**

```
/**
 * @author 杨标
 * @description roomInfo路由模块
 */
const express = require("express");
const router = express.Router();
const ServiceFactory = require("../factory/ServiceFactory");

//http://127.0.0.1:16888/roomInfo/getList?ownername=杨标roomname=1-1-203&telephone=18627387654
router.get("/getList",async (req, resp) => {
    // req.query就会得到?后面的请求参数，我们拿到这个参数以后要在数据库当中去查询
    /**
     * req.query应该就是下面的对象
     * {
     *     ownername:"杨标",
     *     roomname:"1-1-203",
     *     telephone:18627387654
     * }
     */
    try {
        let roomInfoServcie = ServiceFactory.createRoomInfoServcie();
        let roomInfoList = await roomInfoServcie.getList(req.query);
        resp.json(roomInfoList);
    } catch (error) {
        resp.json({
            msg:"服务器错误",
            status:"fail"
        })
    }
});
module.exports = router;
```

**RoomInfoService.js**

```
/**
 * @author 杨标
 * @description roomInfo模块的操作
 */

const BaseService = require("./BaseService");

class RoomInfoService extends BaseService {
    constructor() {
```

```
        super("roominfo");
    }
    /**
     * 根据条件查询房间信息
     * @param {{ ownername, roomname, telephone }} param0 查询参数
     * @returns {Promise<Array>}
     */
    async getList({ ownername, roomname, telephone }) {
        let strSql = ` select * from ${this.currentTableName} where 1 = 1 `;
        let { strWhere, ps } = this.paramsInit()
            .like(ownername, "ownername")
            .like(roomname, "roomname")
            .like(telephone, "telephone");
        strSql += strWhere;
        let result = await this.executeSql(strSql, ps);
        return result;
    }
}


module.exports = RoomInfoService;
```

当完成了上面的操作以后，我们就去试一下