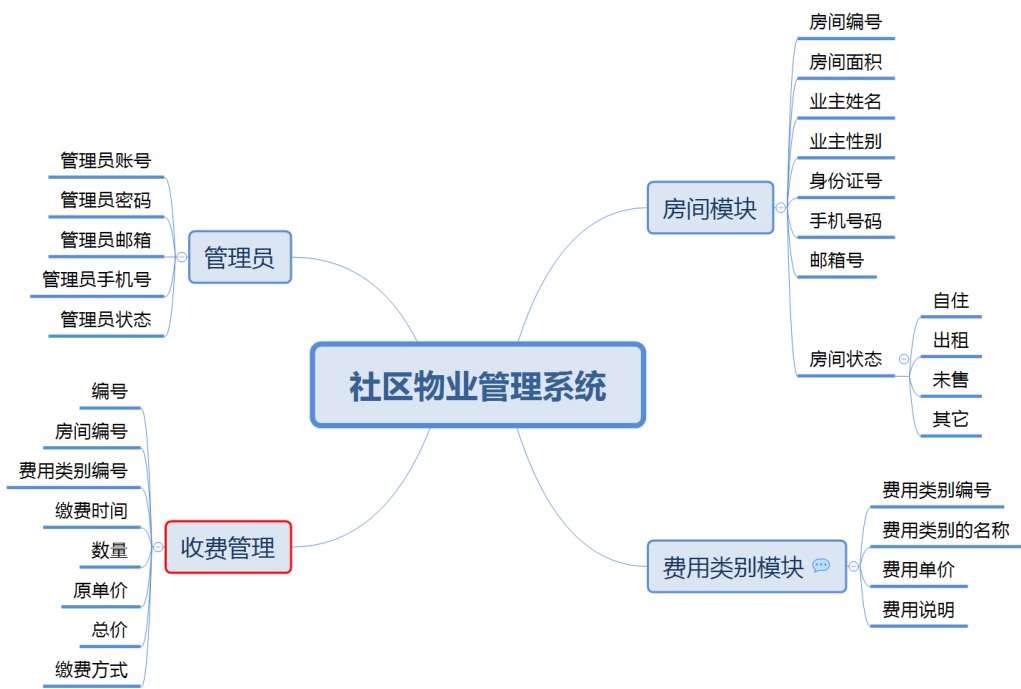


# nodejs数据库项目搭建

- 项目名称：社区物业管理系统
- 开发平台：nodejs+mysql+es6+bootstrap+layer+jquery
- 授课方式：项目式授课

## 功能分析【思维导图】



## 数据库设计

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 列名	varchar(255) 类型	邮箱 说明
roomstatus	int	房间状态,[自住,出租,未售,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

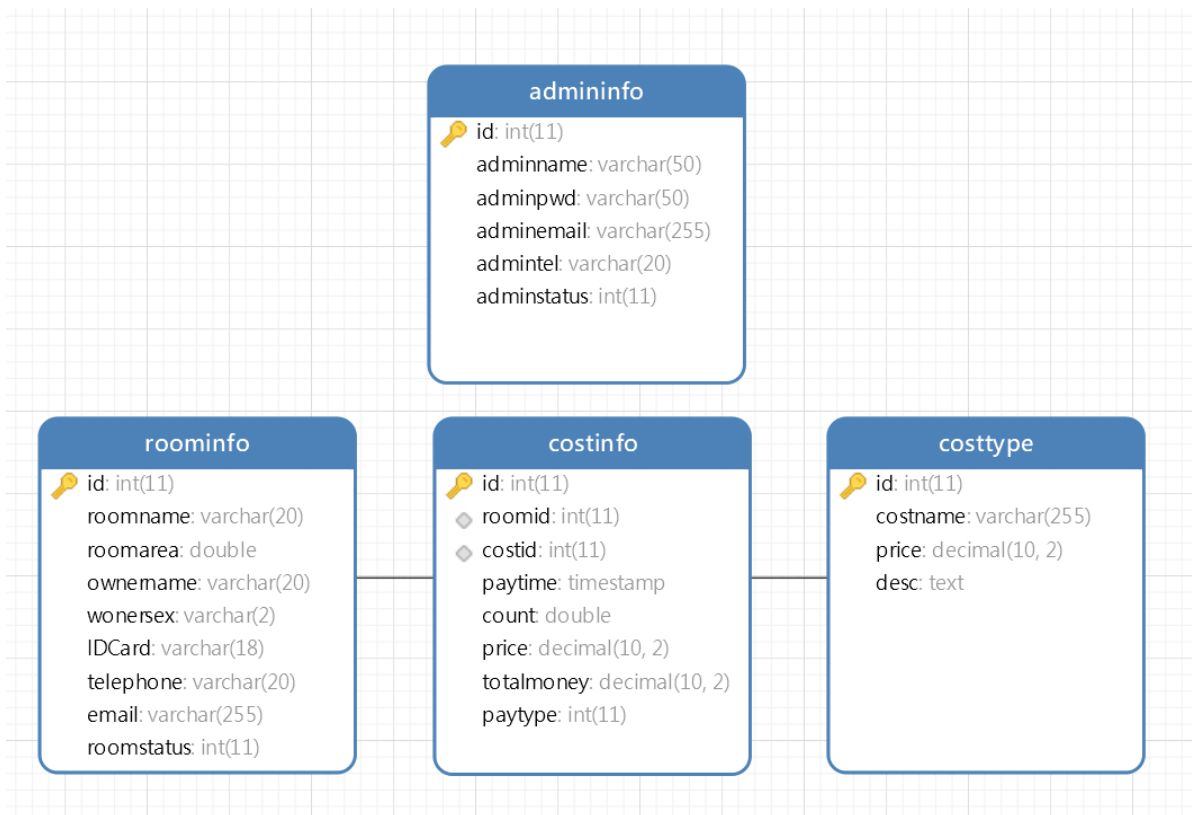
这个表就是收费信息表，也是当前系统的核心表

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

```

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

```

```

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      `wonersex` 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;
100

```

## 初始化项目

当前项目的名称使用的是 `community`，所以我们系统的名称就叫 `community manager`

新建完项目以后，我们就可以初始化了

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

```
1  {
2    "name": "community-manager",
3    "version": "1.0.0",
4    "description": "社区物业管理系统",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "keywords": ["node.js", "mysql", "express", "community"],
10   "author": "杨标",
11   "license": "ISC"
12 }
```

## 安装依赖包

当前项目因为要使用mysql的数据库，所以我们首先需要安装一引起依赖包

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

## 导入数据封装对象

之前的时候，我们已经学习过怎么样使用 `node.js` 去连接mysql的数据库

这里连接数据库的时候有一个小细节，一定要有一个账号去管理当前的数据库

我们在当前的项目下面新建了一个 `config` 的文件夹，创建了一个 `DBConfig.js` 的文件

```
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: "123456",
9    database: "community"
10 }
11
12 /**
13 * 远程数据库的连接
14 */
15 const remoteDBConfig = {
16   host: "www.softeem.xin",
17   port: 3306,
18   user: "dev",
19   password: "123456",
20   database: "community"
```

```

21   }
22
23   module.exports = {
24     localDBConfig,
25     remoteDBConfig
26   }

```

当我们把数据库的配置信息写好了以后，我们就可以开始进行相应的数据库操作的封装了，我们在项目下面新建了一个 `utils` 的文件夹，然后在这个文件夹的下面新建了一个 `DBUtils.js` 的文件，代码封装如下

```

1   /**
2    * @author 杨标
3    * @Date 2022-10-17
4    * @desc mysql数据为操作的相关内容
5    */
6   const mysql = require("mysql2");
7   const { localDBConfig } = require("../config/DBConfig.js");
8
9   class DBUtils {
10     /**
11      * 获取数据库连接
12      * @returns {mysql.Connection} 获取的数据库连接
13      */
14     getConn() {
15       let conn = mysql.createConnection(localDBConfig);
16       conn.connect();
17       return conn;
18     }
19     /**
20      *
21      * @param {string} strSql 要执行的SQL语句
22      * @param {Array} params SQL语句里面的参数
23      * @returns {Promise<Array|mysql.ResultSetHeader>} 返回承诺携带的结果
24      */
25     executeSql(strSql, params = []) {
26       return new Promise((resolve, reject) => {
27         let conn = this.getConn();
28         conn.query(strSql, params, (error, result) => {
29           if (error) {
30             reject(error);
31           }
32           else {
33             resolve(result);
34           }
35           conn.end();
36         });
37       });
38     }
39   }
40
41   module.exports = DBUtils;

```

# 编写服务层

服务层就是用于操作数据库的，为前面的程序提供的服务的

首页 / 居民信息列表

居民信息列表

房主姓名 输入姓名 房间编号 输入房间号 手机号码 输入手机号码 查询

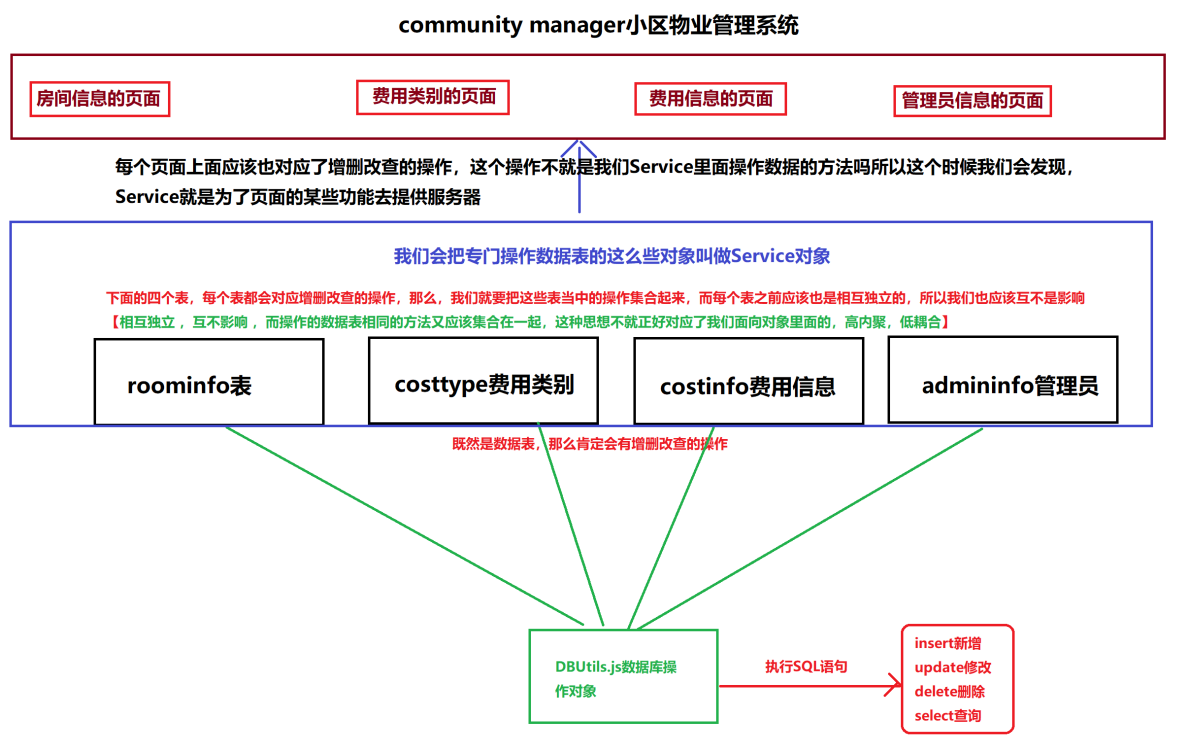
新增 编辑 删除 导出Excel

<input checked="" type="checkbox"/>	业主姓名	性别	房间编号	业主账号	身份证号	手机号码	车位数
<input checked="" type="checkbox"/>	侯刚	男	9-21-31	214021862612267	60454119141122941x	18728009263	0
<input type="checkbox"/>	侯刚	女	5-01-32	7277232476950722738	59061318900210416X	17550891381	0
<input type="checkbox"/>	钱艳	男	12-01-33	216063825748118	723664188203104158	13829448542	0
<input type="checkbox"/>	吴磊	女	2-21-34	486472777450725	57084319581212142X	13655479560	0
<input type="checkbox"/>	于娟	男	15-21-35	3540545457263873512	571585199611162730	18624857856	0
<input type="checkbox"/>	王刚	男	11-01-36	7891348179395180196	18291618501111387X	19198947322	0
<input type="checkbox"/>	王涛	男	6-01-37	892628462274369788	86847320071231687x	15758685746	0
<input type="checkbox"/>	金霞	女	8-21-38	3842461808212405526	57867518001020250X	17999562331	0
<input type="checkbox"/>	叶艳	男	11-21-39	7645305821319241113	431117180112300252	18813725727	0
<input type="checkbox"/>	廖伟	女	16-21-40	472456648546579	314523183810316067	16069627117	0

< 1 2 3 4 5 6 ... 16 >

上面的图片就是我们最终要实现的效果图，我们可以在效果图上面看到一个非常显示的点就间每一个功能模块基本上都对应对了数据表的四个操作，也就是增删改查

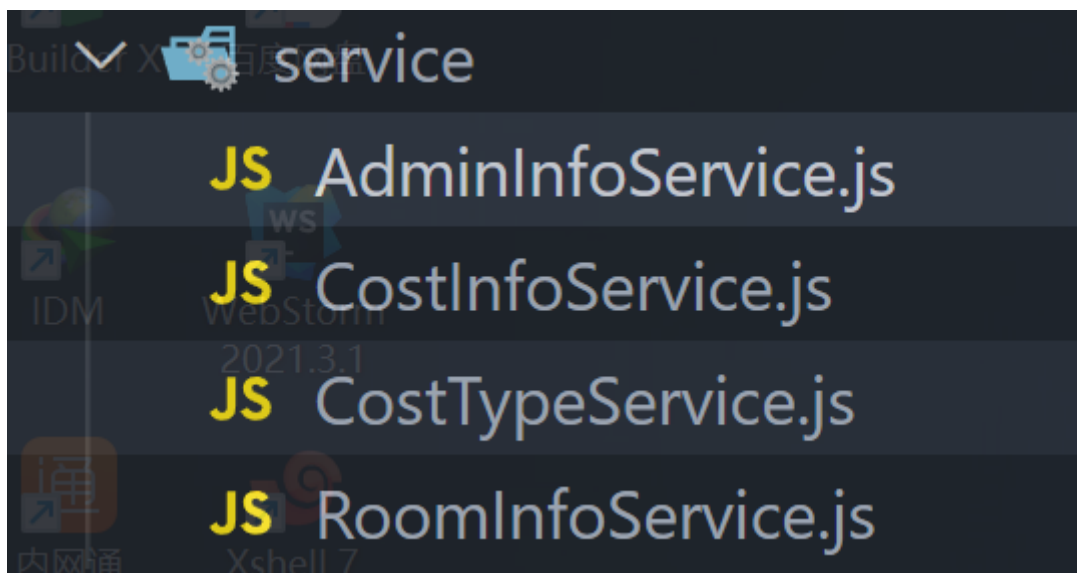
有了这些增删改查的操作以后，我们就可以对数据库里面的数据进行管理



数据库的所有操作最终都是为了给页面提供服务的，所以我们一般把这一层操作操作叫做Service服务器

## service文件夹的创建





我们有四个数据表，所以我们根据每个数据表创建了不同的Service

1. `RoomInfoService` 针对roominfo数据表的操作
2. `CostTypeService` 针对costtype数据表的操作
3. `CostInfoService` 针对costinfo数据表操作
4. `AdminInfoService` 针对adminInfo的操作

我们可以看到每一个Service现在的首字终都是大写，因为它是一个 `class` ,构造函数的首字母就是大写，这个好区分，也是一个约定俗成的规范

## 页面功能的分析

当我们把四个 `Service` 的文件创建好了以后，我们怎么样去书写里面的代码呢？这个时候说法要注意，我们要根据页面的功能去来判断怎么书写

首页 / 居民信息列表

### 居民信息列表

房主姓名  房间编号  手机号码

新增, 修改, 删除, 导出Excel的操作

	业主姓名	性别	房间编号	业主账号	身份证号	手机号码	车位数
<input checked="" type="checkbox"/>	侯刚	男	9-21-31	214021862612267	60454119141122941x	18728009263	0
<input type="checkbox"/>	侯刚	女	5-01-32	7277232476950722738	59061318900210416X	17550891381	0
<input type="checkbox"/>	钱艳	男	12-01-33	216063825748118	723664188203104158	13829448542	0
<input type="checkbox"/>	吴磊	女	2-21-34	48647277450725	57084319581212142X	13655479560	0
<input type="checkbox"/>	于娟	男	15-21-35	3549545457263873512	571585199611162730	18624857856	0
<input type="checkbox"/>	王刚	男	11-01-36	7891348179395180196	18291618501111387X	19198947322	0
<input type="checkbox"/>	王涛	男	6-01-37	8926284462274369788	86847320071231687x	15751685746	0
<input type="checkbox"/>	金霞	女	8-21-38	3842461808212405526	57867518001020250X	17999562331	0
<input type="checkbox"/>	叶艳	男	11-21-39	7645305821319241113	431117180112300252	18813725727	0
<input type="checkbox"/>	廖伟	女	16-21-40	472456648546579	314523183810316067	16069627117	0

分页查询 < 1 2 3 4 5 6 ... 16 >

```
1  /**
2   * 对数据表roominfo的操作
3   */
4  class RoomInfoService {
5      add(){
6  
```

```

7      }
8      deleteById(id){
9
10     }
11     update(){
12
13     }
14     getList(){
15
16     }
17 }
18 module.exports = RoomInfoService;

```

刚刚在图片里面所列举出来的四个操作，我们可以对应 `add()`、`deleteById()`、`update()`、`getList()`

现在我们去编写`add()`的方法去测试一下

```

1  /**
2   * 对数据表roominfo的操作
3   */
4
5  const DBUtils = require("../utils/DBUtils.js");
6
7  class RoomInfoService extends DBUtils {
8      async add(roomname, roomarea, ownername, onersex, IDCard, telephone, email,
9      roomstatus) {
10          let strSql = `INSERT INTO roominfo
11              (roomname, roomarea, ownername, onersex, IDCard, telephone, email,
12              roomstatus)
13              VALUES (?, ?, ?, ?, ?, ?, ?, ?);`;
14          let result = await this.executeSql(strSql, [roomname, roomarea,
15          ownername, onersex, IDCard, telephone, email, roomstatus]);
16          // 根据受影响的行数来看,它是否有新增成功
17          // if(result.affectedRows>0){
18          //     // 新增成功
19          //     return true;
20          // }
21          // else{
22          //     // 新增失败
23          //     return false;
24          // }
25          return result.affectedRows > 0;
26      }
27  }
28  module.exports = RoomInfoService;

```

在当前项目下面创建一个 `test` 的目录，然后去新建一个 `testRoomInfoService.js` 完成测试

```

1  /**
2   * 因为还没有学到单元测试,我们现在先在这里进行测试
3   */
4  const RoomInfoService = require("../service/RoomInfoService.js");
5  //请注意,这里有一个命名规范的问题,小写的变量,大写的对象
6  const roominfoService = new RoomInfoService()
7

```

```

8   const testAdd = async () => {
9       let result = await roominfoService.add("1-1-102", 90, "马淑圆", "女",
        "420965199910102323", "17627384758", "222@abc.com", 0);
10      if (result) {
11          console.log("新增成功");
12      }
13      else{
14          console.log("新增失败");
15      }
16  }
17
18  testAdd();

```

这个时候是新增成功

**代码分析：**当我们在调用roomInfoService.add()的就去的时候，我们看到了很多参数传递进去了，这样非常不好，因为数据要集中管理，这样太散了

## 改造一下

```

1   /**
2    * 对数据表roominfo的操作
3    */
4
5   const DBUtils = require("../utils/DBUtils.js");
6
7   class RoomInfoService extends DBUtils {
8       async add({ roomname, roomarea, ownername, onersex, IDCard, telephone, email,
        roomstatus }) {
9           let strSql = `INSERT INTO roominfo
10              (roomname, roomarea, ownername, onersex, IDCard, telephone, email,
        roomstatus)
11              VALUES (?, ?, ?, ?, ?, ?, ?, ?);`;
12           let result = await this.executeSql(strSql, [roomname, roomarea,
        ownername, onersex, IDCard, telephone, email, roomstatus]);
13           return result.affectedRows > 0;
14       }
15   }
16
17
18   module.exports = RoomInfoService;

```

上面唯一变化的一个点就是把参数使用对象解构的方式来接收，这做的好处就是在传递参数的时候可以使用对象进行

```

1   /**
2    * 因为还没有学到单元测试，我们现在先在这里进行测试
3    */
4
5   const RoomInfoService = require("../service/RoomInfoService.js");
6
7   //请注意，这里有一个命名规范的问题，小写的变量，大写的对象
8   const roominfoService = new RoomInfoService()
9
10  const testAdd = async () => {
11      // 设置一个要新增的对象

```

```

12     let roomInfo = {
13         roomname: "1-1-102",
14         roomarea: 100,
15         ownername: "肖中",
16         onersex: "男",
17         IDCard: "420999199909081716",
18         telephone: "18726374678",
19         email: "333@abc.com",
20         roomstatus: 0
21     }
22     //这里在传递的时候，就已经是一个对象了
23     let result = await roominfoService.add(roomInfo);
24     if (result) {
25         console.log("新增成功");
26     }
27     else{
28         console.log("新增失败");
29     }
30 }
31 testAdd();

```

上面的代码 我们是完成了RoomInfoService，这个Service的操作主要是针对数据表当中roominfo表  
如果我们现在要对其它的数据表进行操作，这个时候的思路也是一样的

```

1     /**
2      * 对数据表进行costtype进行操作
3      */
4     const DBUtils = require("../utils/DBUtils.js");
5
6     class CostTypeService extends DBUtils {
7         /**
8          * 新增
9          * @param {{ costname, price, description }} param 数据库的参数
10          * @return {Promise<Boolean>} true代表新增成功,false代表新增失败
11          */
12         async add({ costname, price, description }) {
13             let strSql = `insert into costtype (costname, price, description)
14                 values (?, ?, ?)`;
15             let result = await this.executeSql(strSql, [costname, price,
16                 description])
17             return result.affectedRows > 0;
18         }
19         /**
20          * 根据id删除一条记录
21          * @param {number} id 费用类别的id,主键
22          * @return {Promise<Boolean>} true代表删除成功,false代表删除失败
23          */
24         async deleteById(id) {
25             let strSql = `delete from costtype where id = ?`;
26             let result = await this.executeSql(strSql, [id]);
27             return result.affectedRows > 0;
28         }
29     }
30     module.exports = CostTypeService;

```

当我们将2个Service都写完了以后，我们再回过头来看一下，总结一下它们的特点

1. 每一个Service里面应该都有对应的增删改查的操作

- `add()` 对应新增
- `deleteById()` 对应删除
- `update()` 对应后期的修改
- `getList()` 对应的就是后期的查询

2. 我们去对比的时候，还发现一个问题，在删除操作的时候，它们的SQL语句非常相似，除了表名不一样的以外，其它的地方都要同

```
1 delete from costtype where id = ?; -- 删除费用
2 delete from roominfo where id = ?; -- 删除房间
```

3. 我们现在的Service对应的是每一个数据表，如果后期DBA要是把数据库的表名更换了，那么我们的所有的SQL语句都要更改表名，这样就很麻烦了

为了解决这样的问题，我们首先考虑的是把表名当成一个参数固定下来

```
1 const DBUtils = require("../utils/DBUtils.js");
2
3 class CostTypeService extends DBUtils {
4     // 在new的时候自动执行
5     constructor(){
6         super();
7         this.currentTableName = "costtype";
8     }
9     /**
10     * 新增
11     * @param {{ costname, price, description }} param 数据库的参数
12     * @return {Promise<Boolean>} true代表新增成功, false代表新增失败
13     */
14     async add({ costname, price, description }) {
15         let strSql = `insert into ${this.currentTableName} (costname, price,
16         description)
17         values (?, ?, ?)`;
18         let result = await this.executeSql(strSql, [costname, price,
19         description])
20         return result.affectedRows > 0;
21     }
22     /**
23     * 根据id删除一条记录
24     * @param {number} id 费用类别的id, 主键
25     * @return {Promise<Boolean>} true代表删除成功, false代表删除失败
26     */
27     async deleteById(id) {
28         let strSql = `delete from ${this.currentTableName} where id = ?`;
29         let result = await this.executeSql(strSql, [id]);
30         return result.affectedRows > 0;
31     }
32     //省略代码
33 }
34 module.exports = CostTypeService;
```

在上面的代码里面，我们可以看到在构造函数里面添加了 `this.currentTableName = "costtype";`

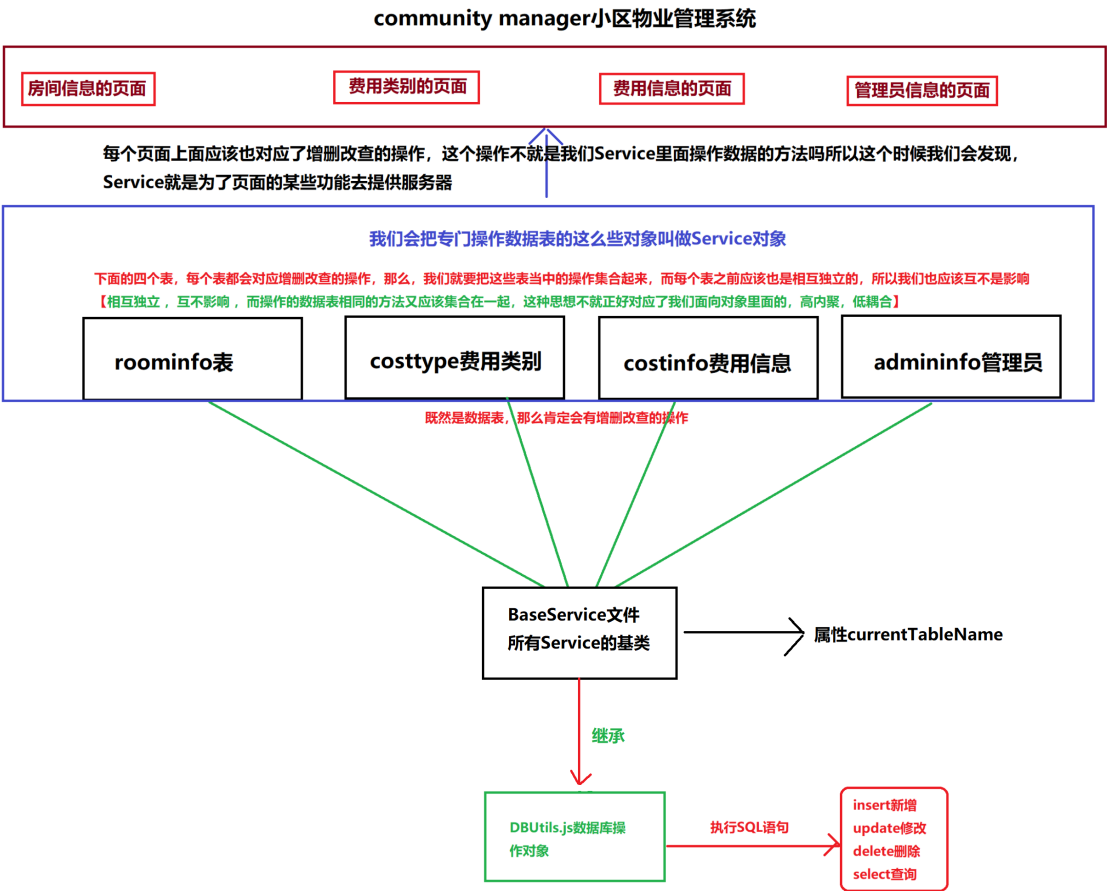
这一段代码就用来设置当前的表名为 `currentTableName`，后期只要调用这一个变量名就知道当前表的名称，如果后期数据库里面的表名发生了更改，我们也只需要改这里就可以了

```
class RoomInfoService extends DBUtils {
  // 在new的时候会自动执行
  constructor(){
    super();
    this.currentTableName = "roominfo";
  }
  async add({ roomname, roomarea, ownername, onersex, IDCard, telephone, email, roomstat
    let strSql = `INSERT INTO ${this.currentTableName}
      (roomname, roomarea, ownername, onersex, IDCard, telephone, email, roomstatus)
      VALUES (?, ?, ?, ?, ?, ?, ?, ?)`;
    let result = await this.executeSql(strSql, [roomname, roomarea, ownername, onersex
    return result.affectedRows > 0;
  }
  async deleteById(id) {
    let strSql = `delete from ${this.currentTableName} where id = ? `;
    let result = await this.executeSql(strSql, [id]);
    return result.affectedRows > 0 ? true : false;
  }
  update() {...
}
```

同时，当我们完成一个表以后，其它的表也应该这样操作，这怎么办呢？

BaseService的使用

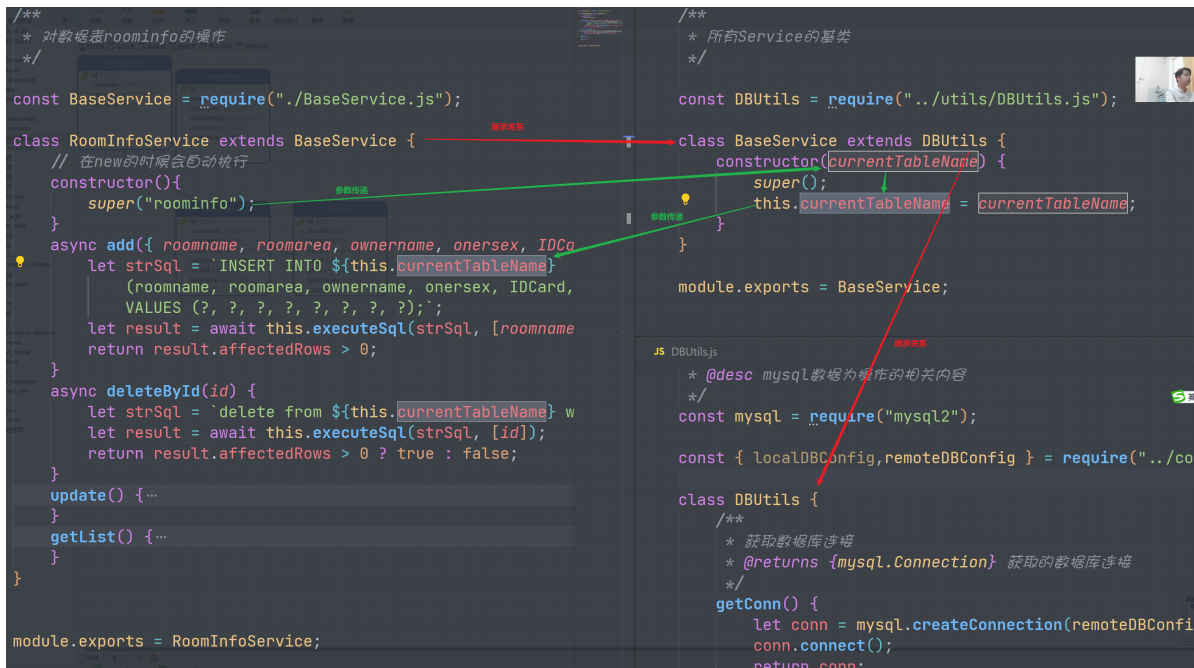
针对上面的问题，我们不得不考虑一个现象，我们发现所有的Service里面都有一个共享的属性叫 `currentTableName`，那么对于共同的东西，我们应该怎么办呢？



```

1  /**
2   * 所有Service的基类
3   */
4  const DBUtils = require("../utils/DBUtils.js");
5
6  class BaseService extends DBUtils {
7      constructor(currentTableName) {
8          super();
9          this.currentTableName = currentTableName;
10     }
11 }
12
13 module.exports = BaseService;

```



## BaseService的重构

BaseService的功能大家也就都知道了，它可以把一些公共的放在这里，然后让所有的Service都继承自这个对象，这样如果有一些共同的方法，我们放在这里以后，其它所有的Service都可以使用

```

-- deleteById(1)
-- 删除了房间
delete from ${this.currentTableName} where id = 1;

-- 删除了费用类别
delete from ${this.currentTableName} where id = 1;

-- 删除了管理员
delete from ${this.currentTableName} where id = 1;

-- 删除了费用信息
delete from ${this.currentTableName} where id = 1;

```

经过分析以后，我们可以看到，其实删除的功能是可以进行重构的

```

1  /**
2   * 所有Service的基类

```

```

3    */
4
5    const DBUtils = require("../utils/DBUtils.js");
6
7    class BaseService extends DBUtils {
8        constructor(currentTableName) {
9            super();
10           this.currentTableName = currentTableName;
11        }
12        /**
13         * 根据id删除某一项数据
14         * @param {number} id 要删除项的id主键
15         * @returns {Promise<boolean>} true代表成功, false代表失败
16         */
17        async deleteById(id) {
18            let strSql = `delete from ${this.currentTableName} where id = ?`;
19            let result = await this.executeSql(strSql, [id]);
20            return result.affectedRows > 0;
21        }
22        /**
23         * 获取所有记录
24         * @returns {Promise<Array>}
25         */
26        async getAllList() {
27            let strSql = `select * from ${this.currentTableName} ;`;
28            let result = await this.executeSql(strSql);
29            return result;
30        }
31    }
32
33    module.exports = BaseService;

```

## Service里面的查询操作

居民信息列表

房主姓名 输入姓名 房间编号 输入房间号 手机号码 输入手机号码 查询

新增 编辑 删除 导出Excel

如果我想把getList对应成查询操作, 怎么办?

业主姓名	性别	房间编号	业主账号	身份证号
侯刚	男	9-21-31	214021862612267	604581111122841x
侯刚	女	5-01-32	7277232476950722738	5906318900210416X
钱艳	男	12-01-33	216063825748118	723641111104158x
吴磊	女	2-21-34	486472777450725	520841111121142X
于娟	男	15-21-35	3549545457263873512	57154111111112730x
王刚	男	11-01-36	7891348179395180196	18294111111111087x
王涛	男	6-01-37	8926284462274369788	8684411111231687x
金霞	女	8-21-38	384246180821319241113	5786318001020250X
叶艳	男	11-21-39	7645305821319241113	43114111112300252x
廖伟	女	16-21-40	472456648546579	3145411381010216067

```

JS RoomInfoService.js
service > JS RoomInfoService.js > RoomInfoService > getList
class RoomInfoService extends BaseService {
    // 在new的时候会自动执行
    constructor(){
        super("roominfo");
    }
    async add({ roomname, roomarea, roomtype, roomstatus, roomprice }) {
        let strSql = `INSERT INTO roominfo (roomname, roomarea, roomtype, roomstatus, roomprice) VALUES (?, ?, ?, ?, ?)`;
        let result = await this.executeSql(strSql, [roomname, roomarea, roomtype, roomstatus, roomprice]);
        return result.affectedRows;
    }
    update() { ... }
    getList() { ... }
}
module.exports = RoomInfoService;

```

当我们重构以后, 相同的功能可以放在BaseService里面, 不同的功能, 我们就要使用各自的Service



现在如果我们想完成页面上面的查询操作，应该怎么办呢？这个查询操作对应的就是 `getList()` 这个方法，怎么办呢？

1. 查询的时候是有三个条件的，房主姓名，房间编号，以及手机号这三个，怎么样动态拼接这三个条件
2. 这里是否会有页码的产生？【这个问题在这里先不解决，我们在项目当中解决】

```
1  /**
2   * 对数据表roominfo的操作
3   */
4  const BaseService = require("../BaseService.js");
5  class RoomInfoService extends BaseService {
6      // 在new的时候会自动执行
7      constructor() {
8          super("roominfo");
9      }
10     //省略部分代码
11     /**
12      * 查询房间信息
13      * @param {{ownername, roomname, telephone}} 查询参数
14      * @returns {Promise<Array>} 返回数据库的查询结果
15      */
16     async getList({ ownername, roomname, telephone }) {
17         //这是最基本的一个SQL语句
18         let strSql = `select * from ${this.currentTableName} where 1 = 1 `;
19         //查询条件拼接的SQL语句
20         let strWhere = ``;
21         // 定义一个参数数组
22         let ps = [];
23         if (ownername) {
24             strWhere += ` and ownername like ? `;
25             ps.push(`%${ownername}%`);
26         }
27         if (roomname) {
28             strWhere += ` and roomname like ? `;
29             ps.push(`%${roomname}%`);
30         }
31         if (telephone) {
32             strWhere += ` and telephone = ? `;
33             ps.push(telephone);
34         }
35         //最后将2条SQL语句合在一起
36         strSql += strWhere;
37         let result = await this.executeSql(strSql, ps);
38         return result;
39     }
40 }
41 module.exports = RoomInfoService;
```

上面的 `getList()` 就是我们完成的功能，我们使用了三个参数，同时对这三个参数进行了判断拼接，同时得到了 `strWhere` 以及 `ps` 2个变量，这样做非常好

万一要是10个参数要判断拼接呢？同时我们还要考虑一个问题，应该是所有的数据表都会有查询，所以也就是所有的Service都应该会有这个操作

我们现在迫切希望将这个简化的操作放在父级里面，只要在父级里面有了这个方法，那所有的子级都可以使用这个方法

#### DBUtils.js

```
1  /**
2   * @author 杨标
3   * @Date 2022-10-17
4   * @desc mysql数据为操作的相关内容
5   */
6  const mysql = require("mysql2");
7  const { localDBConfig, remoteDBConfig } = require("../config/DBConfig.js");
8
9  class DBUtils {
10     //省略部分代码
11
12     /**
13     * 初始化参数的方法
14     * @returns {{strWhere:string,ps:Array}}
15     */
16     paramsInit() {
17         let obj = {
18             strWhere: "",
19             ps: [],
20             /**
21             * 精确查询
22             * @param {string|number|boolean} value
23             * @param {string} name
24             * @returns {obj}
25             */
26             equal(value, name) {
27                 if (value) {
28                     this.strWhere += ` and ${name} = ? `;
29                     this.ps.push(value);
30                 }
31                 return this;
32             },
33             /**
34             * 模糊查询
35             * @param {string|number|any} value
36             * @param {string} name
37             * @returns {obj}
38             */
39             like(value, name) {
40                 if (value) {
41                     this.strWhere += ` and ${name} like ? `;
42                     this.ps.push(`%${value}%`);
43                 }
44                 return this;
45             }
46         }
47         return obj;
48     }
49 }
50
51 module.exports = DBUtils;
```

现在，我们在 `DBUtils.js` 里面添加了一个就去，这个方法就是用于拼接SQL语句的

同时，我们在 `RoomInfoService.js` 里在，也调用了这个方法去实现

```
1  async getList2({ ownername, roomname, telephone }) {
2      let strSql = `select * from ${this.currentTableName} where 1 = 1 `;
3      let { strWhere, ps } = this.paramsInit()
4          .like(ownername, "ownername")
5          .like(roomname, "roomname")
6          .equal(telephone, "telephone");
7
8      //最后将2条SQL语句合在一起
9      strSql += strWhere;
10     let result = await this.executeSql(strSql, ps);
11     return result;
12 }
```

## 工厂模式

### 抽象工厂模式