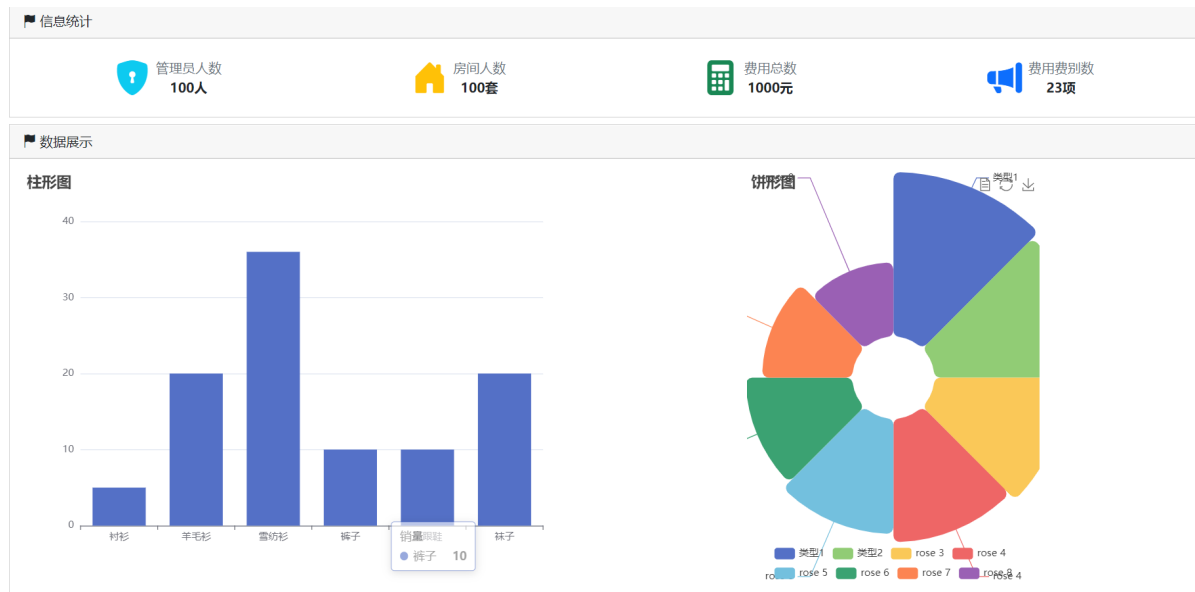


# 数据可视化echarts操作

## 效果图



## 一、界面完成

关于界面布局，我们就不再讲了，现在先讲一下Service

## 二、DataViewService的完成

因为这个功能不涉及到任何一个具体的具，所以我们在 `service` 的文件夹下面新建了一个Service叫 `DataViewService` 代码如下

```
1  /**
2   * 用于数据展示的Service操作，不涉及到任何的具体的表
3   */
4  const BaseService = require("../BaseService");
5
6  class DataViewService extends BaseService {
7    constructor() {
8      super();
9      //因为不涉及到任何具体的表，所以我不用传参给父级，这样我们也不使用currentTableName这个属性
10   }
11
12   /**
13    * 获取计算的总数
14    * @returns {Promise<Object>} 返回查询的结果的对象
15    */
16   async getCalcData() {
17     let strSql = `select
18       max(if(adminCount='adminCount',totalCount,0)) 'adminCount',
```

```

19         max(if(adminCount='roomCount',totalCount,0)) 'roomCount',
        max(if(adminCount='moneyCount',totalCount,0)) 'moneyCount',
        max(if(adminCount='costTypeCount',totalCount,0)) 'costTypeCount'
        from(select 'adminCount',count(*) 'totalCount' from ${this.tableMap.admininfo}
20             union allselect 'roomCount',count(*) 'totalCount' from
        ${this.tableMap.roominfo}
21             union all      select 'moneyCount',sum(totalmoney) 'totalMoney' from
        ${this.tableMap.costinfo}
22             union all      select 'costTypeCount', count(*) 'totalCount' from
        ${this.tableMap.costtype}) a`;
23         let result = await this.executeSql(strSql);
24         return result[0];
25     }
26 }
27
28
29 module.exports = DataViewService;

```

在上面的代码里在，我们可以看到使用了 `union all` 来进行结果集并联，也使用了行转列的操作，这个sql语句执行的结果如所示

adminCount	roomCount	moneyCount	costTypeCount
32.00	73.00	160214.79	44.00

当我们将Service完成了以后，我们就要再进入路由操作

### 三、在工厂里面生产这个Service

```

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     static createDataViewService(){
28         const DataViewService = require("../services/DataViewService");
29         return new DataViewService();
30     }
31 }
32
33 module.exports = ServiceFactory;

```

## 四、完成路由dataViewRouter.js

```

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

```

当我们创建好路由文件以后，一定要在 **app.js** 里面链接我们的路由文件，这样才会有一个一级路径

```

1  app.use("/dataView", require("../routes/dataViewRouter"));

```

当我们把所有的工作都准备好了以后，我们可以在 **dataViewRouter.js** 里面来处理我们的请求了

```

1  /**
2   * @author 杨标
3   * @description dataView的路由模块
4   */
5  const express = require("express");
6  const router = express.Router();
7  const ServiceFactory = require("../factory/ServiceFactory");
8  const ResultJson = require("../model/ResultJson");
9
10 router.get("/getCalcData", async (req, resp) => {
11     let result = await ServiceFactory.createDataViewService().getCalcData();
12     let resultJson = new ResultJson(Boolean(result), result ? "获取数据成功" : "获取数据失败", result);
13     resp.json(resultJson);
14 });
15
16
17 module.exports = router;

```

现在的前端页面只在请求这一个地址就可以获取到数据了

```

1  <script>
2      $(function () {
3          async function getCalcData() {
4              try {
5                  let result = await
request.get(`${baseUrl}/dataView/getCalcData`);
6                  $("#adminCount").text(result.data.adminCount);
7                  $("#costTypeCount").text(result.data.costTypeCount);
8                  $("#moneyCount").text(result.data.moneyCount);

```

```
9         $("#roomCount").text(result.data.roomCount);
10     } catch (error) {
11         console.log(error)
12     }
13 }
14
15 getCalcData();
16 })
17 </script>
```

## 五、百度图形图表的使用

### 1. 图形图表的介绍

图形图表用一个前端专业的术语来说叫数据可视化，它是将一系列的数据转换成可见化的展示操作

目前能够实现数据可视化的框架有很多，但使用的最得最多的就是以下几种

- **hicharts** 这个是国外使用得比较多的一个框架，这个框架功能强大，非常好用，但是收费
- **echarts** 这是百度为了我们推出的一个图形图表框架，完全免费，并且已经将这个框架捐献给了 **apache** 开源组织。它实现了 **hicharts** 中99%的功能
- **datav** 阿里巴大数据展示库，但是这个也是个人免费，企业需要购买授权

Apache ECharts

综合对比以后，其实现在的企业多数使用的都是百度的echarts来完成数据展示

### 2. 百度图表的安装与导入

#### 下载 - Apache ECharts

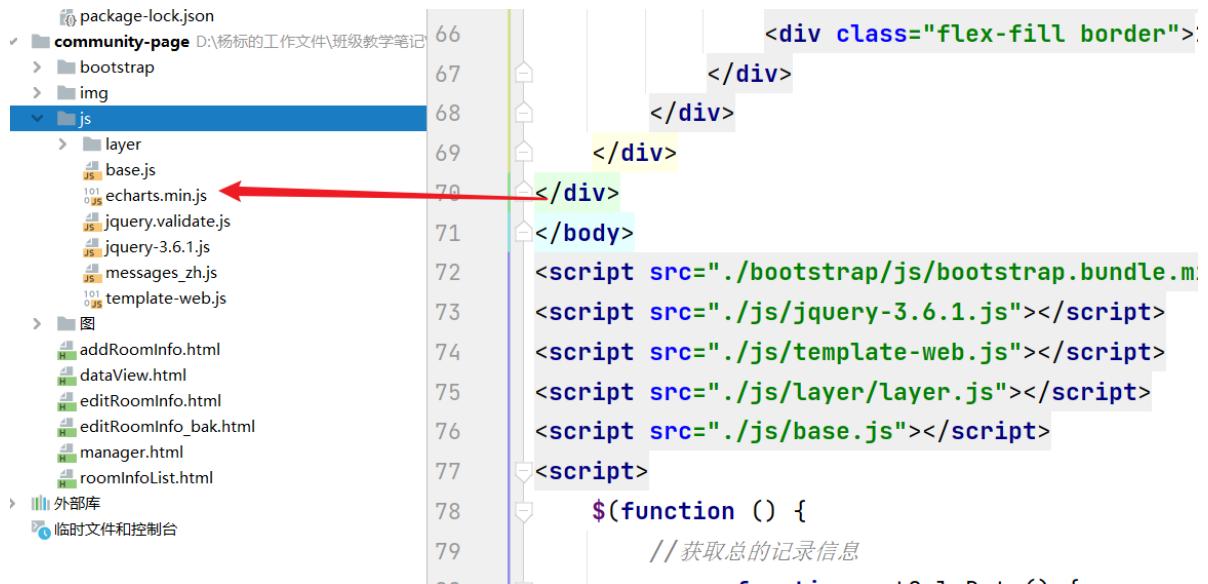
上面就是百度图表的下载地址，我们可以通过 **github** 来进行下载，也可以通过 **npm** 仓库来进行下载

```
1 $ npm install echarts
```

随便找一下空的目录执行上面的命令，这个时候 **ecahrts** 就会从服务器下载下来，我们从 **node\_modules** 里面找到一个文件 **echarts.min.js** 这个文件，它就是我们所需要用到的文件

名称	修改日期	类型	大小
extension	2022-10-25 11:07	文件夹	
echarts.common.js	2022-10-25 11:06	JavaScript 源文件	2,136 KB
echarts.common.js.map	2022-10-25 11:07	MAP 文件	4,523 KB
echarts.common.min.js	2022-10-25 11:06	JavaScript 源文件	640 KB
echarts.esm.js	2022-10-25 11:06	JavaScript 源文件	2,923 KB
echarts.esm.js.map	2022-10-25 11:07	MAP 文件	7,024 KB
echarts.esm.min.js	2022-10-25 11:06	JavaScript 源文件	998 KB
echarts.js	2022-10-25 11:06	JavaScript 源文件	3,246 KB
echarts.js.map	2022-10-25 11:07	MAP 文件	7,025 KB
echarts.min.js	2022-10-25 11:06	JavaScript 源文件	997 KB
echarts.simple.js	2022-10-25 11:06	JavaScript 源文件	1,544 KB
echarts.simple.js.map	2022-10-25 11:07	MAP 文件	3,274 KB
echarts.simple.min.js	2022-10-25 11:06	JavaScript 源文件	454 KB

把这个文件复制到自己的 `js` 的目录下面来



同时，我们要把这个文件导入到所需要的页面，这里我们导入的是 `dataView.html` 这个页面

```
1 <script src="./js/echarts.min.js"></script>
```

### 3. 百度echarts的使用

```
1 function initChart1(){
2     // 基于准备好的dom，初始化echarts实例 将id="chart1"的这个区域初始化为图表的展示区域
3     var myChart = echarts.init(document.getElementById('chart1'));
4     // 指定图表的配置项和数据
5     var option = {
6         title: {
7             text: '费用类别'
8         },
9         tooltip: {},
10        legend: {
11            data: ['费用总数']
12        },
13        xAxis: {
14            data: ['电费', '水费', '保护费', '清洁费', '煤气费', '物业费']
15        },
16        yAxis: {},
17        series: [
18            {
19                name: '费用总数',
20                type: 'bar',
21                data: [50, 20, 36, 10, 10, 20]
22            }
23        ]
24    };
25    // 使用刚指定的配置项和数据显示图表。 这个方法就是用于设置，怎么样去展示图形图表
26    myChart.setOption(option);
27 }
28 initChart1();
```

当我们把所有的图表都设置完成以后，我们就想着办法去从后台服务器获取真实的数据，这里我们以类用类别来汇总，进行费总的统计信息

## 4.从后台获取数据渲染图表

### DataViewService.js

```
1  /**
2   * 用于数据展示的Service操作，不涉及到任何的具体的表
3   */
4  const BaseService = require("../BaseService");
5
6  class DataViewService extends BaseService {
7      constructor() {
8          super();
9          //因为不涉及到任何具体的表，所以我不用传参给父级，这样我们也不使用currentTableName这个属性
10     }
11
12     /**
13      * 获取计算的总数
14      * @returns {Promise<Object>} 返回查询的结果的对象
15      */
16     async getCalcData() {
17         let strSql = `select
18             max(if(adminCount='adminCount',totalCount,0)) 'adminCount',
19             max(if(adminCount='roomCount',totalCount,0)) 'roomCount',
20             max(if(adminCount='moneyCount',totalCount,0)) 'moneyCount',
21             max(if(adminCount='costTypeCount',totalCount,0)) 'costTypeCount'
22             from(select 'adminCount',count(*) 'totalCount' from ${this.tableMap.admininfo}
23                 union allselect 'roomCount',count(*) 'totalCount' from
24                 ${this.tableMap.roominfo}
25                 union all      select 'moneyCount',sum(totalmoney) 'totalMoney' from
26                 ${this.tableMap.costinfo}
27                 union all      select 'costTypeCount', count(*) 'totalCount' from
28                 ${this.tableMap.costtype}) a`;
29         let result = await this.executeSql(strSql);
30         return result[0];
31     }
32
33     /**
34      * 获取费用类别的总金额
35      * @return {Promise<Array>}
36      */
37     async getCostTypeTotalMoney() {
38         let strSql = `select a.costname,sum(a.totalmoney) 'totalMoney' from
39             (select a.*,b.costname from ${this.tableMap.costinfo} a
40             inner join
41             ${this.tableMap.costtype} b on a.costid = b.id) a
42             group by a.costid`;
43         let result = await this.executeSql(strSql);
44         return result;
45     }
46 }
47
48 module.exports = DataViewService;
```

### dataViewRouter.js路由文件

```

1  router.get("/getCostTypeTotalMoney", async(req, resp)=>{
2      let result = await
    ServiceFactory.createDataViewService().getCostTypeTotalMoney();
3      let resultJson = new ResultJson(true, "获取数据成功", result);
4      resp.json(resultJson);
5  })

```

当所有的后台功能完成了以后，我们就可以在页面上面请求这些接口了

```

1  //初始化第一个图形图表
2  async function initChart() {
3      // 第一步： 基于准备好的dom，初始化echarts实例 将id="chart1"的这个区域初始化为图表的展
    示区域
4      let myChart1 = echarts.init(document.getElementById('chart1'));
5      // 指定图表的配置项和数据
6      let option1 = {
7          title: {
8              text: '费用类别'
9          },
10         tooltip: {},
11         legend: {
12             data: ['费用总数']
13         },
14         xAxis: {
15             // data: ['电费', '水费', '保护费', '清洁费', '煤气费', '物业费']
16             data: []
17         },
18         yAxis: {},
19         series: [
20             {
21                 name: '费用总数',
22                 type: 'bar',
23                 // data: [50, 20, 36, 10, 10, 20]
24                 data: [],
25             }
26         ],
27         color: "#c23531"
28     };
29
30     //-----第二个图表-----
31     //第一步：初始化
32     let myChart2 = echarts.init(document.getElementById('chart2'));
33     //第二步：配置option
34     let option2 = {
35         title: {
36             text: '费用类别',
37             subtext: '各费用类别的总数',
38             left: 'center'
39         },
40         tooltip: {
41             trigger: 'item'
42         },
43         legend: {
44             orient: 'vertical',
45             left: 'right'
46         },

```

```

47     series: [
48         {
49             name: 'Access From',
50             type: 'pie',
51             radius: '50%',
52             data: [
53                 // {value: 1048, name: 'Search Engine'},
54                 // {value: 735, name: 'Direct'},
55                 // {value: 580, name: 'Email'},
56                 // {value: 484, name: 'Union Ads'},
57                 // {value: 300, name: 'Video Ads'}
58             ],
59             emphasis: {
60                 itemStyle: {
61                     shadowBlur: 20,
62                     shadowOffsetX: 20,
63                     shadowColor: 'rgba(0, 0, 0, 0.5)'
64                 }
65             }
66         }
67     ]
68 };
69
70
71 //第三步: 请求后台的数据
72 try {
73     let result = await
request.get(`${baseUrl}/dataView/getCostTypeTotalMoney`);
74     result.data.forEach(item => {
75         option1.xAxis.data.push(item.costname);
76         option1.series[0].data.push(item.totalMoney);
77
78         //处理第二个表格的数据
79         option2.series[0].data.push({
80             value:item.totalMoney,
81             name:item.costname
82         })
83     });
84     // 使用刚指定的配置项和数据显示图表。 这个方法就是用于设置, 怎么样去展示图形图表
85     // 第四步:设置这一个option
86     myChart1.setOption(option1);
87     myChart2.setOption(option2);
88
89 } catch (error) {
90     console.log(result);
91 }
92 }
93
94 initChart();

```





```
23 module.exports = AdminInfoService;
```

#### adminInfoRouter.js

```
1  /**
2   * @author 杨标
3   * @description adminInfo路由模块
4   */
5  const express = require("express");
6  const router = express.Router();
7  const ServiceFactory = require("../factory/ServiceFactory");
8  const ResultJson = require("../model/ResultJson");
9
10 router.post("/addAdminInfo", async (req, resp) => {
11     //当我们接收到前台传递过来的参数以的,我们就要插入到数据库
12     let result = await ServiceFactory.createAdminInfoService().add(req.body);
13     let resultJson = new ResultJson(result, result ? "新增成功" : "新增失败");
14     resp.json(resultJson);
15 });
16
17 module.exports = router;
```

经过上面的操作以后,我们后台代码已经完在了,现在的页面上面只要提交数据就可以保存到后台的数据库里面

#### addAdminInfo.html

```
1  <!DOCTYPE html>
2  <html lang="zh">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta http-equiv="X-UA-Compatible" content="IE=edge">
7      <meta name="viewport" content="width=device-width, initial-scale=1.0">
8      <title>新增管理员</title>
9      <link rel="stylesheet" href="../bootstrap/css/bootstrap.min.css">
10     <link rel="stylesheet" href="../bootstrap/font/bootstrap-icons.css">
11     <link rel="stylesheet" href="../js/layer/theme/default/layer.css">
12 </head>
13
14 <body>
15     <div class="container-fluid">
16         <ul class="breadcrumb my-2">
17             <li class="breadcrumb-item"><a href="dataView.html">首页</a></li>
18             <li class="breadcrumb-item active">新增管理员</li>
19         </ul>
20     </div>
21     <div class="container">
22         <div class="h4 text-center text-primary border-bottom py-3">新增管理员</div>
23         <form id="addAdminInfoForm">
24             <div class="form-floating mt-3 mb-3">
25                 <input type="text" class="form-control"
26                     placeholder="管理员姓名" id="adminname" name="adminname"
27                     data-rule-required="true" data-msg-required="管理员姓名不能为
28                     空">
29                 <label for="adminname">管理员姓名</label>
30             </div>
```

```

30     <div class="form-floating mt-3 mb-3">
31         <input type="text" class="form-control"
32             placeholder="管理员密码" id="adminpwd" name="adminpwd"
33             data-rule-required="true" data-msg-required="管理员密码不能为
空">
34         <label for="adminpwd">管理员密码</label>
35     </div>
36     <div class="form-floating mt-3 mb-3">
37         <input type="text" class="form-control"
38             placeholder="请输入确认密码" id="confirmPwd" name="confirmPwd"
39             data-rule-required="true" data-msg-required="确认密码不能为空"
40             data-rule-equalTo="#adminpwd" data-msg-equalTo="两次密码必须相
同">
41         <label for="confirmPwd">确认密码</label>
42     </div>
43     <div class="form-floating mt-3 mb-3">
44         <input type="text" class="form-control"
45             placeholder="管理员邮箱" id="adminemail" name="adminemail"
46             data-rule-required="true" data-msg-required="管理员邮箱不能为空"
47             data-rule-regexp="\w+([-+.]\\w+)*@\\w+([-.]\\w+)*\\.\\w+
([-.]\\w+)*" data-msg-regexp="请输入正确的邮箱格式">
48         <label for="adminemail">管理员邮箱</label>
49     </div>
50     <div class="form-floating mt-3 mb-3">
51         <input type="text" class="form-control"
52             placeholder="请输入手机号码" id="admintel" name="admintel"
53             data-rule-required="true" data-msg-required="手机号不能为空"
54             data-rule-regexp="^(0|86|17951)?(13[0-
9]|15[012356789]|166|17[3678]|18[0-9]|14[57])[0-9]{8}$"
55             data-msg-regexp="请输入正确的手机号">
56         <label for="admintel">手机号码</label>
57     </div>
58     <div class="form-floating mt-3 mb-3">
59         <select class="form-select" name="adminstatus" id="adminstatus">
60             <option value="0">正常</option>
61             <option value="1">禁用</option>
62         </select>
63         <label for="adminstatus">管理员状态</label>
64     </div>
65     <div class="d-flex">
66         <button type="button" class="btn btn-primary" id="btn-save">
67             <i class="bi bi-file-pdf"></i>
68             保存
69         </button>
70         <button type="button" class="btn btn-warning mx-3">
71             <i class="bi bi-arrow-90deg-down"></i>
72             重置
73         </button>
74     </div>
75 </form>
76 </div>
77 </body>
78 <script src="./bootstrap/js/bootstrap.bundle.min.js"></script>
79 <script src="./js/jquery-3.6.1.js"></script>

```

```

80 <script src="./js/template-web.js"></script>
81 <script src="./js/layer/layer.js"></script>
82 <script src="./js/jquery.validate.js"></script>
83 <script src="./js/messages_zh.js"></script>
84 <script src="./js/base.js"></script>
85 <script>
86     $(function () {
87         $.validator.addMethod("regexp", function (value, element, params) {
88             var reg = new RegExp(params);
89             return reg.test(value);
90         });
91         //表单验证
92         var addAdminInfoFormResult = $("#addAdminInfoForm").validate({
93             errorPlacement: function (error, element) {
94                 // error代表的就是这个错误提示的消息
95                 // element代表你当前正在验证的这个元素
96                 element.parent().after(error);
97             },
98             errorClass: "text-danger"
99         });
100
101         $("#btn-save").click(async function () {
102             //第一步:做表单验证
103             if (addAdminInfoFormResult.form()) {
104                 //验证通过,我们就将数据通过ajax发送到后台
105                 try {
106                     let result = await
107 request.post(`${baseUrl}/adminInfo/addAdminInfo`, {
108                     adminname: $("#adminname").val(),
109                     adminpwd: $("#adminpwd").val(),
110                     adminemail: $("#adminemail").val(),
111                     admintel: $("#admintel").val(),
112                     adminstatus: $("#adminstatus").val()
113                 });
114                 layer.alert("新增成功");
115             } catch (error) {
116                 layer.alert("服务器错误");
117                 console.log(error);
118             }
119         } else {
120             layer.alert("请检测你的输入信息...");
121         }
122     })
123 </script>
124
125 </html>

```

开始事务 文本 筛选 排序 导入 导出

id	adminnam	adminpwd	adminemai	admintel	adminstatus	isDel
53	杨标	123456	lovesnsfi@	18627193876	0	0

这个时候的数据库已经可以存储我们的数据了，但是仍然要注意，这个时候的密码是明文存储的，很危险，我们要对密码 `adminpwd` 进行加密


## 七、md5加密


为了解决密码的存储问题，我们要对密码这一个字段进行加密，加密的方式们采用 md5，在 nodejs 的平台下面，有一个第三方模块可以完成这个操作

md5 


2.3.0 • Public • Published 2 years ago

 Readme

 Explore BETA

 3 Dependencies

 5,864 Dependents

 7 Versions

### MD5

build passing downloads 380M

a JavaScript function for hashing messages with MD5.

node-md5 is being sponsored by the following tool; please help to support us by taking a look and signing up to a free trial

 GitAds

### Installation

You can use this package on the server side as well as the client side.

Node.js:

```
npm install md5
```

Install

```
> npm i md5
```

Repository

 github.com/pvorb/node-md5

Homepage

 github.com/pvorb/node-md5#readme

Weekly Downloads

5,567,709

Version

2.3.0

License

BSD-3-Clause

Unpacked Size

Total Size

```
const BaseService = require("./BaseService");
const md5 = require("md5");
```

```
class AdminInfoService extends BaseService {
  constructor() {
    super( currentTableName: "admininfo");
  }
}
```

```
/**
 * 新增管理员信息
 * @param {{adminname, adminpwd, adminemail, adminintel, adminstatus}}
 * @return {Promise<boolean>} true代表新增成功,false代表新增失败
 */
async add({adminname, adminpwd, adminemail, adminintel, adminstatus}) {
```

```
  // 现在在这里,要执行sql语句的新增之前,我们要把密码进行md5的加密
```

```
  adminpwd = md5(adminpwd);
```

在这里，我们就把密码进行了一次加密操作

```
  let strSql = `INSERT INTO ${this.currentTableName} (adminname, adminpwd, adminemail, adminintel, adminstatus) VALUES`
  let result = await this.executeSql(strSql, {params: [adminname, adminpwd, adminemail, adminintel, adminstatus]});
  return result.affectedRows > 0;
}
```

对象admininfo @community (杨标) - 表

开始事务

文本

筛选

排序

导入

导出

id	adminname	adminpwd	adminemail	adminintel	adminstatus
54	杨标	e10adc3949ba59abbe56e057f20f883e	lovesnsfi@	18627193876	0

这个时候当我们再次去新增的时候，我们发现密码就已经不再是原来的明文存储了，而是像乱码一样的存储，这就是md5

到这一步为止，我们就已经完成了md5密码的加密操作

问题：现在有一个非常大的问题，当我们拿着这个加密以后的字符串去cmd5的网站去解密的时候，发现这个密码竟然可以反解密出来。如下所示

![(assets/Pasted image 20221025164154.png)]

这样就非常不安全，怎么办？

md5的加密是不可破解的，为什么cmd5的网站可以解密出来是因为123456太常见了，这个网站就一个一个的去试，它试出来。如果我们要把密码强大改大一点，这样别人就破解不了

第一种常见的解决方法

管理员姓名	杨标
管理员密码	123456
确认密码	123456
管理员邮箱	lovesnsfi@163.com
手机号码	18627193876
管理员状态	正常

保存
重置

**第一种方式：直接在界面上面就限定好，密码必须是一个复杂的长度或复杂的组成方式，这个后台在进行md5加密的时候，加密出来的密码就是一个不常见的md5字符串，别人就反解不了**

还有一种解决方案就是md5加密+加盐

## 八、md5加盐

上面的问题已经展示得非常清楚，我们在这里如果进行md5的简单值加密的时候是会破解的，我们在得到用户的密码以后，可以手动的在用户的密码后面或前面添加一个特殊的字符，以保证加密的字符串的不规则性

**第一步：我们在 config 目录下面的AppConfig下面配置的**

```

1  /**
2   * 整个程序的配置文件，所以需要配置的东西，我都写在这里
3   * @author 杨标
4   * @date 2022-10-24
5   */const path = require("path");
6
7  const AppConfig = {
8      excelDir: path.join(__dirname, "../excelDir"),
9      salt: "098lskdf.!@#09sdfj"
10 }
11
12 module.exports = AppConfig;

```

**第二步：在加密的时候，我们在原来用户输入的密码上面，添加了盐的值**

```

const md5 = require("md5");
const AppConfig = require("../config/AppConfig");

class AdminInfoService extends BaseService {
    constructor() {
        super({ currentTableName: "admininfo" });
    }

    /**
     * 新增管理员信息
     * @param {{adminname, adminpwd, adminemail, admintel, adminstatus}}
     * @return {Promise<boolean>} true 代表新增成功, false 代表新增失败
     */
    async add({adminname, adminpwd, adminemail, admintel, adminstatus}) {
        // 现在在这里，要执行sql 语句的新增之前，我们要把密码进行md5的加密
        adminpwd = md5(message: adminpwd+AppConfig.salt);
        let strSql = `INSERT INTO ${this.currentTableName} (adminname, adminpwd, adminemail, admintel, adminstatus) \
        let result = await this.executeSql(strSql, params: [adminname, adminpwd, adminemail, admintel, adminstatus]);
        return result.affectedRows > 0;
    }
}

```

**在密码的后面就加了一个盐**

完成上面的操作以后，当我们再去增管理员的进修，我们就可以看到，同样会新增成功，同样会得到一个加密的密码，如下所示

id	adminnamadminpwd	adminemaiadmintel	adminstatus	i:
55 杨标	677bc50bda3009a6b3f452d57a4b389e	lovesnsfi@18627193876		0

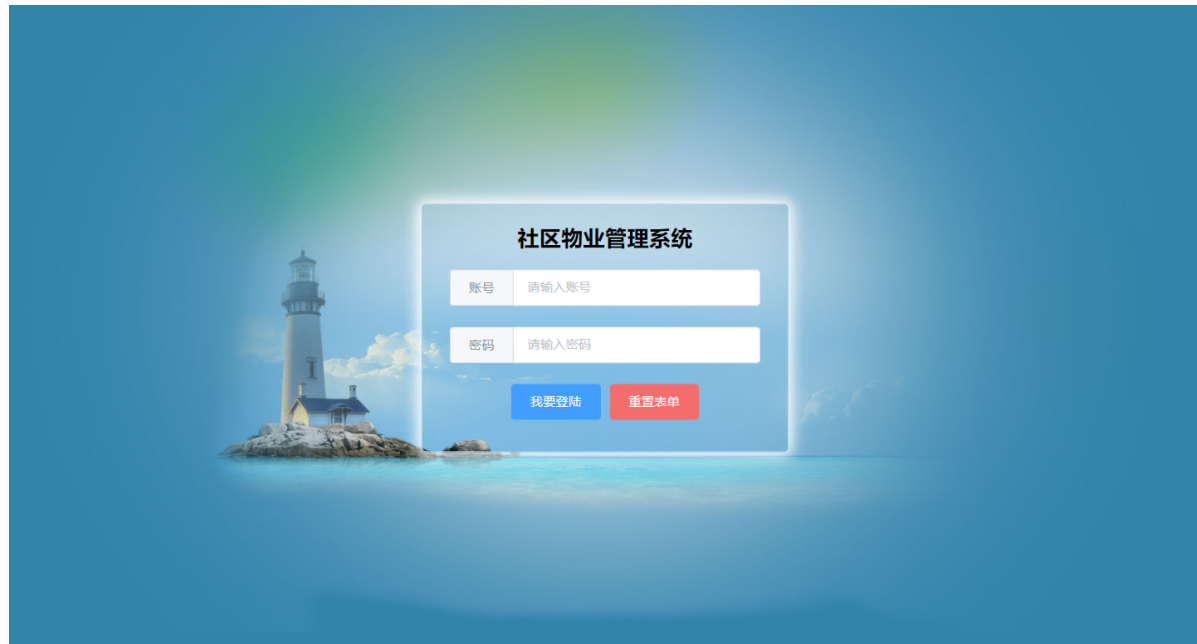
]

但是当我们再去拿着这个加密的密码去解密的时候，就得到不出结果了

![(assets/Pasted image 20221025165245.png)]

## 九、管理员的登录功能

### 登录界面效果图



```

1  <!DOCTYPE html>
2  <html lang="zh">
3  <head>
4      <meta charset="UTF-8">
5      <title>登录社区物业管理系统</title>
6      <link rel="stylesheet" href="./bootstrap/css/bootstrap.min.css">
7      <link rel="stylesheet" href="./bootstrap/font/bootstrap-icons.css">
8      <link rel="stylesheet" href="./js/layer/theme/default/layer.css">
9      <style>
10         .box {
11             position: fixed;
12             left: 0;
13             top: 0;
14             width: 100%;
15             height: 100%;
16             background-image: url("./img/admin-login-bg.pg.jpg");
17             background-size: 100% 100%;
18             /*这个就是让背景图片固定,不要随着滚动条去滚动*/
19             background-attachment: fixed;
20         }
21
22         .login-form {
23             width: 420px;
24             height: 300px;
25             background-color: rgba(255, 255, 255, 0.6);
26             position: absolute;
27             left: 0;

```

```

28         top: 0;
29         right: 0;
30         bottom: 0;
31         margin: auto;
32         box-shadow: 0 0 15px 5px white;
33         box-sizing: border-box;
34         padding: 20px;
35         transition: all 0.5s linear;
36         transform: scale(0) rotateY(-180deg);
37     }
38
39     .login-form.show {
40         transform: scale(1) rotateY(0deg);
41     }
42
43     </style>
44 </head>
45 <body>
46 <div class="box">
47     <form class="login-form" id="login-form">
48         <h3 class="text-center text-primary py-3 border-bottom border-primary">
社区物业管理系统</h3>
49         <div class="my-3">
50             <div class="input-group">
51                 <span class="input-group-text">
52                     账号
53                 </span>
54                 <input type="text" class="form-control" id="adminname"
name="adminname" placeholder="请输入账号"
55                     data-rule-required="true" data-msg-required="账号不能为
空">
56             </div>
57         </div>
58         <div>
59             <div class="input-group">
60                 <span class="input-group-text">密码</span>
61                 <input type="text" class="form-control" id="adminpwd"
name="adminpwd" placeholder="请输入密码"
62                     data-rule-required="true" data-msg-required="密码不能为空">
63             </div>
64         </div>
65
66         <div class="d-flex justify-content-around my-3">
67             <button type="button" class="btn btn-primary" id="btn-login">登录系统
</button>
68             <button type="button" class="btn btn-danger">重置表单</button>
69         </div>
70     </form>
71 </div>
72 </body>
73 <script src="../bootstrap/js/bootstrap.bundle.min.js"></script>
74 <script src="../js/jquery-3.6.1.js"></script>
75 <script src="../js/template-web.js"></script>
76 <script src="../js/layer/layer.js"></script>
77 <script src="../js/jquery.validate.js"></script>

```



```

78 <script src="./js/messages_zh.js"></script>
79 <script src="./js/base.js"></script>
80 <script>
81     $(function () {
82         //表单验证
83         var loginFormResult = $("#login-form").validate({
84             errorPlacement: function (error, element) {
85                 element.parent().after(error);
86             },
87             errorClass: "text-danger"
88         });
89
90         $(".login-form").addClass("show");
91
92         //登录系统
93         $("#btn-login").click(function () {
94             //第一步:表单验证
95             if (loginFormResult.form()) {
96                 console.log("表单验证成功")
97             } else {
98                 layer.alert("请检查你输入的内容");
99             }
100         });
101     })
102 </script>
103 </html>

```

上面的代码就是页面布局的基本代码

```

1 <script>
2     $(function () {
3         //表单验证
4         var loginFormResult = $("#login-form").validate({
5             errorPlacement: function (error, element) {
6                 element.parent().after(error);
7             },
8             errorClass: "text-danger"
9         });
10
11         $(".login-form").addClass("show");
12
13         //登录系统
14         $("#btn-login").click(async function () {
15             //第一步:表单验证
16             if (loginFormResult.form()) {
17                 //第二步:应该将用户名与密码发送到后台 ,然后去进行验证
18                 //发送请求,我们肯定是用ajax来进行的
19                 try {
20                     let result = await
21 request.post(`${baseUrl}/adminInfo/checkLogin`, {
22                     adminname: $("#adminname").val(),
23                     adminpwd: $("#adminpwd").val()
24                 });
25                 console.log(result.data);
26
27                 sessionStorage.setItem("adminInfo", JSON.stringify(result.data));
28             }
29         });
30     });
31 </script>

```

```

26             //登录以后的信息要保存
27             location.replace("manager.html");
28         } catch (error) {
29             console.log(error);
30             layer.alert("服务器错误");
31         }
32     } else {
33         layer.alert("请检查你输入的内容");
34     }
35 });
36 })
37 </script>

```

在这里也要注意一下，我们登录成功以后，我们要把用户的登录信息保存在 `sessionStorage` 里面，方便其它的页面去使用

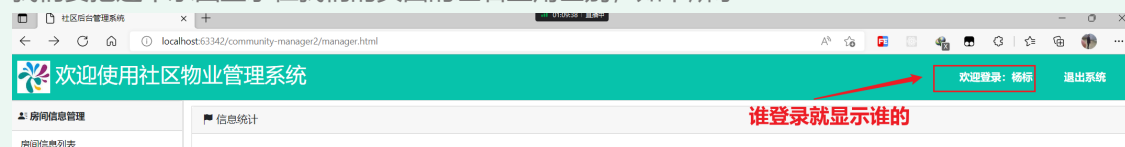
#### manager.html

```

1  <script>
2      $(function () {
3          if (sessionStorage.getItem("adminInfo")) {
4              let adminInfo = JSON.parse(sessionStorage.getItem("adminInfo"));
5              $("#adminName").text(adminInfo.adminname)
6          }
7      })
8  </script>

```

我们要把这个东西显示在我们的页面的左右上角区别，如下所示



#### AdminInfoServcie.js

```

1  /**
2   * 检测用户名的登录
3   * @param {{adminname, adminpwd}}
4   * @return {Promise<Object>} 返回用户登录的对象, 有可能为空
5   */
6  async checkLogin({adminname, adminpwd}){
7      adminpwd = md5(adminpwd+AppConfig.salt);
8      let strSql = `select * from ${this.currentTableName} where isDel = false and
9      adminname=? and adminpwd = ? `;
10     let result = await this.executeSql(strSql, [adminname, adminpwd]);
11     return result[0];
12 }

```

这里有一个小细节要注意，我们的 `adminpwd` 因为在之前新增的时候是进行了 `md5` 的加密与加盐操作的，所以在登录的时候，我们也要加密与加盐来进行

#### adminInroRouter.js

```

1 //处理登录的请求
2 router.post("/checkLogin", async (req, resp) => {
3     //拿到用户在前台传递过来的参数以后,我们要进行数据库的校验操作
4     let result = await
    ServiceFactory.createAdminInfoService().checkLogin(req.body);
5     Reflect.deleteProperty(result, "adminpwd");
6     //delete result.adminpwd;
7     let resultJson = new ResultJson(Boolean(result), result ? "登录成功" : "登录失
    败", result);
8     resp.json(resultJson);
9 })

```

在这里也要注意一下,我们的密码是能发送到前端浏览器的,所以们要通过  
`Reflect.deleteProperty` 或使用ES5里面的 `delete` 来删除对象的某一个属性

## 十、页面的权限验证

当用户在没有登录的进修,我们是不让它进入到其它的页面的,我们可以看到在这个地方,每个页面都加载了 `base.js` 的这个文件,所以我们可以在这个文件上面进行相应的操作

```

1 // 当页面渲染完成以后,去检查一下,是否有登录,如果没有登录就去登录
2 document.addEventListener("DOMContentLoaded", function (event) {
3     let adminInfo = sessionStorage.getItem("adminInfo")
4     if (adminInfo) {
5         //说明用户已经登录过了
6     } else {
7         //说明用户没有登录
8         //如果用户正好就在登录界面,我们就不管了
9         // /login.html
10        if (/\/login\.html/.test(location.pathname)) {
11            //正好处理登录页面,不我们不用管
12        } else {
13            location.replace("login.html");
14        }
15    }
16 });

```

上面的代码就是当页面加载完成以后,我们再去判断是否有登录,如果没有登录,我们就跳转到登录页面

但是这种写法并不完善,正常情况下是页面还没有加载或正在加载的时候,就要判断是否有登录,所以我们要换一个事件判断的方法,

### 第二种方式

```

1 document.addEventListener("readystatechange", function () {
2     if (document.readyState === "interactive") {
3         let adminInfo = sessionStorage.getItem("adminInfo")
4         if (adminInfo) {
5             //说明用户已经登录过了
6         } else {
7             //说明用户没有登录
8             //如果用户正好就在登录界面,我们就不管了
9             // /login.html

```

```

10         if (/\login\.html/.test(location.pathname)) {
11             //正好处理登录页面,不我们不用管
12         } else {
13             location.replace("login.html");
14         }
15     }
16 }
17 })

```

这种方式是在页面 `inactive` 的时候就判断了，这样会更好一些

```

<button type="button" class="btn btn-danger" >退出登录</button>
</div>
</form>
</div>
</body>
<script src="./bootstrap/js/bootstrap.bundle.min.js"></script>
<script src="./js/jquery-3.6.1.js"></script>
<script src="./js/template-web.js"></script>
<script src="./js/layer/layer.js"></script>
<script src="./js/jquery.validate.js"></script>
<script src="./js/messages_zh.js"></script>
<script src="./js/base.js"></script>
</script>

```

现在我们是在所有的页面都导入了 `base.js` 这个文件。这个文件导入在了 `body` 标签的结束，也就是整个页面加载完了才会触发加载这个js。这个时候的js代码就不能写在 `body` 标签的后面，应该写在页面加载之前

1. 没有defer或async属性，浏览器会立即加载并执行相应的脚本。  
不等待后续加载的文档元素，读到就开始加载和执行，此举会阻塞后续文档的加载
2. 有了async属性，表示后续文档的加载和渲染与js脚本的加载和执行是并行进行的，即异步执行；
3. 有了defer属性，加载后续文档的过程和js脚本的加载是并行进行的(异步)，此时的js脚本仅加载不执行，js脚本的执行需要等到文档所有元素解析完成之后，DOMContentLoaded事件触发执行之前。

总结来说了：async和defer都不会阻止页面的加载，但是 `async` 在加载完 `js` 代码以后会立即执行，`defer` 不阻止页面加载，如果加载完了也不会立即执行，它要等到页面加载完了以后执行

```

1 <script src="./js/base.js" async></script>

```

我们把每个页上面的 `base.js` 的这一个加载放在了 `head` 里面，并且加上了一个 `async` 的加载，它就是异步加载执行的

当然还少不了另一个东西，就是退出登录

```

1 $("#log-out").click(function (){
2     layer.confirm("确定要退出登录吗",{
3         btn:["确定","我点错了"],
4         btn1:function (){
5             sessionStorage.removeItem("adminInfo");
6             location.replace("login.html");
7         }
8     });
9 })

```

在manager的页面上面，我们可以做一个退出登录的功能，如上代码

## 十一、接口请求的权限验证

虽然在上面一个章节里面，我们完成了页面级别的权限验证，但是最重要的一点数据安全还没有做，如在没有登录的情况下，我们去请求下面的地址

```
1 http://127.0.0.1:16888/roomInfo/getList
```

我们仍然可以获取到数据，所以怎么们确定用户必须登录以后才可以获取数据呢

什么是接口，接口就是一个 `http` 的请求，它一般情况下就返回一个 `json` 的数据，所以我们刚刚的 `router.get()` 或 `router.post()` 都是接口  
当我们去请求这些接口的时候，它就会返回数据  
但是这么做就会有问题，我们的接口是没有做权限验证的，这个时候，当我们进行相就的请求的时候，就直接返回数据了，所以即使用户没有登录，也可以通过这个接口来得到数据，这样做非常危险

目前能够做到接口安全验证的方案有很多，最常见的一种就是 `token`，在 `http` 的 Web 下面，最流行的方案叫 `JWT`

`JWT` 的全名叫 `json web token`

### 简介

`JSON Web Token (JWT)` 是一个轻量级的认证规范，这个规范允许我们使用 `JWT` 在用户和服务器之间传递安全可靠的信息。其本质是一个 `token`，是一种紧凑的 `URL` 安全方法，用于在网络通信的双方之间传递。

在使用这个东西之前，我们先理解一个原理 就是 `http` 请求的原理。大家都知道，我们现在所学习的所有的请求都是基于 `http` 的，如下所示

```
1 http://localhost:63342/community-manager2/login.html
2 http://127.0.0.1:16888/roomInfo/getList
```

我们现在所学习的所有的请求地址都是 `http` 的，那么，什么是 `http` 呢

  收藏  3456  733

 本词条由“科普中国”科学百科词条编写与应用工作项目 审核。

超文本传输协议（Hyper Text Transfer Protocol，HTTP）是一个简单的请求-响应协议，它通常运行在 `TCP` 之上。它指定了客户端可能发送给服务器什么样的消息以及得到什么样的响应。请求和响应消息的头以 `ASCII` 形式给出；而 <sup>[9]</sup> 消息内容则具有一个类似 `MIME` 的格式。这个简单模型是早期 `Web` 成功的有功之臣，因为它使开发和部署非常地直截了当。

上面的解释当中我们只用看以下几个点就可以了

1. 所有的 `http` 都有 2 个过程，一个叫请求 `request`，还有一个叫响应 `response`
2. 所有的请求和响应里面都包含 2 个部分，一个叫 `header` 头，还有一个叫 `body` 内容

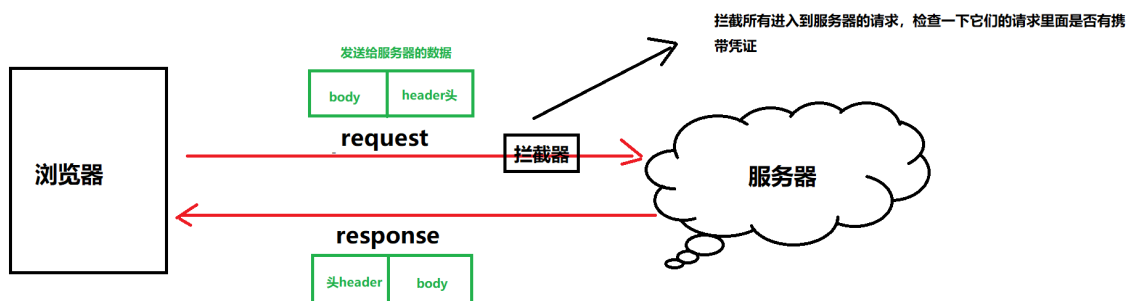


我们以前使用的是 **body** 请求体的部分，那么这个头header到底有什么作用

```
// 添加一个拦截器
app.use((req: Request<RouteParameters<string>, any, any, ParsedQs, Record<string, any>>, resp: Response<any, Record<string, any>>, next: NextFunction) => {
  // req: 浏览器到服务器的请求
  // resp: 服务器返回浏览器的
  // next: 是否放行
  // 颁发跨域能行证
  resp.setHeader( name: "Access-Control-Allow-Origin", value: "*");
  resp.setHeader( name: "Access-Control-Allow-Methods", value: "GET,POST,PUT,DELETE,OPTIONS");
  resp.setHeader( name: "Access-Control-Allow-Headers", value: "Content-Type");
  next();
});
```

这个东西大家应该很熟悉，这是用于ajax跨域的响应头，加了这个头以后，我们就可以实现 **CORS** 的跨域

http请求里面，数据最先到达服务器的是 **header**，而浏览器最先得到响应的也是 **header**



这个凭证由谁来发???

现在是由浏览器访问服务器，所以这个令牌应该是由服务器来发放，那么发放的这个令牌我们就叫 **token**，做的这个令牌就叫 **JWT**

## 十二、JWT令牌的发放

刚刚也说过，令牌就是一个 **token**，它由服务器发放，服务器可以通过一个第三方技术来进行，这个技术就是 **json web token**【这个东西就是专门用于打造令牌的】

安装json web token

```
1 $ npm install jsonwebtoken
```

### 颁发令牌

首先我们要搞清楚一个问题，我们在什么时候颁令牌？我们不可能对所有的人都发布令牌，只有登录成功以后的人才发布令牌，所以我们应该在登录的路由那里颁发令牌

在颁发令牌之前，我们一定要做防伪码【加密码】

上面的 `secret` 就是我们的防伪码

正面就是我们得到的token的结果

这个时候我们在前端已经得到这了这样的一个token了

```

▼ {id: 55, adminname: '杨标', adminemail: 'Lovesnsfi@163.com', admintel: '18627193876', adminstatus: 0, ...}
    adminemail: "lovesnsfi@163.com"
    adminname: "杨标"
    adminstatus: 0
    admintel: "18627193876"
    id: 55
    isDel: 0
    token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZGUiOmZm8iOmsiaWQiOiJlcnBm5hbmVwIiwiaWF0IjEzMjY3NTBjZGZzMDA5YyJ9.eyJhZGUiOmZm8iOmsiaWQiOiJlcnBm5hbmVwIiwiaWF0IjEzMjY3NTBjZGZzMDA5YyJ9"
    [[Prototype]]: Object

```

## 十三、携带token进行请求

现在我们已经得到了这个 `token` 问题就在于如果在这个地方每次发起 `ajax` 请求的时候都携带这个 `token` 呢

```
1  const request = {
2    get(str, data = {}) {
3      return new Promise((resolve, reject) => {
4        $.ajax({
5          method: "GET",
6          url: str,
7          dataType: "json",
8          data: data,
9          beforeSend: (xhr) => {
10             //在每次发起请求之前,我都从缓存里面拿到adminInfo
11             let adminInfoStr = sessionStorage.getItem("adminInfo");
12             if (adminInfoStr) {
13               let adminInfo = JSON.parse(adminInfoStr);
14               xhr.setRequestHeader("authorization", adminInfo.token);
15             }
16           },
17           success: result => {
18             //这里的success只能代表请求回来了,并不能代码这个地方的请值是成功的
19             if (result.status === "success") {
20               resolve(result);
21             } else {
22               reject(result);
23             }
24           },
25           error: error => {
26             reject(error);
27           }
28         })
29       });
30    },
31    post(str, data = {}) {
32      return new Promise((resolve, reject) => {
33        $.ajax({
34          method: "POST",
35          url: str,
36          dataType: "json",
37          data: data,
38          beforeSend: (xhr) => {
39             //在每次发起请求之前,我都从缓存里面拿到adminInfo
40             let adminInfoStr = sessionStorage.getItem("adminInfo");
41             if (adminInfoStr) {
42               let adminInfo = JSON.parse(adminInfoStr);
43               xhr.setRequestHeader("authorization", adminInfo.token);
44             }
45           },
46           success: result => {
47             if (result.status === "success") {
48               resolve(result);
49             } else {
50               reject(result);
51             }
52           }
53         })
54       });
55    }
56  }
```



```

52         },
53         error: error => {
54             reject(error);
55         }
56     });
57 });
58 }
59 }

```

#### ▼ 请求标头

⚠ 临时标头已显示 [了解详细信息](#)

**Accept:** application/json, text/javascript, \*/\*; q=0.01

**authorization:** eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhZG1pbklzM8iOnsiaWQiOiJ1LCJhZG1pbm5hbWUiOiLmr  
oIciLCJhZG1pbmB3ZCI6IjY3NTJjNTBiZGEzMDA5YTZiM2Y0NTJkNTdhNGIzODI1IiwiaWRTaw5lbWFPbCI6ImxvdmVzbnNmaUA  
MuY29tIiwiaWRTaw50ZWwiOiIxODYyNzE5Mzg3NiIsImFkbWluc3RhdHVzIjowLCJpc0RlbCI6MH0sIm1hdCI6MTY2Njc2NTU4C  
ZXhwIjoxNjY2NzY5MTg5fQ.6rPUiDdt9pfRu9z5viT3rsTM31\_yVXa...

**Content-Type:** application/x-www-form-urlencoded; charset=UTF-8

**Referer:** http://localhost:63342/

**sec-ch-ua:** "Chromium":v="106", "Microsoft Edge":v="106", "Not:A=Brand":v="99"

当我们再次去发起请求的时候，我们就可以看到这个请求头里面会携带一个 **authorization** 的头，它里面的内容就是我们的 **token**

## 十四、解决自定义请求头的错误问题

当我们的请求携带了自定义请求头以后，这个时候再去通过 **ajax** 请求应该会报错。如下所示

名称	状态	类型	发起程序	大小	时间	履行者	时间线
checkLogin		xhr	jquery-3.6.1.js:10135	0 B	41 ms		
checkLogin	200	preflight	预检 (1)	0 B	9 ms		

现在打开控制台，看一下它报错的原因是什么

现在它又报 **ajax** 跨域错误

```

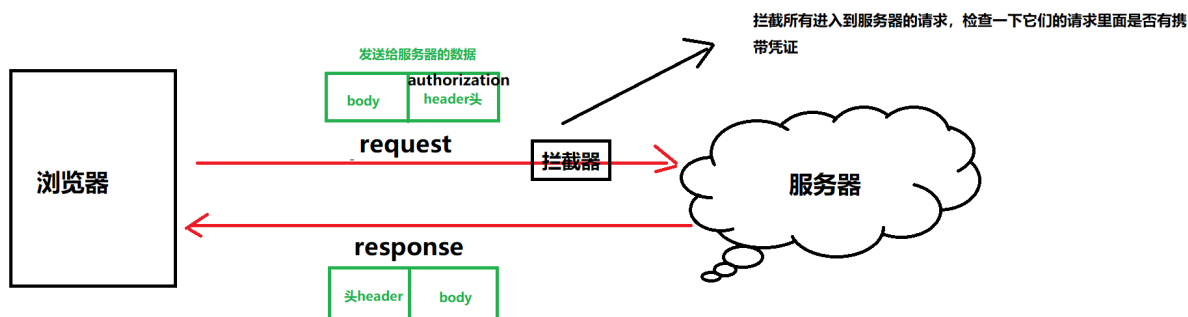
1 //添加一个拦截器
2 app.use((req, resp, next) => {
3     //req: 浏览器到服务器的请求    //resp: 服务器返回浏览器的    //next: 是否放行
4     //颁发跨域能行证
5     resp.setHeader("Access-Control-Allow-Origin", req.headers.origin || "*");
6     resp.setHeader("Access-Control-Allow-Methods", "GET,POST,PUT,DELETE,OPTIONS");
7
8     resp.setHeader("Access-Control-Allow-Headers", "Content-Type, authorization");
9
10    next();
11 });

```

在这里我们可以看到，我们在 `Access-Control-Allow-Headers` 里面添加了 `authorization` 这个头，这样就又解决了跨域的问题了

## 十五、编写后台拦截器，检查是否携带token

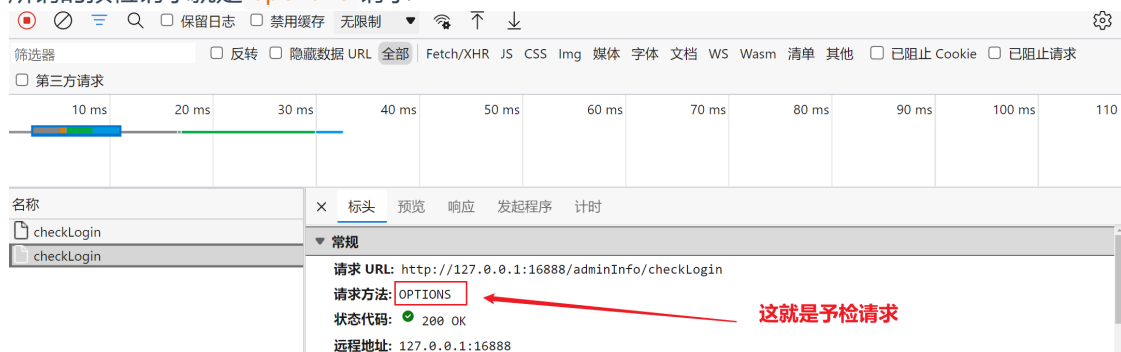
按照我们的思路，我们正常情况要检测用户的请求是否携带token,思路如下



请注意：Ajax的post请求一旦跨域以后，它会在发起POST请求之前先做一次预检请求  
预检请求只有会以下两种条件同时成立才会发送

1. 你要是POST请求
2. 你要是跨域请求

所谓的预检请求就是 `options` 请求



```
1
2 //再添加一个拦截器
3 app.use((req, resp, next) => {
4   //第一步：检查请求是否是预检的options请求
5   if (req.method.toUpperCase() === "OPTIONS") {
6     next();
7   } else {
8     let token = req.header("authorization");
9     if (token) {
10      //说明这个人带了令牌，我们就要去验证这个令牌
```

```

11         try {
12             let decode = jwt.verify(token, AppConfig.secret);
13             //如果没有报错,就说明是令牌验证成功了
14             next();
15         } catch (e) {
16             //如果报错了,就说明是令牌验证失败了
17             let resultJson = new ResultJson(false, "令牌验证失败");
18             resp.status(403).json(resultJson);
19         }
20
21     } else {
22         //说明这个人没有带令牌
23         //1. 这是一些特殊的路径, 要直接放行, 如登录
24         let flag = AppConfig.noRequireAuth.some(item => item.test(req.path));
25
26         if (flag) {
27             //说明是特殊路径, 我们要放行
28             next();
29         } else {
30             //2. 说明不是特殊路径, 就直接返回403的状态
31             let resultJson = new ResultJson(false, "请求未授权");
32             resp.status(403).json(resultJson);
33         }
34     }
35 }
36 });

```

我们在 `app.js` 里面设置了一个拦截器, 通过相应的逻辑去拦截它  
同时要注意, 这个拦截器一定要在 `CORS` 跨域拦截器的后面才可以

但是现在就会有一个问题, 如果我们已经登录了, 但是长时间不操作页面, 操作了我 `token` 的授权时限【目前我们的授权只有1个小时】。当 `token` 过期或缓存丢失的时候就会出现下面的情况

用户已经登录了, 但是因为某些特殊原因, token丢失了, 或失效了  
但是点击查询的时候就会报403的错误

全选	房间编号	业主姓名	性别	房间面积	房间状态	业主账号	身份证号	手机号码	业主邮箱	操作
<input type="checkbox"/>	1-1-101	0	男	100	自住	11	42058319990303003X	188****8888	xxiaohanxxx163.com	
<input type="checkbox"/>	1-2-101	0	男	200	自住	42	420581202210173306	188****3019	12138@qq.com	

名称	状态	类型	发起程序	大小	时间	履行者	时间线
getListByPage?roomname=&ownname=&tel...	403	xhr	jquery-3.6.1.js:10135	0 B	11 ms		

403的状态码就是没授权, 没授权, 那我就去权限就可以了啊, 这样时候应该重新跳转到登录页面, 让用户重新授权

## 十六、ajax的响应拦截

当 ajax 的请求返回的状态码是 403 的时候，我们就知道它没有授权，我们要让它跳转到 login.html 的页面重新登录

```
1  const request = {
2    get(str, data = {}) {
3      return new Promise((resolve, reject) => {
4        $.ajax({
5          method: "GET",
6          url: str,
7          dataType: "json",
8          data: data,
9          beforeSend: (xhr) => {
10             //在每次发起请求之前,我都从缓存里面拿到adminInfo
11             let adminInfoStr = sessionStorage.getItem("adminInfo");
12             if (adminInfoStr) {
13               let adminInfo = JSON.parse(adminInfoStr);
14               xhr.setRequestHeader("authorization", adminInfo.token);
15             }
16             return xhr;
17           },
18           success: result => {
19             //这里的success只能代表请求回来了，并不能代表这个地方的请值是成功的
20             if (result.status === "success") {
21               resolve(result);
22             } else {
23               reject(result);
24             }
25           },
26           error: (error) => {
27             if (error.status === 403) {
28               //说明ajax的请求没有授权
29               top.location.replace("login.html");
30             }
31             console.log(error)
32             reject(error);
33           }
34         })
35       });
36     },
37     //post与get是一样的操作，所以在这里就不再列举.....
38   }
```

## 十七、关于 window.open() 打开的页面如果设置授权

在我们的项目里面，我们经常会使用 `window.open()` 打开一个链接，然后下载一个 `excel` 文件，但是通过 `window.open()` 打开的链接是不能设置 `header` 的，所以我们只能采用另一种方式来进行

```
// 点击导出excel的按钮以后
$("#btn-export-excel").click(function () {
    // 现在我们教一下大家快速的生成search参数
    let urlSearchParams = new URLSearchParams();
    urlSearchParams.append("roomname", $("#roomname").val());
    urlSearchParams.append("ownername", $("#ownername").val());
    urlSearchParams.append("telephone", $("#telephone").val());
    let adminInfoStr = sessionStorage.getItem("adminInfo");
    if (adminInfoStr) {
        let adminInfo = JSON.parse(adminInfoStr);
        urlSearchParams.append("authorization", adminInfo.token);
    }
    // 这个时候无非就是向后台发送请求，查询数据库的结果，然后将这个结果转换成excel，然后下载这个文件
    let str = `${baseUrl}/roomInfo/exportToExcel?${urlSearchParams.toString()}`;
    // 我们其它的请求都是ajax请求，唯独导出excel的时候使用的是window.open()
    // ajax的请求已经被我们设置了`token`，但是`window.open`没有设置token啊
    window.open(str);
});
```

我们在地址栏的 `search` 参数上面添加了一个 `authorization` 的参数，然后在后台的拦截器里面做如下更改

```
app.use((req : Request<RouteParameters<string>, any, any, ParsedQs, Record<string, any>>, resp : Response<any, Record<string, any>>, ne
    // 第一步：检查请求是否是预检的options请求
    if (req.method.toUpperCase() === "OPTIONS") {
        next();
    } else {
        let token = req.header( name: "authorization") || req.query.authorization;
        if (token) {
            // 说明这个人带了令牌，我们就要去验证这个令牌
            try {
                let decode = jwt.verify(token, AppConfig.secret);
                // 如果没有报错，就说明是令牌验证成功了
                next();
            } catch (e) {
```

在获取授权的时候就不仅仅只是从header里面获取，还可以从body里面获取了

## 结束

1. `html+css` 布局
2. 弹性盒子
3. `bootstrap` ,以及 `bootstrap iconfont`
4. `art-template` 进行模板渲染
5. `layer` 插件
6. `jquery` 及 `jquery` 相关的插件，如表单验证
7. `echarts` 图形图表
8. `ajax` 的前后端数据交互
9. `express` 后台搭建
10. `mysql` 数据库，建表，建库，软删除，Service层应用，框架搭建
11. `ES6` 语法，`async/await` 箭头函数，`Promise`，`const/let`，`Reflect`，`CommonJs` 模块化，`class` 及 `extends` 的使用
12. `nodemon` 热重启，`package.json` 的配置
13. `md5` 加密与加盐处理
14. `excel` 的导出，`nodejs` 对 `excel` 的处理
15. `jwt` 权限验证，拦截器的使用
16. `CORS` 中 `ajax` 的跨域处理
17. `http` 的状态码使用，`GET` 及 `POST` 的应用，`body-parser` 的使用
18. `try...catch` 异常的处理，以及 `express-async-errors` 全局异常的处理

19. `localStorage`、`sessionStorage`、`cookie` 缓存的应用, `js-cookie` 的使用