数据可视化echarts操作

效果图

![[Pasted image 20221025090900.png]]

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

```
/**
 * 用于数据展示的Service操作，不涉及到任何的具体的表
 */
const BaseService = require("./BaseService");

class DataViewService extends BaseService {
    constructor() {
        super();
        //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
    }

    /**
     * 获取计算的总数
     * @returns {Promise<Object>} 返回查询的结果的对象
     */
    async getCalcData() {
        let strSql = `select
         max(if(adminCount='adminCount',totalCount,0)) 'adminCount',
         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}
        union allselect 'roomCount',count(*) 'totalCount' from ${this.tableMap.roominfo}
        union all        select 'moneyCount',sum(totalmoney) 'totalMoney' from ${this.tableMap.costinfo}
        union all        select 'costTypeCount', count(*) 'totalCount' from ${this.tableMap.costtype}) a`;
        let result = await this.executeSql(strSql);
        return result[0];
    }
}


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
> ![[Pasted image 20221025093924.png]]

当我们把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

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

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

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

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

    static createCostInfoService() {
        const CostInfoService = require("../services/CostInfoService");
        return new CostInfoService();
    }

    static createDataViewService(){
        const DataViewService = require("../services/DataViewService");
        return new DataViewService();
    }
}

module.exports = ServiceFactory;
```

四、完成路由dataViewRouter.js

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


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

```
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

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

router.get("/getCalcData", async (req, resp) => {
    let result = await ServiceFactory.createDataViewService().getCalcData();
    let resultJson = new ResultJson(Boolean(result), result ? "获取数据成功" : "获取数据失败", result);
    resp.json(resultJson);
});


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

```
<script>
    $(function () {
        async function getCalcData() {
            try {
                let result = await request.get(`${baseURL}/dataView/getCalcData`);
                $("#adminCount").text(result.data.adminCount);
                $("#costTypeCount").text(result.data.costTypeCount);
                $("#moneyCount").text(result.data.moneyCount);
                $("#roomCount").text(result.data.roomCount);
            } catch (error) {
                console.log(error)
            }
        }

        getCalcData();
    })
</script>
```