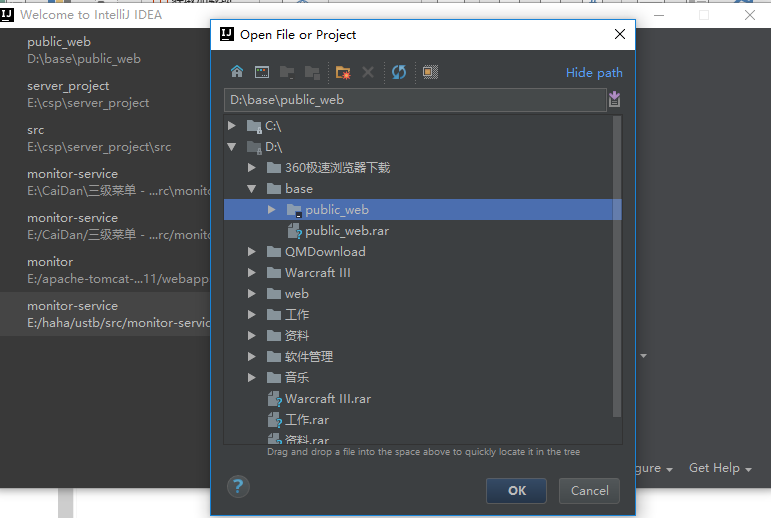
# 新框架HMI操作手册

## 打开初始HMI项目

（在需要进行一个项目之前请确保自己电脑上的环境和软件已经配置完全，目前推荐使用idea编辑器）

用ideal打开初始项目，点击open找到你的项目地址打开

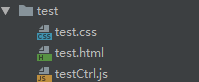


## 页面创建

#### 创建页面所需文件

(在view/page文件夹下按模块新建文件夹文件夹新建，html，css，js文件（css文件是Stylesheet）)

命名规则：html和css名字应由该页面名称的英文拼接而成，大小写规则同上，js的名字在后面加Ctrl



HTML

打开刚刚创建的HTML文件，将其中默认的代码全部删除，并插入如下代码

<div class="test">  
   
</div>

其中**test**改为相应的页面名称（不加.html），此后该页面HTML开发都在这个div元素中进行，该div为底层div

**Js文件:**

打开刚刚创建的JS文件，此时其中默认应该是空的，在其中添加如下代码：

monitor.**controller**('testCtrl', function ($scope, $state, $stateParams, ngDialog , $interval,GetHmiParam,GetLogData,SendMessage,SendTag,GetLogData, commonFindData

) {  
//具体开发  
})

其中**mainCtrl**改为相应页面的Ctrl的名称（不加.js），此后该页面的JavaScript开发都在花括号{ }中进行。

**Css文件**

打开刚刚创建的CSS文件，此时应该是空的，在接下来的开发时，如果需要通过该CSS对相应页面进行样式的定义，应遵循： .class名称+空格+元素 这种格式，其中的class名称即为对应HTML文件中底层div的类class。

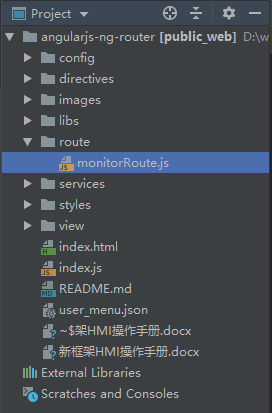
例如

.**test button**{**border**: **#2f5597** 1**px solid**;**background-color**: **unset**}

#### 创建路由

每个页面对应一个url(在路由控制)

在总目录下route文件夹下面找到monitorRoute.js打开



在monitorRoute.js中添加以下代码

.state('monitor-web.test, {  
 **url**: '/test,  
 **templateUrl**: root + page/test/test.html' + **version**,  
 **controller**: 'testCtrl',  
 **title**: "测试",  
 **resolve**: {  
 loadMyCtrl: ['$ocLazyLoad', function ($ocLazyLoad) {  
 return $ocLazyLoad.load({  
 **name**: "测试",  
 **files**: [  
 root + "page/test/testCtrl.js" + **version**,  
 root + "page/test.css" + **version**,  
 ]  
 })  
 }]  
 }  
})

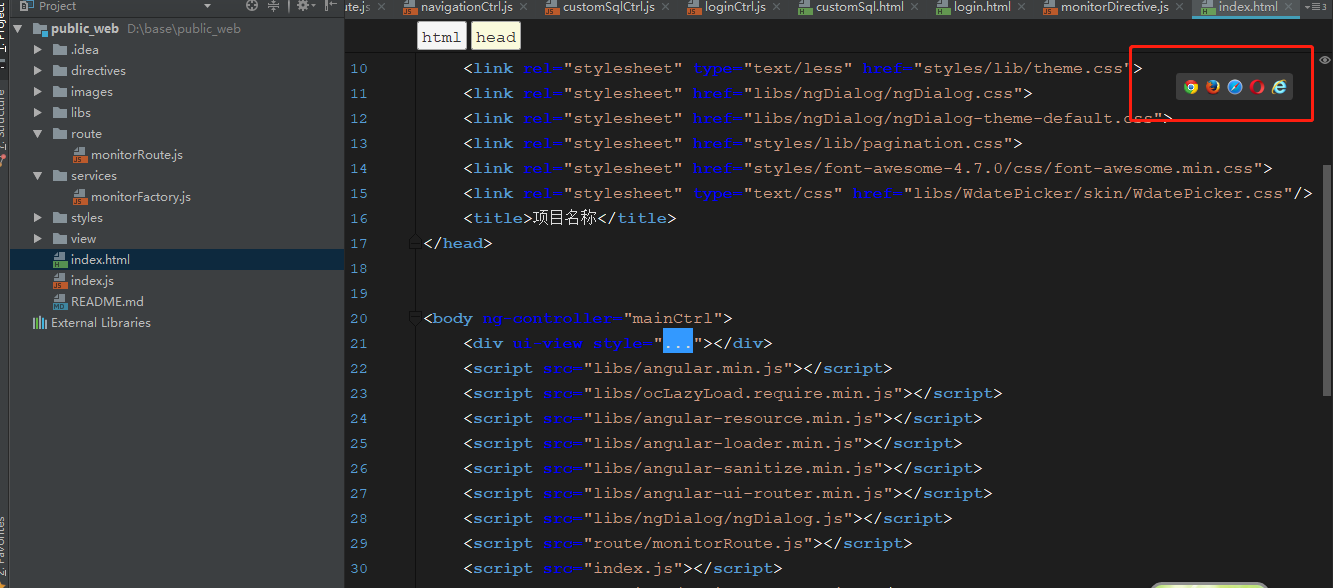
在在最下面两行代码上方编写如下代码，参照实际文件名，仿照上述代码

.state(**'monitor-web.新建页面文件名'**, {  
 **url**: **'/新建页面文件名'**,  
 **templateUrl**: root + **'****外部文件夹名称 /新建页面文件名.html'** + **version**,  
 **controller**: **'****新建页面文件名Ctrl'**,  
 **title**: **"页面标签上显示的名字"**,  
 **resolve**: {  
 **loadMyCtrl**: [**'$ocLazyLoad'**, **function**($ocLazyLoad) {  
 **return** $ocLazyLoad.load({  
 **name**: **"新建页面文件名"**,  
 **files**: [  
 root + **"外部文件夹名称/新建页面文件名Ctrl.js"** + **version**,

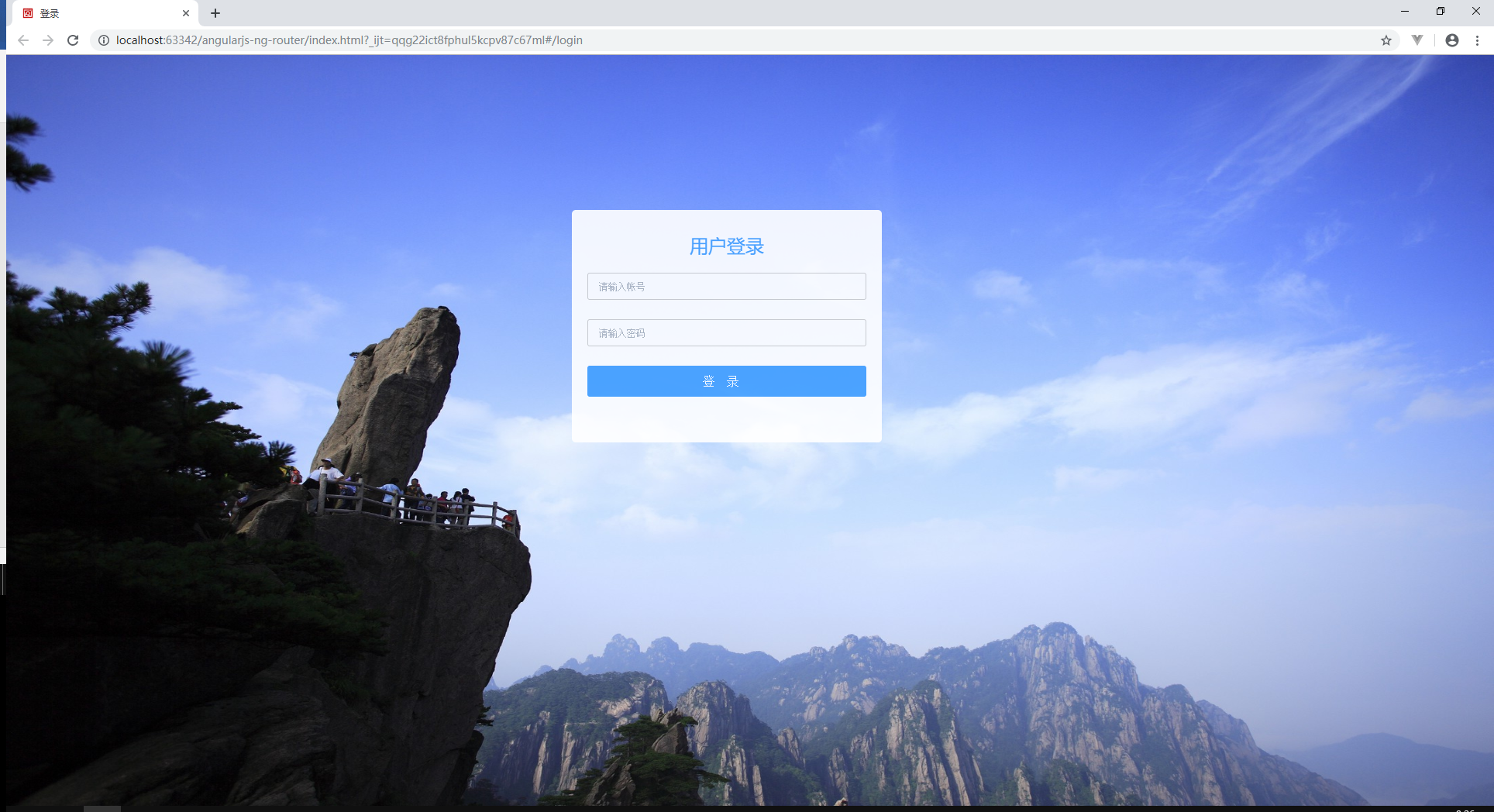
root + **"外部文件夹名称/新建页面文件名.css"** + **version**  
 ]  
 })  
 }]  
 }  
})

其中红色字体的根据实际文件名称写，css如果没有可以不用引，但是js必须要引，否则无法通过路由打开相应页面。

配置好后我们可以进行测试，首先我们找到index.html文件打开，



红框中可以自己选择浏览器，这里推荐使用谷歌浏览器打开



然后测试我们自己设置的页面将地址改为

<http://localhost:8080/public_web/index.html#/monitor-web/>新页面url名

到此一个页面创建完成

#### 给新页面配置导航菜单

(修改user\_menu.json文件)

在下方加入一个用户 例如：张三

并且为其添加菜单 一级菜单url不需要配置，二级菜单url既是新建页面的名称

{  
 **"name"**: **"张三"**,  
 **"account"**: **"zhangsan"**,  
 **"password"**: **"123456"**,  
 **"menu"**: [  
 {  
 **"menu\_name"**: **"测试一级画面"**,  
 **"menu\_url"**: **""**,  
 **"children"**: [  
 {  
 **"menu\_name"**: **"测试画面"**,  
 **"menu\_url"**: **"test"** }  
 ]  
 }  
 ]  
}

然后退出登陆，使用使用（zhangsan/123456 登陆系统即可看到test导航菜单）

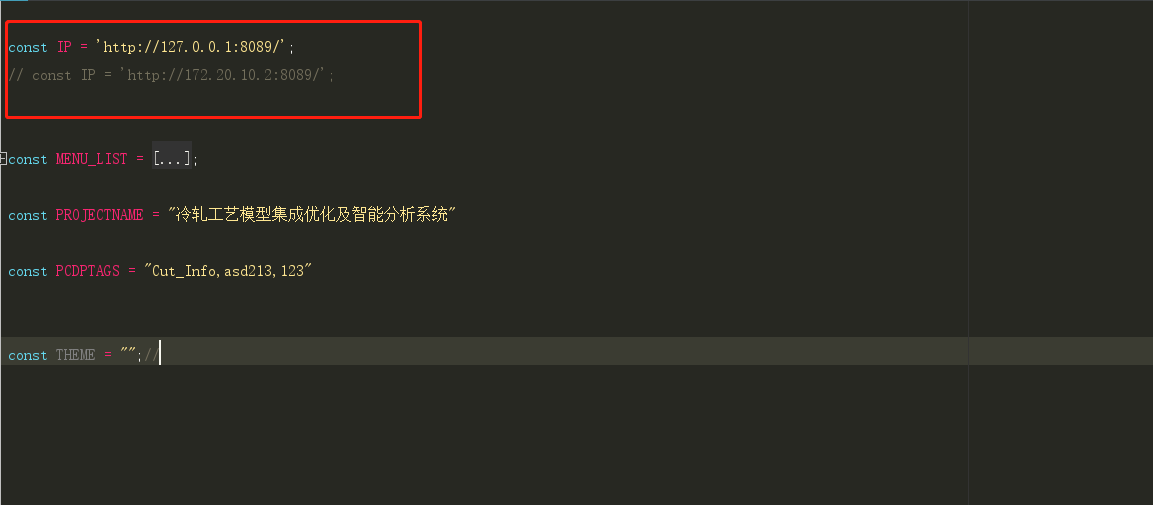




## HMI系统基本配置

#### 1、配置java服务ip地址

在项目目录中config/config.js中的const IP是对服务器java地址，默认使用‘http://127.0.0.1:8080/service\_bkgy’，如果java服务和前端程序不在同一机器，需更改对应IP



#### 2、配置项目名称

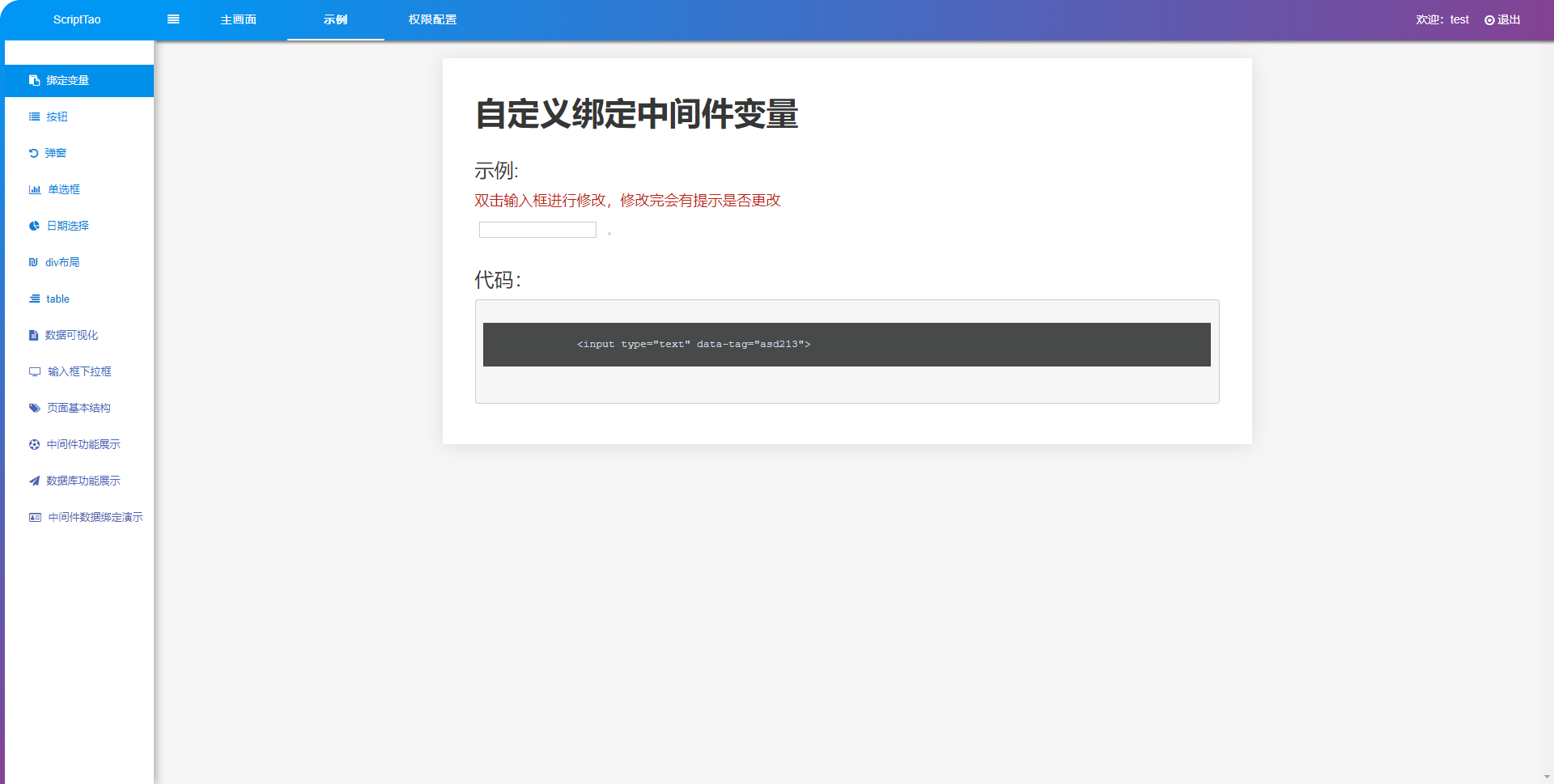
在项目目录中config文件夹下找到config.js文件 如下

const PROJECTNAME = "冷轧工艺模型集成优化及智能分析系统"

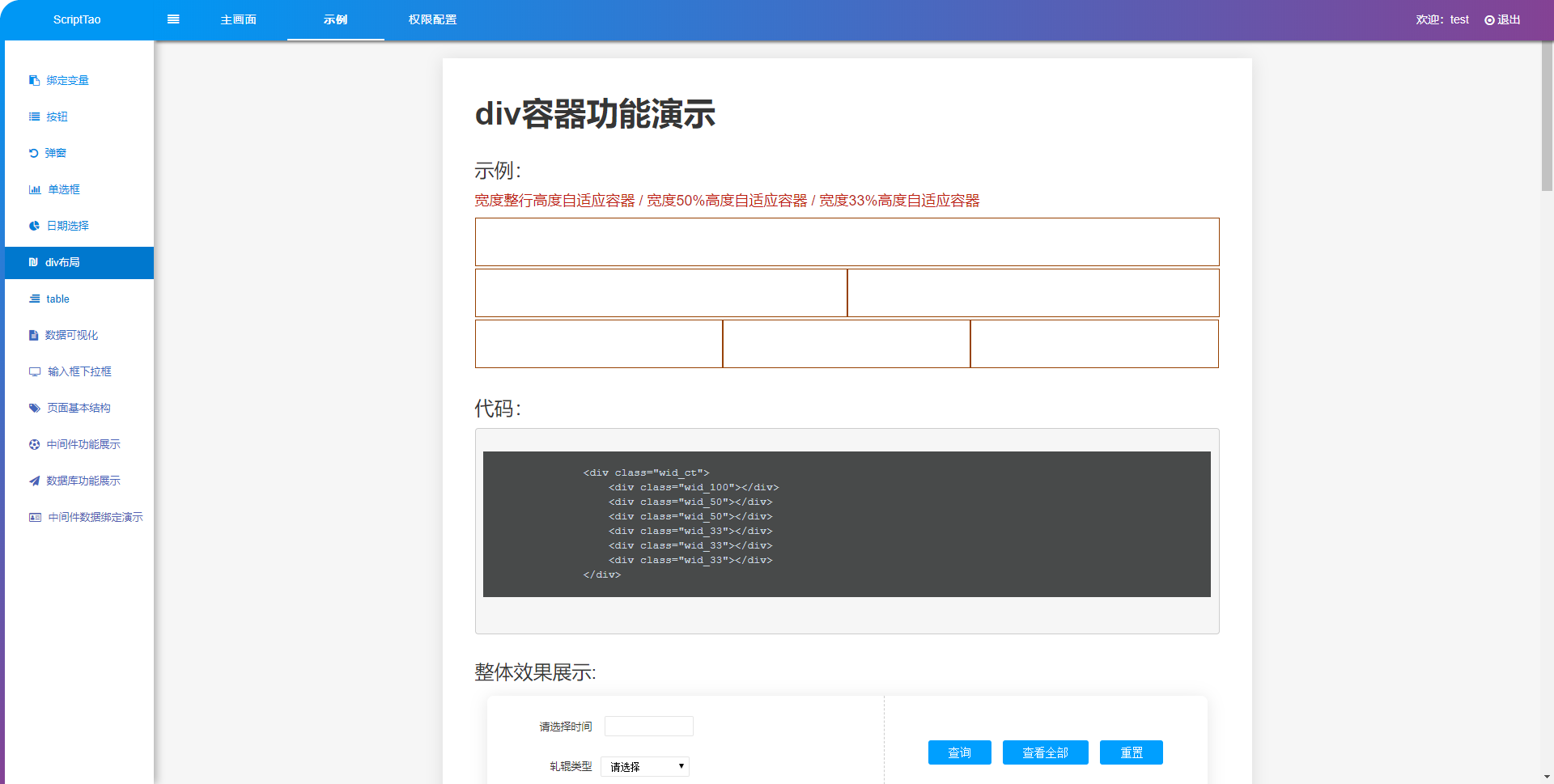
## 页面开发

（不同页面根据下面模块示例进行组装）

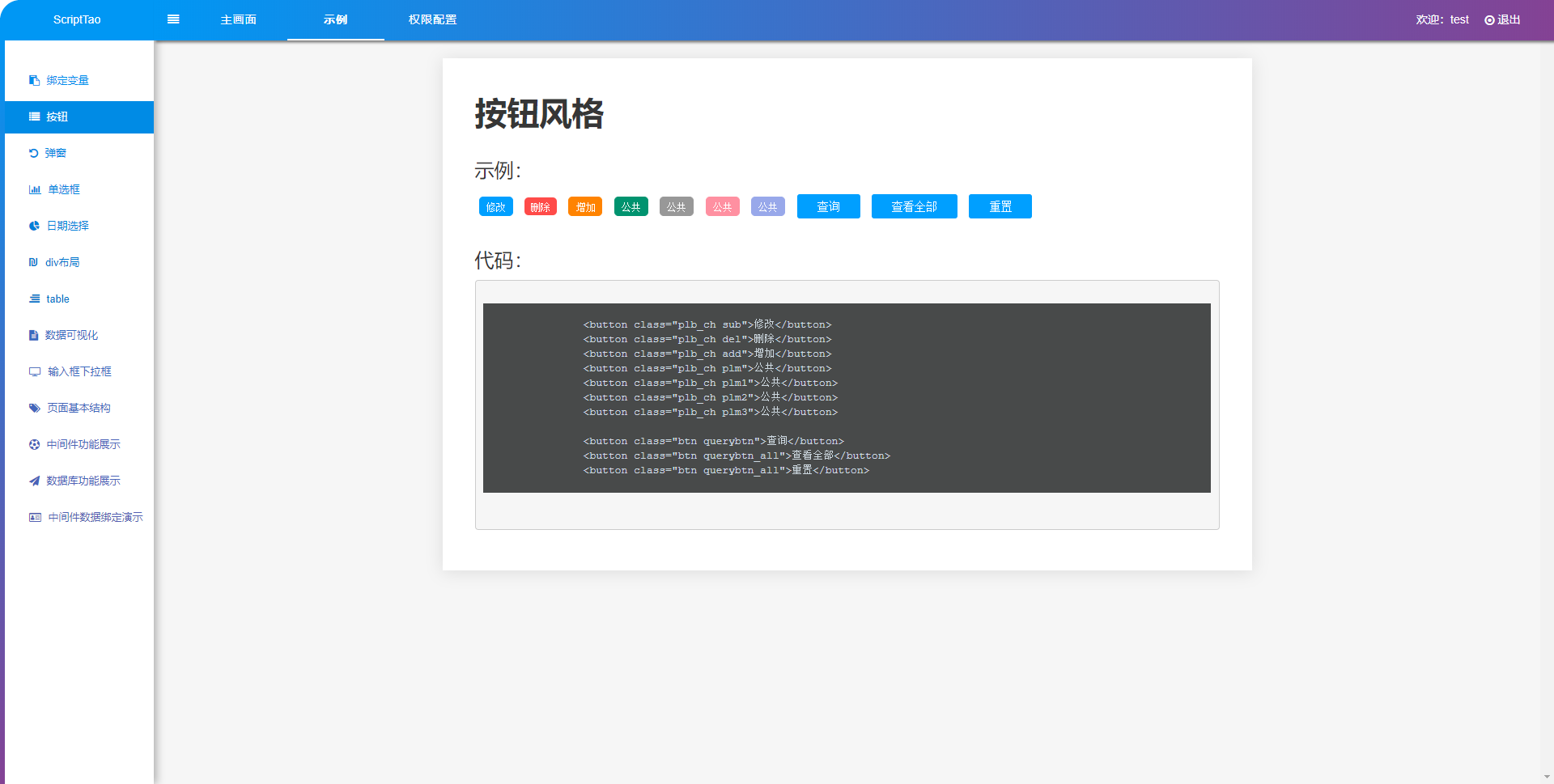
项目跑起来之后，使用test登陆 即可看到示例导航菜单（具体操作，代码页面中可查看）



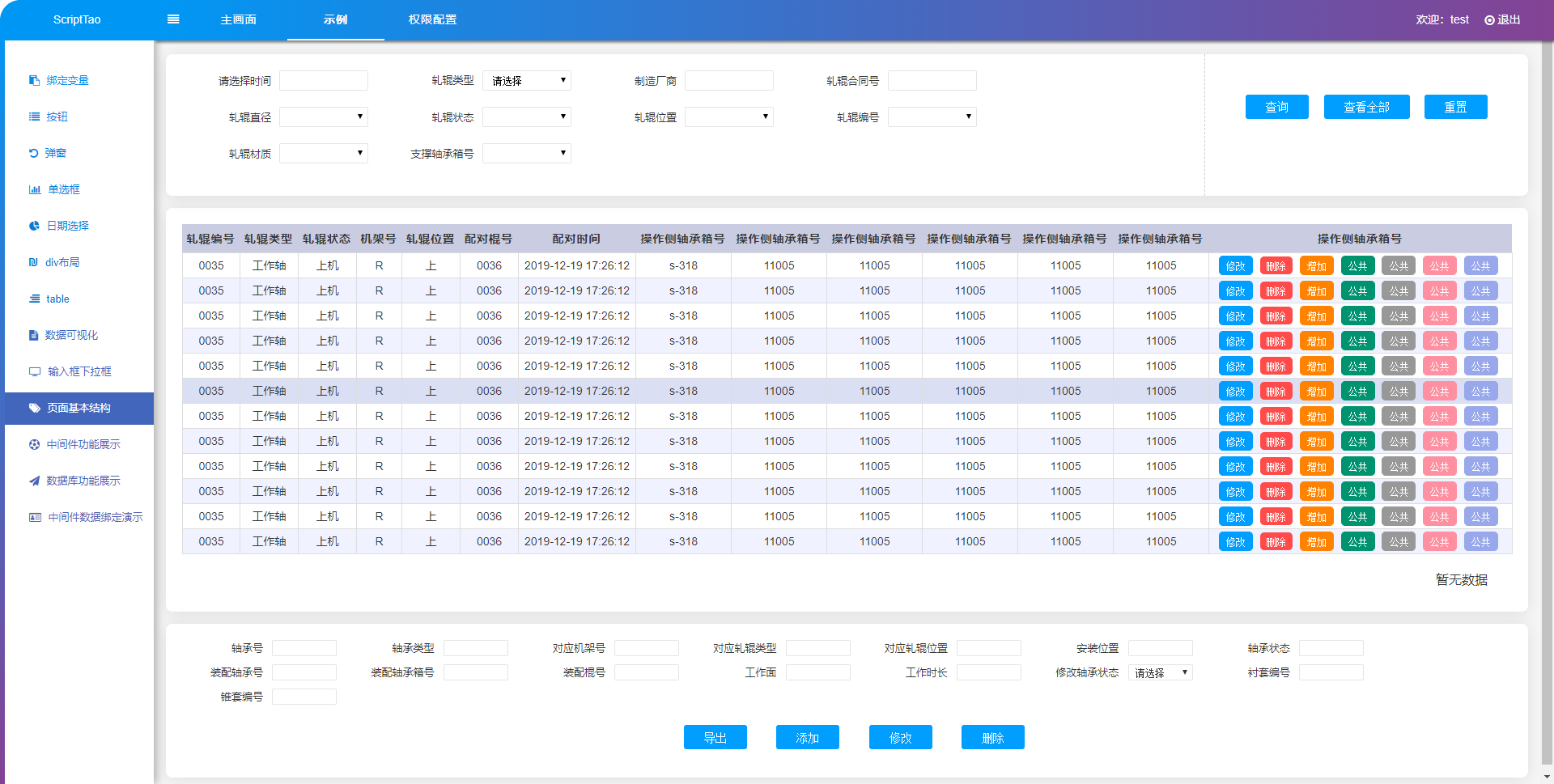
#### 1．div布局容器



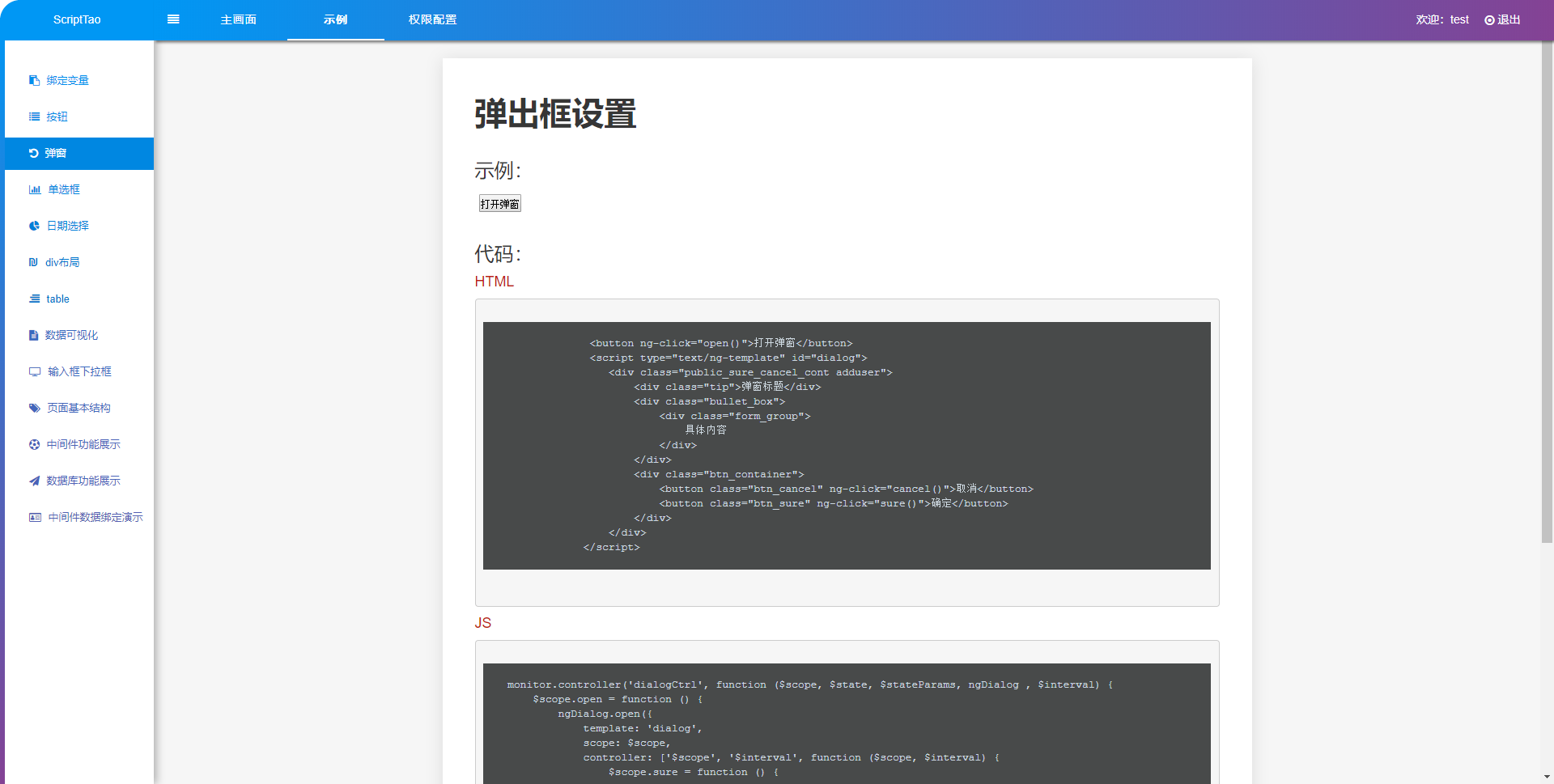
#### 2．按钮风格



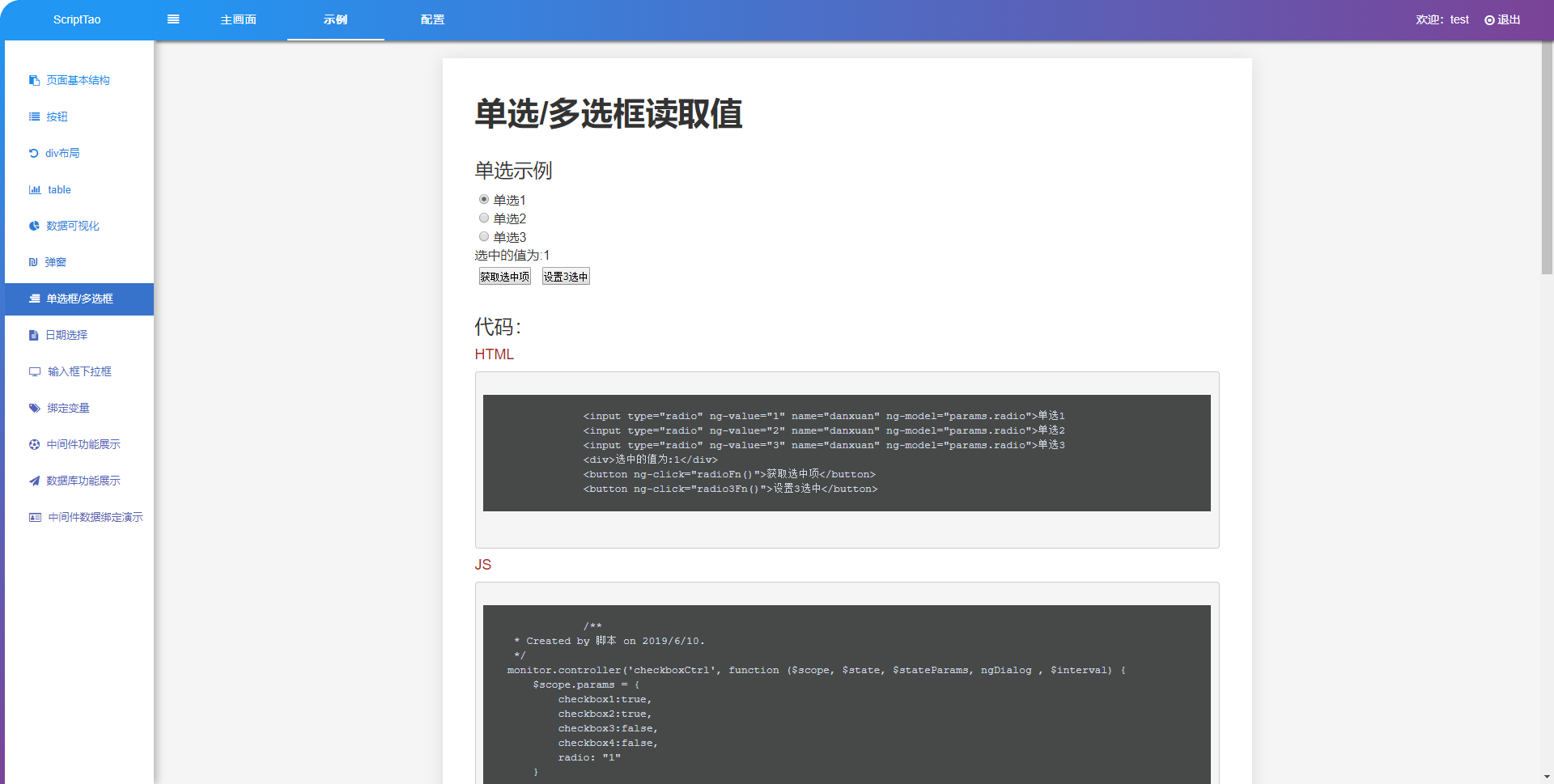
#### 3．页面基本布局结构



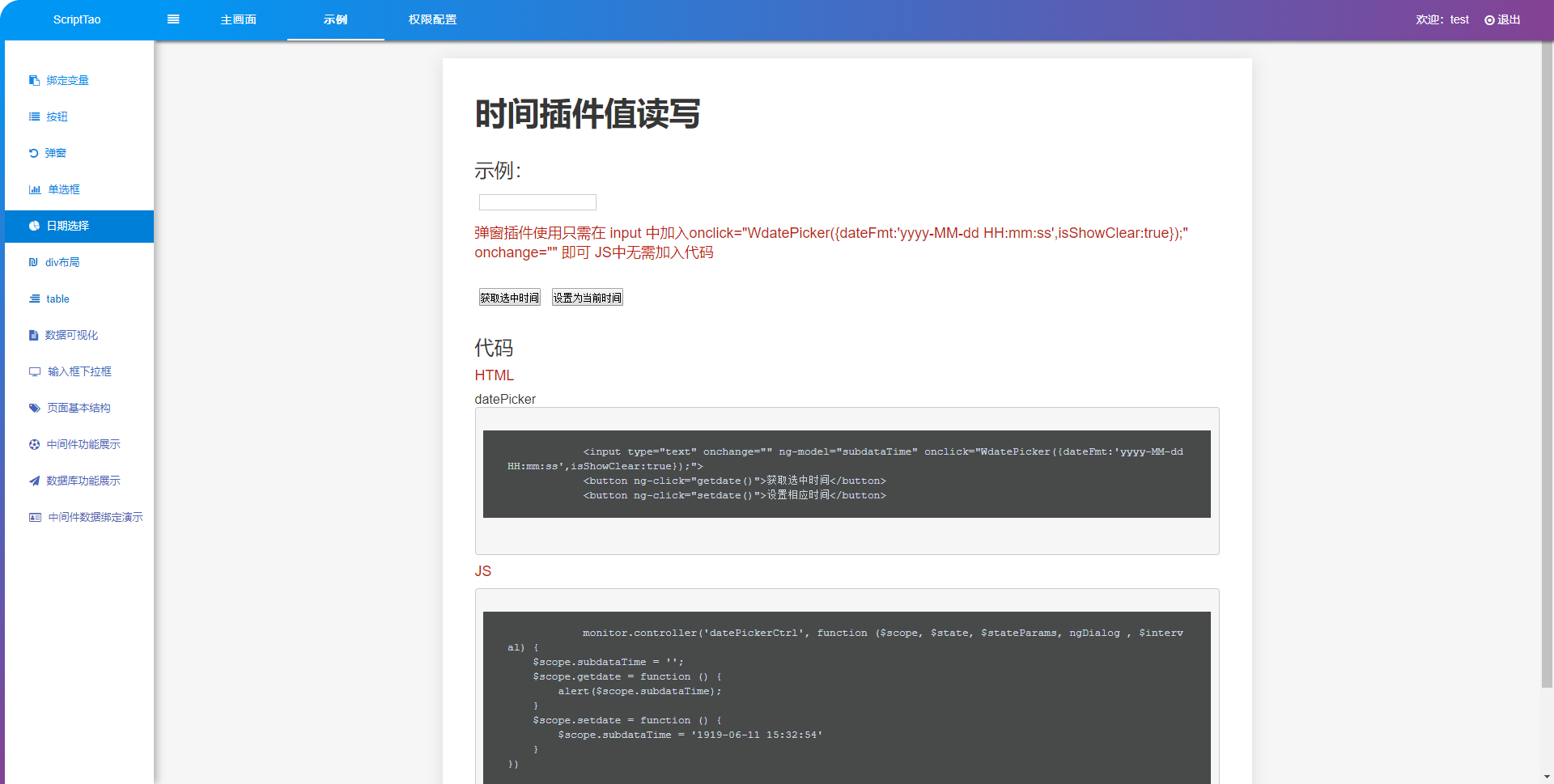
#### 4．弹窗设置



#### 5．单选/多选框框读写值



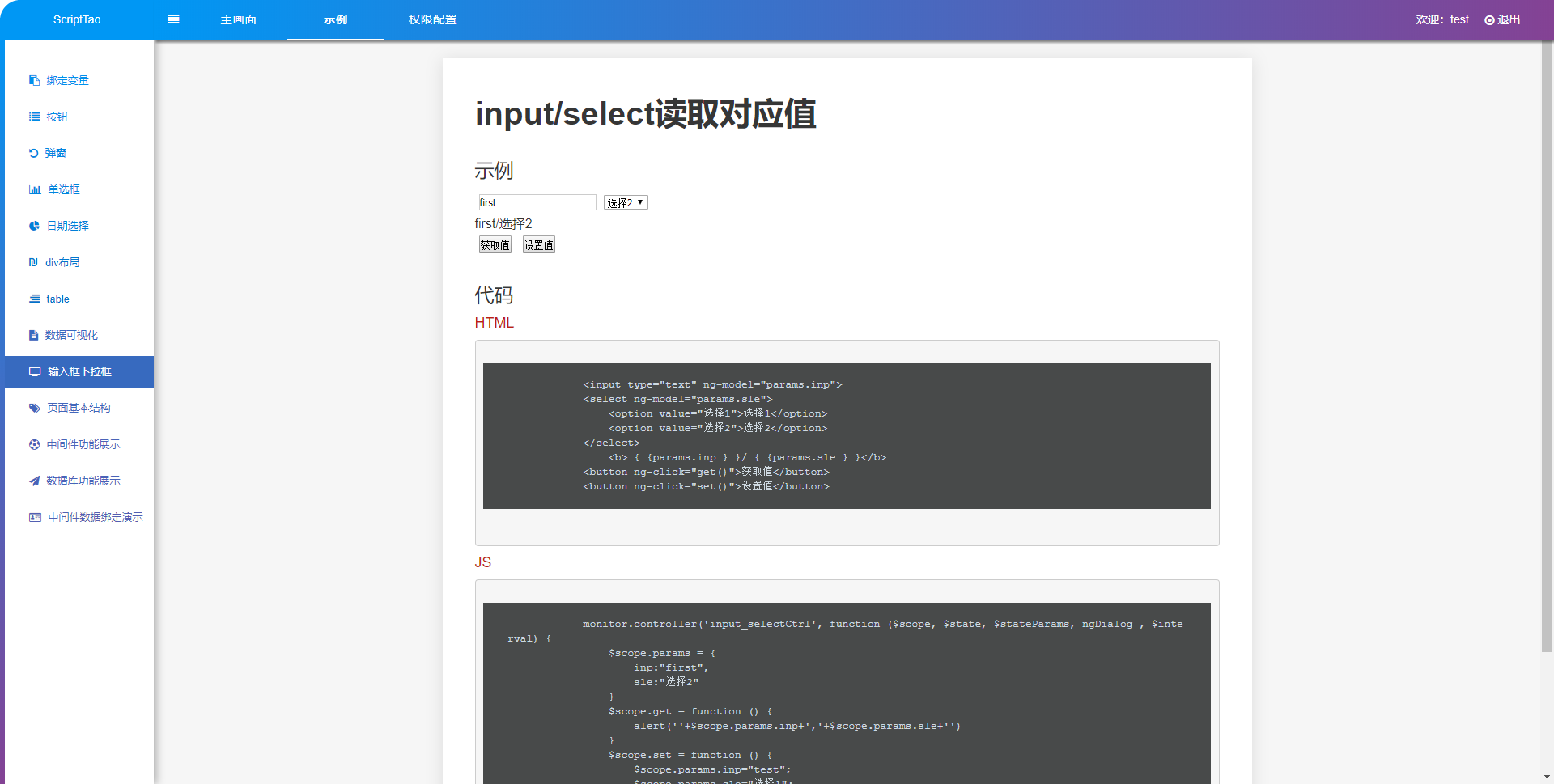
#### 6．日期插件使用



#### 7．数据可视化插件使用



#### 8．输入框下拉框读写值



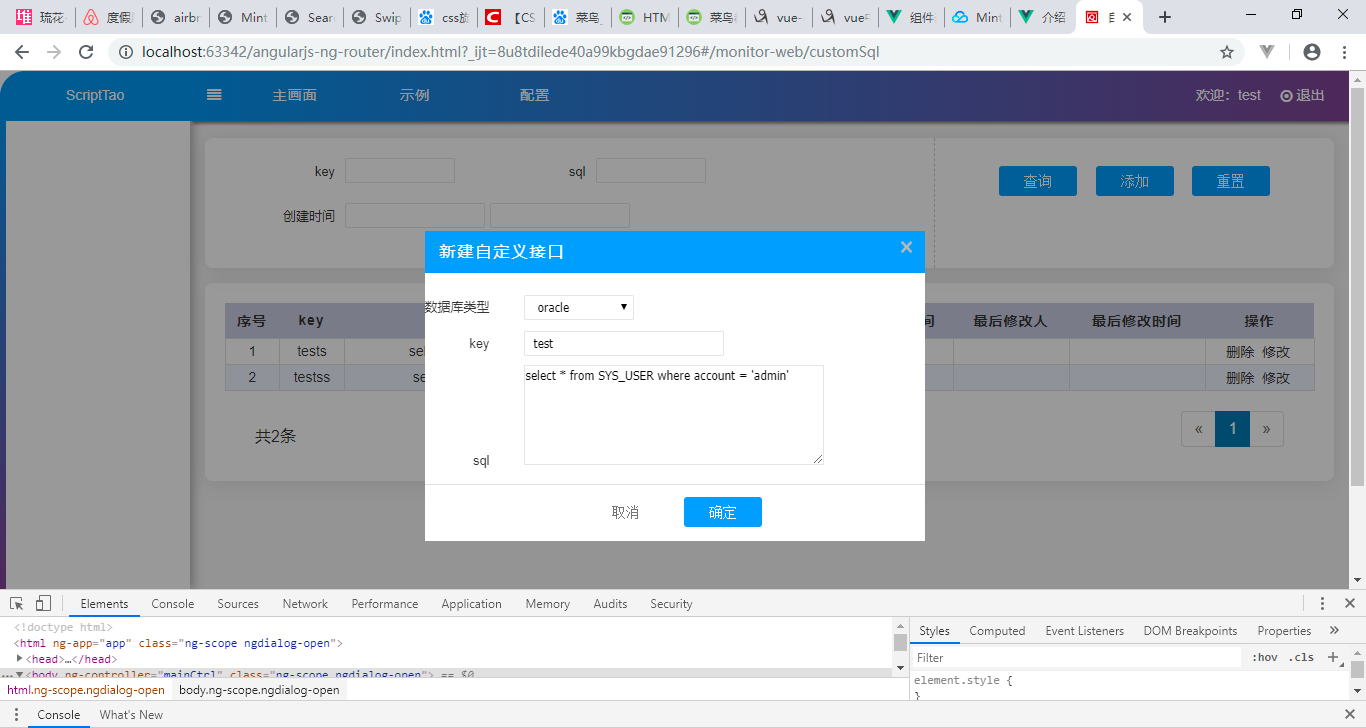
## HMI系统操作数据库

* 1. 在自定义sql页面（/monitor-web/customSql）新增sql语句:

可通过test账户登录查看菜单(配置/自定义sql配置)



首先，看到这样一个页面，点击添加键，在弹出框内键入如下信息



选择数据库类型(oracle/mysql/sqlservice)、

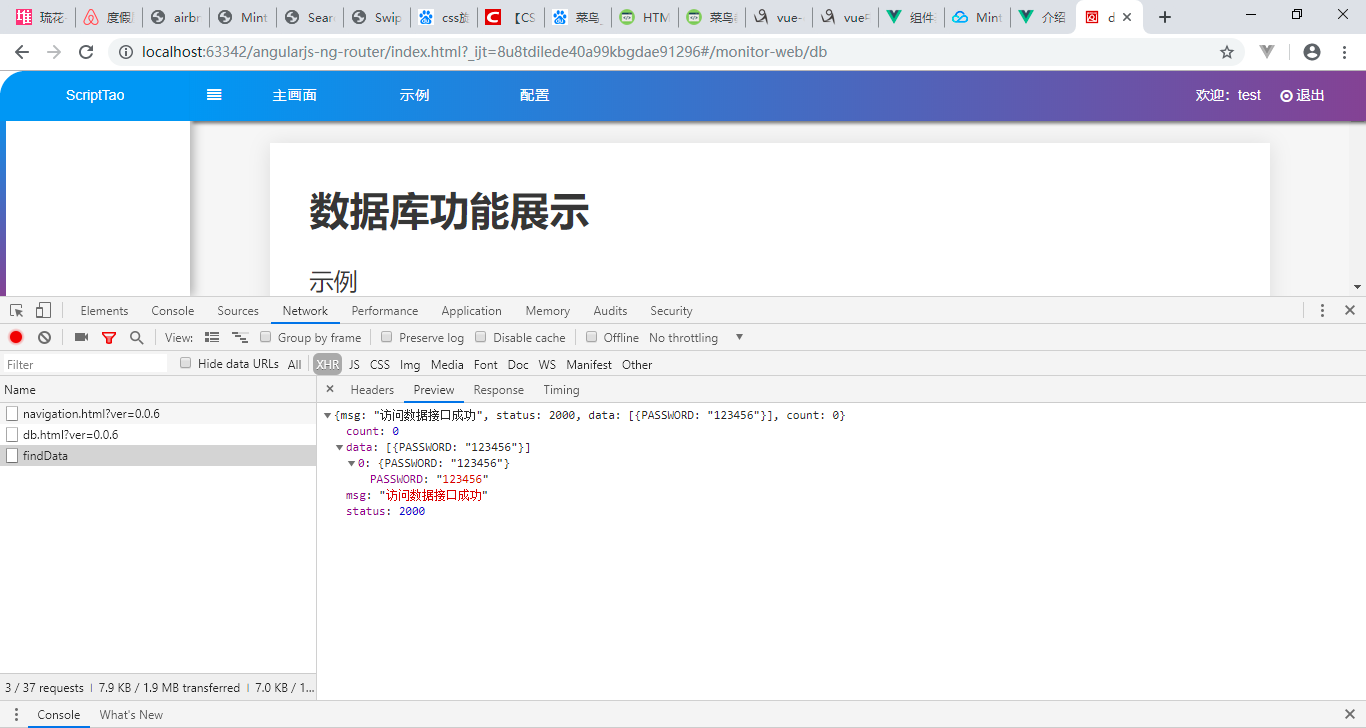
key(给自己的sql语句起个名字)、

sql(实际的sql语句)

之后在js文件里进行如下编写

commonFindData.save({ //调用数据库接口  
 **key**: "test"  
}, function (res) {  
 console.log(res.data[0].PASSWORD)//此处用于写想要实现的功能  
})

console.log()是将括号中的内容打印在控制台，这里作为实现功能的例子.

刷新页面进行测试：  


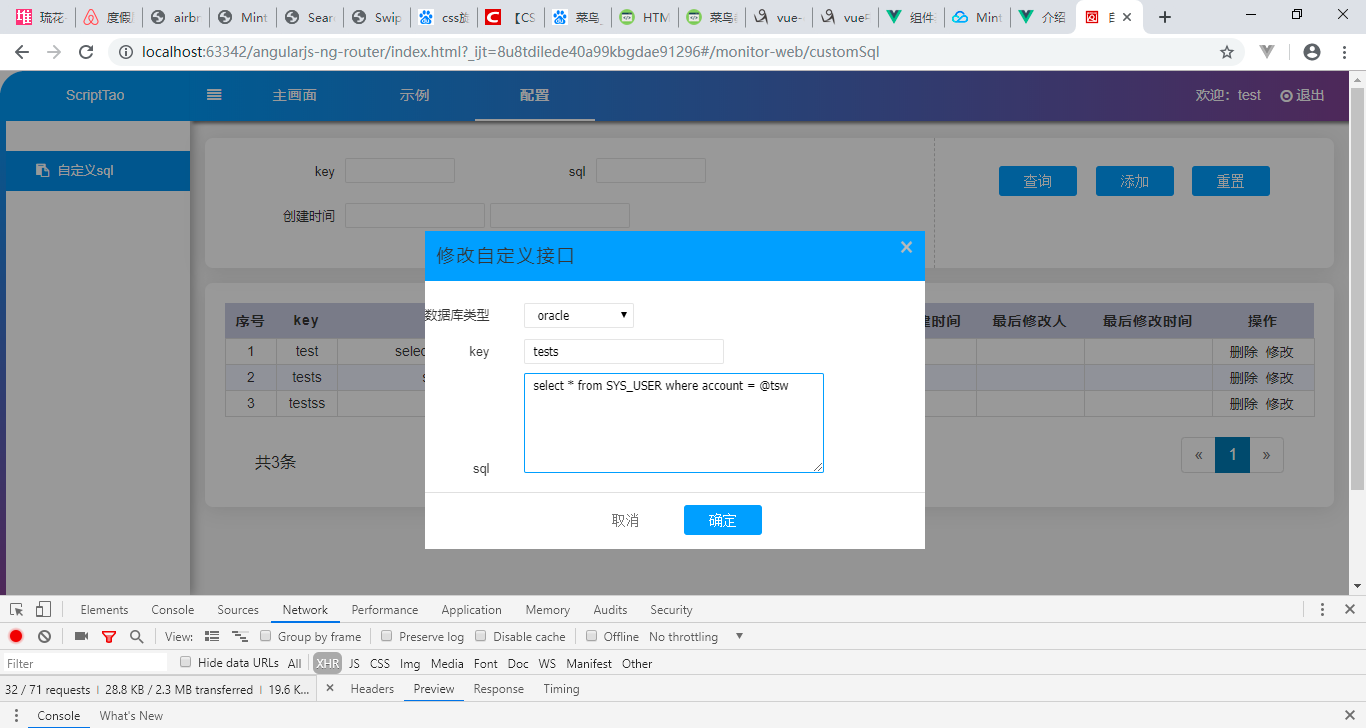
发现数据接口已经访问成功

并且想实现的控制台打印的功能也成功实现。

* 1. 约束条件可变的sql语句：

我们在多个页面中可能会引用同一个sql语句，但是希望他们有单独的约束条件。

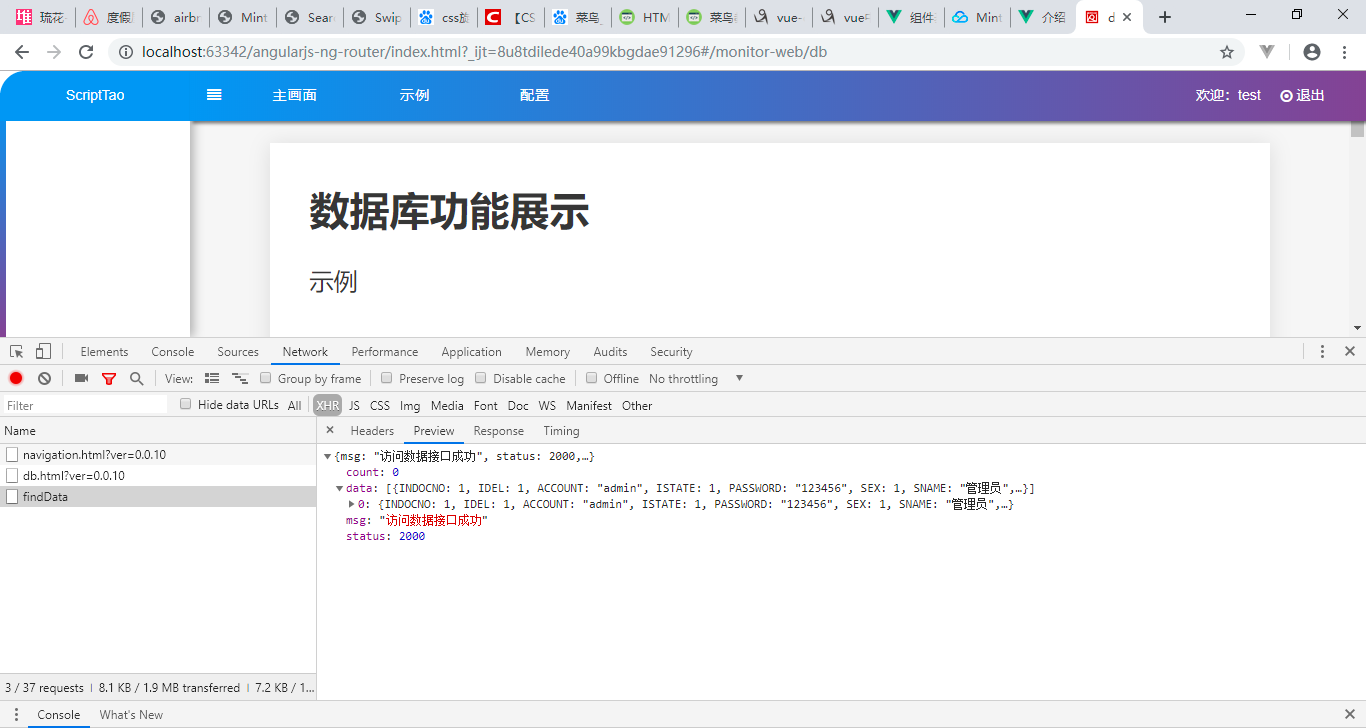
在这种情况下，要将约束条件写在相应页面的js中。



其中的@+字符串，相当于将约束条件视为一个变量，在js中对该变量进行相应的规定，此时js需要做如下改动：

commonFindData.save({   
 **key**: "tests",  
 **list**: [  
 {'@tsw': "\'admin\'"} //将'admin'绑定至@tsw  
 ]  
}, function (res) {  
 console.log(res.data[0].PASSWORD)  
})

在list里，你可以对sql的多个约束条件进行单独的定义，注意要用"\'admin\'"这种格式。



进行测试，成功。用这种方法，在不同页面的js中，就可以给同一条sql语句的约束条件赋予不同的值。

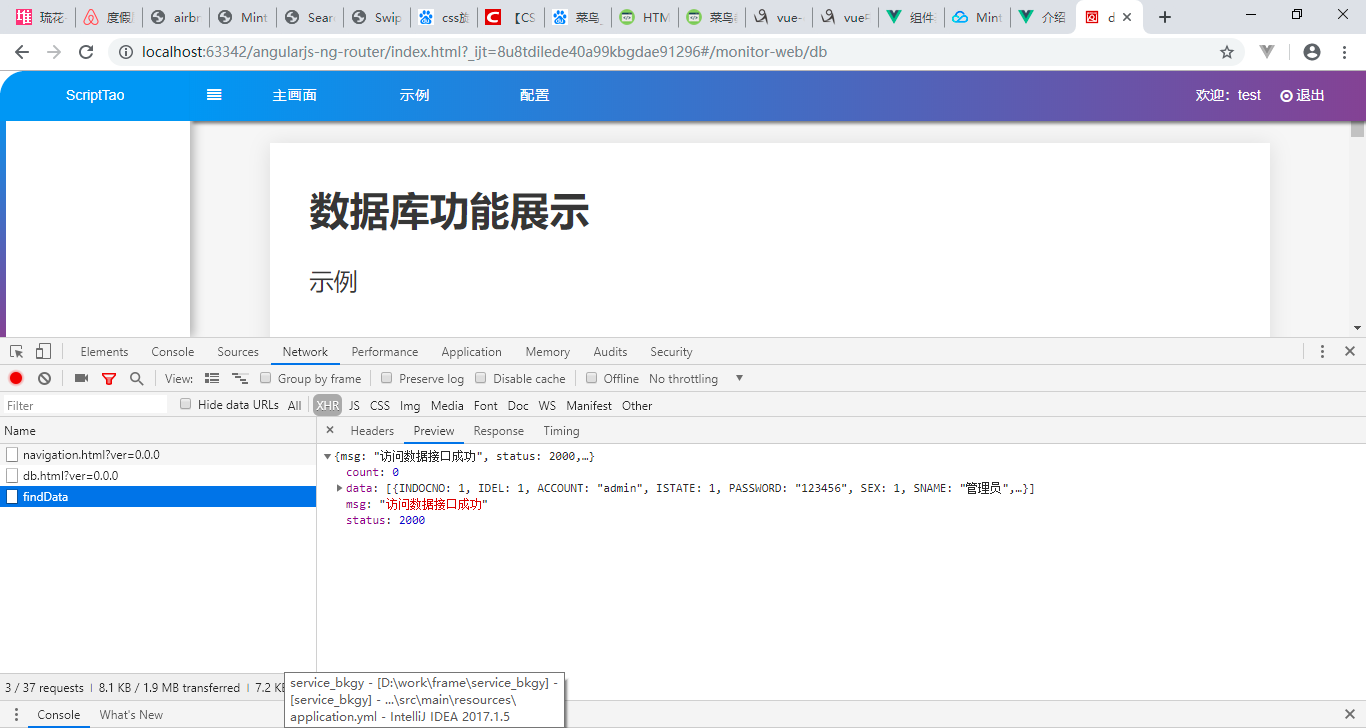
* 1. 将变量绑定至约束条件：

其中在自定义sql页面的操作和2.部分一样，在js里需要进行如下添加

$scope.**ss** = 'admin' //新定义一个变量ss，他的值为admin

commonFindData.save({   
 **key**: "tests",  
 **list**: [  
 {'@tsw': "'"+$scope.**ss**+"'"} //将ss绑定至@tsw  
 ]  
}, function (res) {  
 console.log(res.data[0].PASSWORD)  
})

进行测试，取数成功！



用这种方式，你可以给input等控件绑定变量，在前端输入改变变量，进而对sql语句的约束条件进行改变，以实现在前端页面查询的功能。

## HMI系统操作中间件

#### 页面双向绑定

<input type="text" data-tag="L3\_HEADID">

其中双击输入框可对中间值进行修改(有拦截控制)

#### 变量中心读取中间件变量

在config/config.js中维护项目所需所有中间件变量(每次新增/修改/删除 都需要重新登录)



使用：读取Cut\_Info,asd213变量

GetHmiParam.save({  
 **tag\_list**: "Cut\_Info,asd213"  
},function (res) {  
 console.log(res);  
})

#### 读取中间件变量：GetHmiData

GetHmiData.save({  
 **tags**: [{  
 **name**: "asd",  
 **ts**: "0"  
 }],  
 **msg**: []  
}, function (res) {

})

#### 4. 修改中间件变量：sendTag

var tags = [  
 {  
 "name": "RECStretchParamF1",  
 "value": "ccc"  
 },  
];  
  
sendTag(tags, function (response) {  
 **$**("#result").html(JSON.stringify(response, null, 4));  
});

#### 5. 发送消息：SendMessage

SendMessage.save({  
 "id": "msg",  
 "timeout": 3,  
 "reply": true,  
 "data": [  
 {  
 "x": "1",  
 "y": "2"  
 },  
 {  
 "x": "3",  
 "y": "4"  
 }  
 ]  
},function (res) {  
  
})

#### 6. 获取日志：GetLogData

var data = {  
 "startTime": "",  
 "endTime": "",  
 "keyword": "",  
 "count": 3  
};  
  
GetLogData.save(data, function (response) {  
 **$**("#result").html(JSON.stringify(response, null, 4));  
});

#### 7. 监听oldValue和newValue 回调函数

GetHmiData.save({  
 **tags**: [{  
 **name**: "asd",  
 **ts**: "0"  
 }],  
 **msg**: []  
}, function (res) {  
 HMICallBack(res,function (oldValue,newValue) {  
 /\*发生改变具体处理在此处理\*/  
 console.log(oldValue+","+newValue)  
 })  
})

## 基础知识补充

### 基础页面结构

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Document</title>  
</head>  
<body>  
  
</body>  
</html>

开发一个完整的前端页面需要html,css,js三种文件，html文件实现的页面的基本框架，css文件实现的是一个页面样式、风格，js文件只要实现页面的功能。

#### 基本标签语意

##### Link

<link> 标签定义了文档与外部资源之间的关系。<link> 标签通常用于链接到样式表:

<head>

<link rel="stylesheet" type="text/css" href="mystyle.css">

</head>

##### meta

meta标签描述了一些基本的元数据。<meta> 标签提供了元数据.元数据也不显示在页面上，但会被浏览器解析。META 元素通常用于指定网页的描述，关键词，文件的最后修改时间，作者，和其他元数据。元数据可以使用于浏览器（如何显示内容或重新加载面）搜索引擎（关键词），或其他Web服务。<meta> 一般放置于 <head> 区域

##### script

<script>标签用于加载脚本文件，如： JavaScript。<script> 元素

外部 JavaScript 文件的文件扩展名是 .js。如需使用外部文件，请在 <script> 标签的 "src" 属性中设置该 .js 文件：

<!DOCTYPE html> <html> <body> <script src="myScript.js"></script> </body> </html>

本项目不得在HTML中嵌入css与js语句，必须全部以外部引用的形式进行连接。项目中需要引用css样式文件和js文件都在index文件中引用完成，正常开发页面时无需重复引用。

以下是HTML和CSS的一些基本元素

#### 基本标签/样式总结

##### HTML

|  |  |  |
| --- | --- | --- |
| 基本类 | <!--…--> | 注释 |
| <html>…</html> | 定义 HTML 文档 |
| <head>…</head> | 文档的信息 |
| <meta> | HTML 文档的元信息 |
| <title>…</title> | 文档的标题 |
| <link> | 文档与外部资源的关系 |
| <body>…</body> | 可见的页面内容 |
| 格式类 | <div></div> | 定义文档中的节 |
| <h1>...</h1> | 标题字大小（h1~h6） |
| <b>...</b> | 粗体字 |
| <strong>...</strong> | 粗体字(强调) |
| <i>...</i> | 斜体字 |
| <em>...</em> | 斜体字(强调) |
| <ul>…</ul> | 无序列表 |
| <ol>…</ol> | 有序列表 |
| <li>…</li> | 列表项目 |
| <span></span> | 用于对文档中的行内元素进行组合 |
| <lable></lable> | 用于为 input 元素做出标记 |
| 图片/链接类 | <a href=”…”>…</a> | 超链接 |
| <img src=’”…”> | 定义图像 |
| 表格类 | <table>…</table> | 定义表格 |
| <tr>…</tr> | 定义表格中的行 |
| <th>…</th> | 定义表格中的表头单元格 |
| <td>…</td> | 定义表格中的单元 |
| 输入类 | <form></form> | 定义供用户输入的 HTML 表单 |
| <input type=”text”/> | 文本框 |
| <input type=”password”/> | 密码框 |
| 下拉列表 | <select></select> | 定义选择列表 |
| <option value=”值”></option> | 定义选择列表中的选项 |
| 按钮类 | <button></button> | 通用按钮 |
| <input type=”radio”/> | 单选按钮 |
| <input type=”checkbox”/> | 复选框 |
| <input type="submit"/> | 提交按钮 |
| <input type="reset"/> | 重置按钮 |

##### CSS

css样式统一使用外联式（在路由monitorRoute.js中引入）

基础代码:

|  |  |  |
| --- | --- | --- |
| 文本设置 | font-size： | 字号大小 |
|  | font-style： | 字体格式 |
|  | font-weight： | 字体粗细 |
|  | color: | 文本颜色 |
| 背景 | background-color: | 设置背景色 |
|  | background-image: url(URL) | 背景图片（URL就是背景图片的存放路径，none表示无） |
|  | background-repeat: 参数 | 背景图片重复，参数取值范围 ： no-repeat：不重复平铺背景图片 repeat-x：使图片只在水平方向上平铺 repeat-y：使图片只在垂直方向上平铺 如果不指定背景图片重复属性，浏览器默认的是背景图片向水平和垂直两个方向平铺。 |
| 文本对齐 | text-align: 参数 | 参数的取值： left：左对齐 right：右对齐 center：居中对齐 justify：相对左右两端对齐 |
| 垂直对齐 | vertical-align: 参数 | 参数的取值：  top：顶对齐 bottom：底对齐 text-top：相对文本顶对齐 text-bottom：相对文本底对齐 baseline：基准线对齐 middle：中心对齐 sub：以下标的形式显示 super：以上标的形式显示 |
| 文本缩进 | text-indent: 缩进距离 | 12px相当于一个文字距离 |
| 空格 | white-space: 参数 | 参数取值范围：  normal 默认，空白会被浏览器忽略  pre 保留空白  nowrap 文本不换行 |
| 显示样式 | display: 参数 | 参数取值范围：  block：块级元素，在对象前后都换行  inline：在对象前后都不换行  list-item：在对象前后都换行，增加了项目符号  none：无显示 |
| 方框 | height | 高度 |
|  | width | 宽度 |
|  | padding | 内边距 |
|  | margin | 外边距 |
|  | float | 可以让块级元素在一行中排列，例如横向菜单。 |
| 边框 | border:border-width border-style color | border-style的参数： none：无边框  dotted：边框为点线 dashed：边框为长短线 solid：边框为实线 double：边框为双线 |
| 定位 | position:参数 | 参数范围absolute | relative | static |
| 透明度 | visibility:参数 | 参数范围inherit | visible | hidden; |

### angular.js基础指令

#### ng-model

用于绑定 HTML 控制器的值到应用数据<input>, <select>, <textarea>, 元素支持该指令。

首先在页面需要取值的input，select，textarea元素中加入ng-model如

<**input ng-model=**"parm.demo">

如果想在js部分获取到这个input的值需要在js开头先定义一个对象，将ng-model里的内容在这个对象里声明如

$scope.*parm* = {  
 *demo*:""  
}

这个对象可以声明你需要在这个页面用到任何ng-model的内容名，当然html与js部分必须保持一致，在js中声明之后就可以直接进行赋值或者取值操作如

$scope.getData = **function**(){  
 $scope.teatOne = $scope.parm.demo;  
}

这里我做了一个按钮可以更直观的理解，当点击按钮时input的值就会赋值给teatOne,如果想重新赋值给页面另一个元素可以直接引用如

<span>{{demo1}}</span>

这两个{}作用是双向数据绑定，如果想赋值给另一个input直接将这个input的ng-model的内容改为teatOne就可以如

<input ng-model="teatOne">

特别注意：在js部分调用所有的方法名，变量名都必须在$scope的子集中，也就是说必须是以$scope.+名的形式

#### ng-repeat

ng-repeat主要用数据的遍历操作，如生成表格，生成菜单，下拉生成等操作，这里主要以生成表格为列(对于我们项目而言大都是用于生成表格)如

遍历渲染到html中

$scope.**list** = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]

<table>  
 <thead>  
 <tr>  
 <th>轧辊编号</th>  
 <th>轧辊类型</th>  
 <th>轧辊状态</th>  
 <th>机架号</th>  
 <th>轧辊位置</th>  
 <th>配对棍号</th>  
 <th>配对时间</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 <th>操作侧轴承箱号</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr ng-repeat="item in list">  
 <td>0035</td>  
 <td>工作轴</td>  
 <td>上机</td>  
 <td>R</td>  
 <td>上</td>  
 <td>0036</td>  
 <td>2019-12-19 17:26:12</td>  
 <td>s-318</td>  
 <td>11005</td>  
 <td>11005</td>  
 <td>11005</td>  
 <td>11005</td>  
 <td>11005</td>  
 <td>11005</td>  
 </tr>  
 </tbody>  
</table>

这里的list是我写的假数据，实际应用中数据都来自数据库或者中间件，我们一般将渠道的数据存入list然后进行遍历操作。

#### ng-class

通过js动态操作样式

变量$scope.param.suerNameFlag 等于true时用username\_check 这个class类名，等于false时用input\_border 这个class类名



这个主要用于判断中间件或者数据库的值的变化从而改变页面某个元素的样式，具体用法就如上所述在元素中如div，加上ng-class

#### ng-click

点击事件 与onclick=“clickFn()”的用法相同，区别在于在js部分的引用必须是以angular.js的形式如

$scope.getData = **function**(){  
 $scope.teatOne = $scope.parm.demo;  
}

#### ng-change

内容改变事件,语法同上

#### ng-if

控制html元素显示隐藏 true显示 false隐藏

变量$scope.chartFlag = true 显示 false隐藏



详细文档查阅 http://www.angularjs.net.cn/

### 插件的使用

#### 分页插件使用

在文件table文件夹中有示例可查看

1. 在html文件中

<tm-pagination conf="paginationConf"></tm-pagination>

1. Js中

$scope.**paginationConf** = {  
 **currentPage**: 1,//当前页  
 **totalItems**: 0,//allItem数  
 **itemsPerPage**: 10,//一页多少个  
 **pagesLength**: 10,//页码长度  
 **perPageOptions**: [10, 20, 30, 40, 50],  
 onChange: function () {  
 $scope.queryList();  
 }  
};

注意事项：

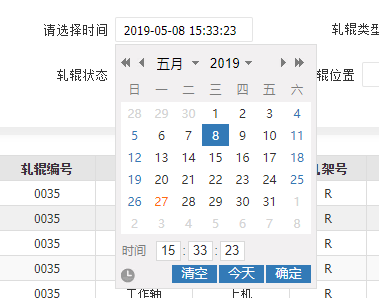
/\*  
\* 1.此分页为实时交互数据,非一次性拿到所有数据,前端做分页  
\* 2.其中在onchange回调函数中 做数据库查询  
\* 3.onchange在第一次加载页面时就会被调用  
\* 4.$scope.queryList 查询方法中需要加上当前页数,一页多少个 传到后端  
\* 5.拿到后端返回值 需要把总共多少条数 再赋值给$scope.paginationConf.totalItems  
\* \*/

#### 时间插件

在项目中table文件夹下可查看具体使用示例

<input type="text" onchange="" ng-model="subdataTime" onclick="WdatePicker({**dateFmt**:'yyyy-MM-dd HH:mm:ss',**isShowClear**:true});">

页面效果如下图：



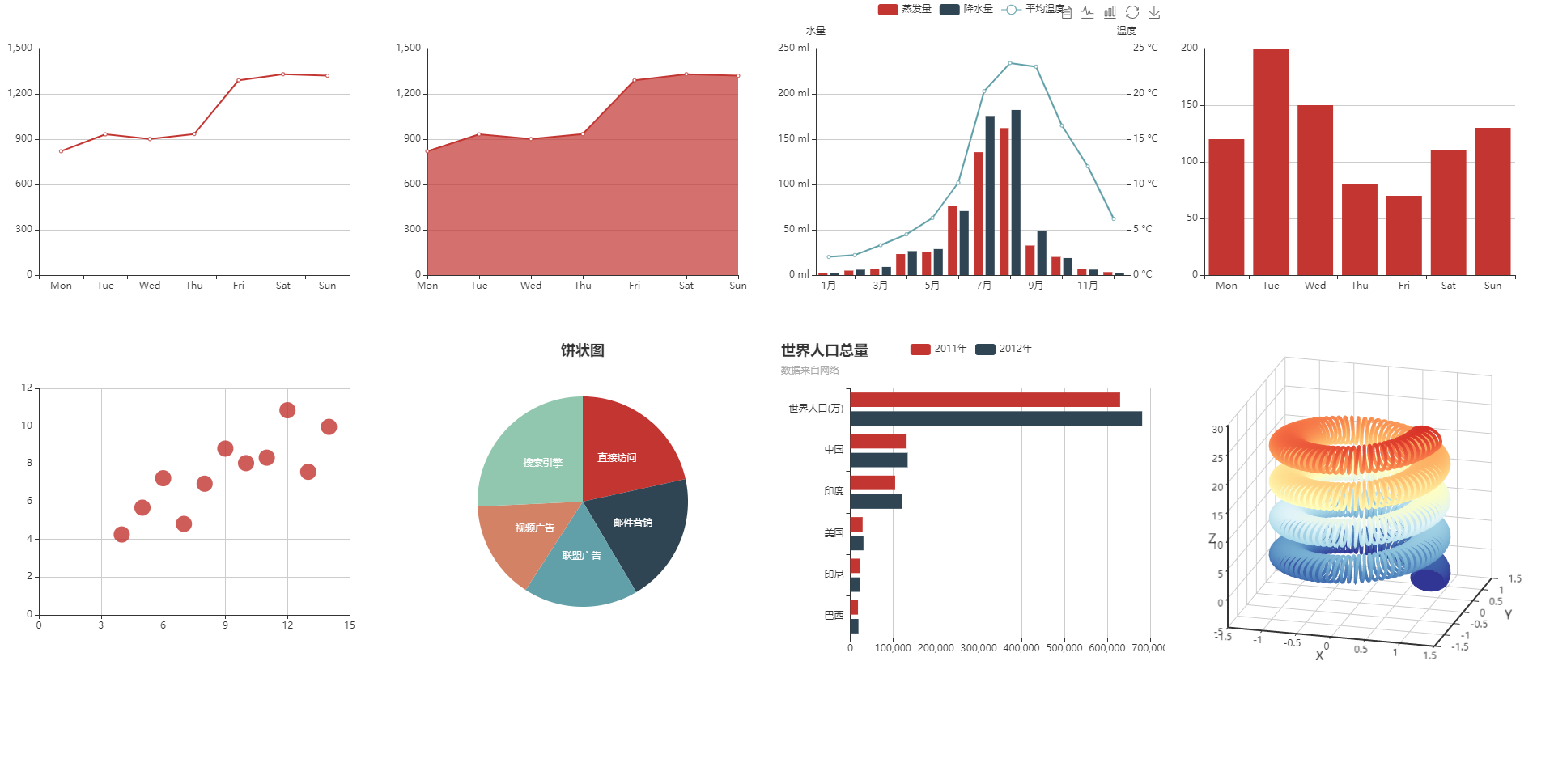
Js中取值方法：

console.log($scope.subdataTime); // 2019-05-08 15:33:23

#### 基本画图功能

Echart是H5的一种插件，可以用来绘制折线图、柱状图、饼状图等等一系列数据展示图。

示例请查看demo/echart文件夹



echart更多的配置请移步<https://echarts.baidu.com/option.html#title>

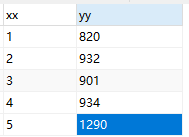
#### 获取数据库/中间件中的数据并制图

**拿到数据库/中间件的值组装成echart图能够使用的数据格式数据,此处可变性较大可参考demo/echart文件夹或者查询官网示例**

**举例:**

遍历数据库，并将数据库中读出的数据分别写进数组中，用这些数组制图。

在测试数据库中增加测试用字段xx,yy。

，输入一些测试用数据，xx为横坐标，yy为纵坐标。

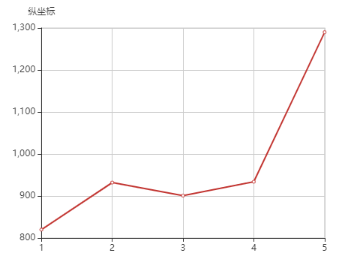
在HTML中的**body\_box**中设置容器：

<div id="basicLine" class="basic"></div>

在js中键入如下设置，其中//后面的内容是备注，在实际网页中不发挥作用，供学习使用。

**var** testarray=[]*//定义制图用数组*commonFindData.save({*//调用接口* **key**: **"test"***//key在权限管理中配置，对应一个sql语句*}, **function** (response) {*//接口返还* **for** (**var** i = 0; i < response.**data**.**length**; i++){*//定义取数循环，用push将数据库中的数加进数组里* testarray.push([***Number***(response.**data**[i].xx),***Number***(response.**data**[i].**yy**)])*//Number是将字符串类型转换为数字* }*//xx为数据库中作为横坐标的字段名，yy为数据库中作为纵坐标的字段名* **var** option = {  
 **grid**:{**show**:**true**},*//网格线是否显示，是为true，否为false* **xAxis**: {  
 **scale**:**true***//坐标最小值是否从数组中横坐标的最小值开始* },  
 **yAxis**: {  
 **name**: **'纵坐标'**,*//纵坐标的名称，可以显示在坐标轴顶端* **scale**:**true**,*//坐标最小值是否从数组中纵坐标的最小值开始* },  
 **series**: [{  
 **data**: testarray,*//制图用数据，即前面定义的数组* **type**: **'line'**,*//图类型，我们示例用折线图* }]  
 }  
 **var** myChart = **echarts**.init(***document***.getElementById(**'**basicLine**'**));*//将配置注入到html中定义的容器* myChart.setOption(option);  
})

进行测试：



#### 格式化时间格式:

|date:'yyyy-MM-dd HH:mm:ss'

<td>{{item.createtime |**date**:'yyyy-MM-dd HH:mm:ss'}}</td>