**QsidaEIS开发总结**

历时2个多月的时间，QisdaEIS APP开发的主要阶段已经结束，目前Staging版本测试OK，Production也准备上线。从最初的实现UI，到绑定数据，最后到测试修改，认识到了自身的一些问题，也从中吸取到了经验。在这里总结一下开发过程当中遇到的一些问题，为了下次开发做好更充分的准备。

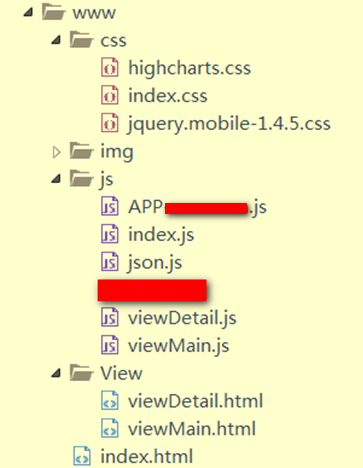
# UI原型

## Cordova

UI部分主要参考2个文件：一是UIFlow.pdf，静态原型；二是PES，需求和功能描述，以及DB的字段和获取。首先要实现静态原型，这部分属于前端开发当中的结构层和表示层，即html + css，这也是最基础的部分。QPlay APP开发使用的是cordova + jquery mobile的方式，cordova的优点是跨平台和可以调用设备相关API。

### 目录结构

我们利用cordova创建项目的时候会自动生成一个www的文件夹，www文件夹下有一个index.html，这个就是程式入口，我们不能修改这个文件的名字，还要将body里面的代码删除。因为index.html只是程式入口，实际并没有需要显示的内容，所以要将body里面自带的内容删除，并保留<script type="text/javascript" src="cordova.js"></script>，然后按照自己的需求引入其他css和js文件即可。接着我们需要手动在www文件夹下新建4个文件夹，分别是css、js、img、View，img用来放图片，View用来放page，APP有几个页面我们就创建对应数量的page，并且将第一个页面命名为viewMain.html，将所有page放入pageList = [‘viewMain’, ‘viewDetail’]，然后每个页面都有对应的viewMain.js和viewDetail.js。我们所有的代码都放在www文件夹下，以下是www的目录结构。



index.css是我们自定义的样式表，APP.js是所有QPlay APP都有的并且是我们自己封装的公函，里面包括检查网络连接状况，呼叫API，记录登录名等方法。index.js对应index.html，是程式入口。

## jQuery Mobile

jquery mobile是基于jquery的移动UI框架，所以在移动开发中使用jquery mobile是比较适合的。

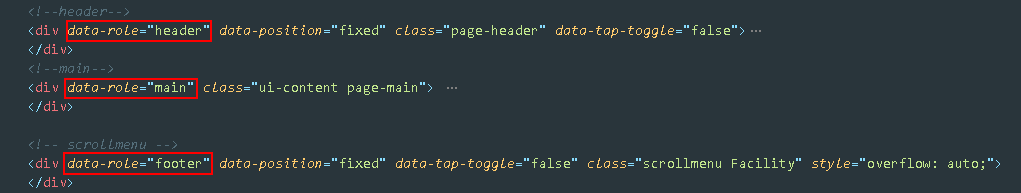
### View

在单页面开发中，有view视图这一说法，正好jqm也提供了一个方法来实现。之前描述过在www下有一个View文件夹，是专门用来放page的，每个page都有对应的html，因为是SPA单页面，所以在每个页面切换时并不能通过a标签进行页面之间的跳转，所以我们是通过jqm提供的方法$.mobile.changePage("#viewMain")来切换page。并且在每个page的html中并不需要head、body等标签。直接使用div进行布局，并且在最外层的div当中设置id属性值，对应各自的page，这就是view视图的概念。



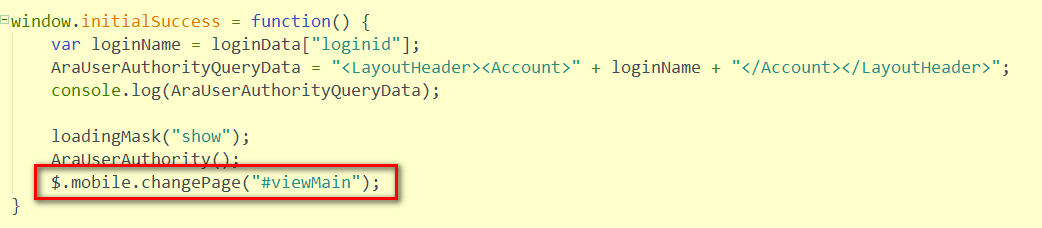
### 框架结构

QisdaEIS使用最多的是jqm的整体框架结构，比如data-role = “header”，data-role = “main”等将页面分成3部分。



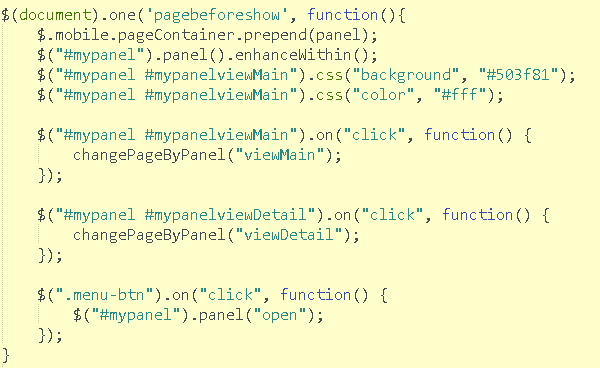
### 页面事件

Index.js它是我们页面的入口，除了声明一些全局变量和函数之外，它还有3个不可缺少的部分，第一是initialSuccess：function，即页面加载成功后进入第一个页面(view)，使用的是$.mobile.changePage( )方法。

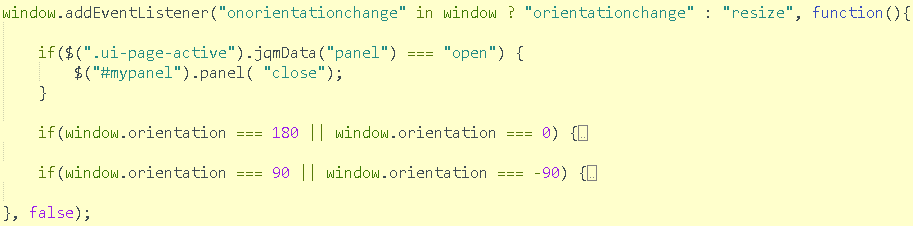


第二是pageBeforshow事件，该事件的主要作用是在页面显示之前将panel添加到页面当中，使用方法是$.mobile.pageContainer.prepend()。

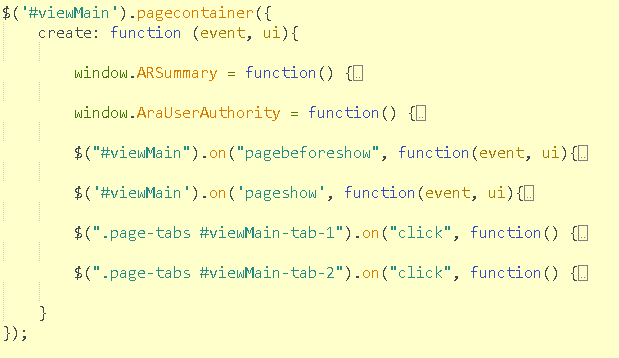
然后通过panel菜单切换页面changePageByPanel( )等一系列操作。



第三是横竖屏切换事件及监听，利用window.orientation来判断手机旋转角度。



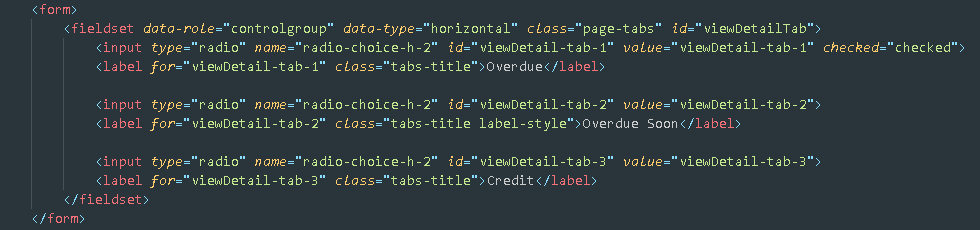
我们每个page都有自己的js文件，它的作用是对本页面进行操作。



除了一些全局变量和函数可以放在pagecontainer()外，其他对本页面进行操作的部分都要写在pagecontainer()内，主要功能是调用API，在pagebeforeshow的时候监听pull refresh事件，在pageshow的时候将数据导入图表或页面，还有切换tab标签时修改样式颜色并setData( )不同数据。

### 组件

实际上jqm自带组件使用并不多，因为团队已经封装了一些常用的组件，比如导航栏我们使用的实际上是一个单选框。



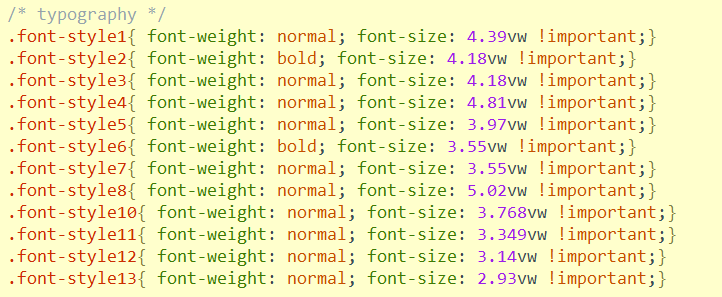
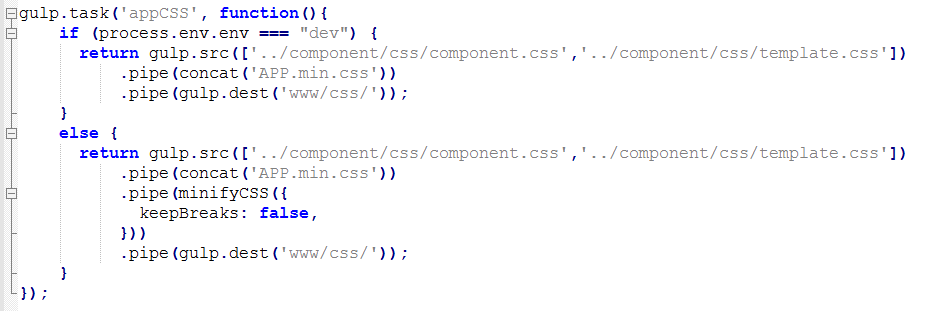
如上图，我们是将input = “radio”单选按钮作为tab标签， <fieldset>是将所有单选按钮分组，data-type = “horizonral”是水平排列，<label for = “id”>是单选文本并可点击选择。这种方式相较于jqm自带组件更加灵活，也便于控制。

## CSS

### 单位

UI部分样式有两点需要注意，一个是我们容器宽高单位和字体大小单位使用的是vw和vh，而不是常用的px，字母“w”代表width，“h”代表height，例如width: 100vw就等同于width: 100%，height: 40vh等同于height: 40%。这样就做到了页面的自适应，可以配适任何尺寸的移动设备。

### 公共css

另外一个是我们封装了一些公共的css样式，包括字体大小，popup，button等元素。文件路径是在EnterpriseAPPPlatform\APP\component\css文件夹下的template.css和component.css，但我们并不需要把这些css文件拷贝到项目文件夹下，我们所有QPlay下的APP都是通过gulp前端构建工具帮助我们完成公共文件的合并，包括app.js，app.css，app.html，还有一些多语言文件等，gulp还可以对js和css文件进行压缩，功能十分强大。这么做的目的是减少服务器响应时间，减少资源占用，提高性能。

我们只需要在index.html里面引入<link rel=”stylesheet” type=”text/css” href=”css/APP.min.css”>即可。

## Highcharts

### 图表配置

UI部分Hichcharts占了很大一部分比重，QisdaEIS总共有2个page，每个page都有2个图表，并且第一页有且只有2个图表，所以说图表十分重要，也是花费时间修改较多的地方。Highcharts作为第三方图表库，使用起来十分方便。在引入所需的highcharts.js文件后，首先配置图表内容，然后给图表一个div容器，接着对容器设置id属性，并且设置容器的宽和高，最后通过id对相应容器进行SVG图表绘制。

第一步：配置图表。其实图表就是一个js对象，它包含很多属性。常用的属性有chart(图表配置)、title(标题)、tooltip(提示框)、xAxis(x轴)、yAxis(y轴)、series(数据列)等。



这样我们就配置好了一个简单的图表，如何把它放到页面上呢。

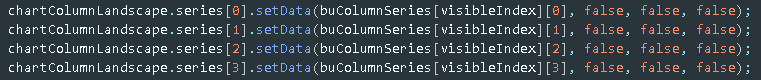
第二步：创建带有id属性和宽高的容器，<div id=”areaChart” style=”width: 200px; height: 100px”></div>

第三步：把图表放入div容器中，var chart1 = new Highcharts.Chart(“areaChart”, areaOption)，经过这3步我们就完成了一个简易的图表配置及渲染。

### API

Highcharts的API较多，我这里就着重讲几点，首先是通过构造函数：var areaChart = new Highcharts.Chart(id, option)，这样的优点是一个图表可以应用到不同的容器当中，只需要重新声明一个chart（变量），并绑定对应id即可。如果我们需要使用相同类型的图表，虽然他们之间可能有一些差别，但绝大多数代码都是一样。这种情况下我们就可以使用update( )方法。比如我们有一个已经配置好的图表A，我们准备把它放到一个< div id=”div1” style=”width: 200px; height: 100px”>的容器当中，var chart1 = new Highcharts.Chart(‘div1’, A)。然后我们还有一个<div id=”div2” style=”width: 200px; height: 100px”>的容器，但是它需要的title是和图表A不同的，其他代码都一样。那我们可以var chart2 = new Highcharts.Chart(‘div2’, A)。然后使用update( )方法来改变title即可，chart2.update({ title: ’……’ })，而且它并不会影响div1当中的图表。所以配置过后的图表，我们都可以通过update( )方法来改变图表的属性值，这样我们就不需要配置大量的图表对象option。

Highcharts还提供一些方法都是比较常用的，比如setData( )来设置图表series的值，addSeries( )可以添加新的数据列，redraw( )用来重绘图表，reflow( )让图表自适应图表容器等等。因为我们图表的数据可能会是动态添加的，并不是固定写死的。所以会大量使用setData( )方法。如下图，



chartColumnLandscape就是我们的图表对象，series[i]就是我们的数据列，这里说明我们有4组数据。可以看到setData( )方法带有4个参数，第一个参数是数组，也就是我们需要动态添加的数据，是必选项。后面3个参数是可选项，第二个参数是redraw: Boolean，是否在更新图表后对图表进行重绘操作，默认是true，如果同时有多个图表更新操作，建议将此参数设置为 false，并在所有操作结束后调用 chart.redraw( ) 函数；第三个参数是animation: Mixed，如果新的数据长度和当前数据列的数据长度相等，那么将对数据点进行更新操作并包含默认的动画效果，可以设置此参数为 false 来关闭动画，另外还可以设置成包含 duration 及 easing 属性的对象形式；第四个参数是updatePoints: Boolean，当更新的数据长度和当前数据列数据长度想等时，数据点将会被更新而不是替换（如果数据长度不相等，则是替换），这种情况下更新数据是包含动画的，并且更新的数据不是以对象引用的形式传参的。可以通过将此参数设置为 false 来阻止。

Highcharts还有一个对于初学者来说容易搞混淆的地方，就是在tooltip属性里的pointFormat，pointFormatter，formatter这3个属性。简单来说format结尾的属性值是String字符串，而formatter结尾的属性值是函数Function，而point代表数据点，会根据数据点的不同显示不同数据。

pointFormat是数据点格式化字符串。提示框中该点的HTML代码， 变量用花括号括起来。可选变量有 point.x，point.y，series.name 和series.color 以及以相同的形式表示的其它属性。tooltip.pointFormat能够被每一个数据列覆盖，因此给数值赋予个性化单位就很方便。例如：pointFormat：'{series.name}: <b>{point.y}元</b><br/>'。pointFormat经常与headerFormat(标题格式化字符串)、footerFormat(尾部格式化字符串)一起使用，它们的区别其实就是point是在循环内，它有多少个数据点就会显示多少条，而headerFormat和footerFormat是在循环外，每个提示框只会出现一次。

pointFormatter是数据点格式化回调函数，返回格式化提示框中该点的HTML代码。类似于 pointFormat，但是更灵活。

Formatter是提示框内容格式化回调函数，返回 false 可以针对某个点或数据列关闭提示框，该函数里可以执行复杂的逻辑来返回需要的内容。默认情况下只支持部分 HTML 标签，因为最终会转换成 SVG 标签，默认支持的 HTML 标签有：<b>， <strong>，<i>， <em>，<br/>，<span>。Span标签可以通过 style 属性来设置样式，但是只支持和文字相关及 SVG 相关的样式属性。当设置 useHTML 为 true 时，提示框可以支持 HTML 标签，并最终以 [HTML 默认](https://www.hcharts.cn/docs/basic-labels-string-formatting#h2-0) 来渲染提示框。从 Highcharts 2.1 开始，可以通过共享（shared） 属性来将多个数据列的信息展示在同一个提示框里。提示框格式化函数里数据点对象的获取和共享与否有关，在共享的提示框里，数据点对象可以 this.points 来获取，该对象是个数组，表示同一个 x 值下的所有数据点；在非共享的提示框中，数据点对象是通过 this.point 来获取，表示当前数据点对象。根据提示框共享与否，提示框里可以用的变量有：

1. this.percentage （非共享） / this.points[i].percentage （共享）

只对堆叠数据列有效，表示当前点在堆叠中的比例

1. this.point （非共享） / this.points[i].point （共享）

数据点对象或对象数组

1. this.points

共享提示框情况下，相同 x 轴的所有点对象数组

1. this.series （非共享） / this.points[i].series （共享）

数据列对象，数据列的名字可以通过 this.series.name 或 this.points[i].series.name 来获取

1. this.total （非共享） / this.points[i].total （共享）

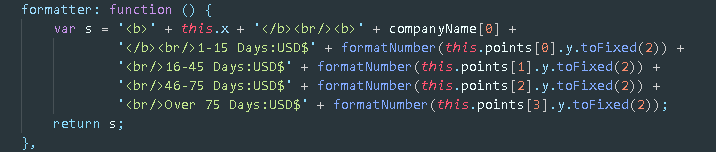
只对堆叠图数据列有效，表示当前 x 轴下堆叠总值

1. this.x

x 值，提示框共享与否该值的取值方式一样

1. this.y （非共享） / this.points[i].y （共享）

y 值



## div布局

UI布局，每个page对应一个view，之前也提到过一些，就不再赘述。每个页面当中又都有tab，切换tab可以显示不同内容。比如说第一页有2个tab，分别是“BU”和“CSD”，他们虽然是2个不同的tab，但里面的内容形式都一样，都是2个图表，只是数据上有差别，所以我们使用同一个div容器，只是切换tab时，使用上面提到的setData( )方法进行动态赋值即可。第二页有3个tab，他们里面的布局都各不相同，所以我们设置了3个div分别对应3个tab，在切换tab时只显示对应的那个div，其他2个隐藏即可。然后标签下面的内容大部分都是list形式，格式相同，所以我使用的ul li里面嵌套div来实现。

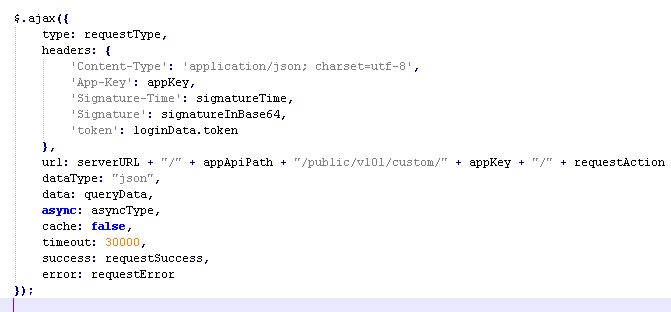
因为scroll menu是紧贴页面底部的，所以将scroll menu放在了data-role = “footer”当中，并设置data-position = "fixed"将它定位在底部，data-tap-toggle="false"让它不能移动。这就是我在布局方面的一些简单思路和方法。

# 数据绑定

## 获取API

### Ajax

数据方面的获取是通过ajax发送异步请求，并且是QPlay APP统一呼叫API的方式，函数在APP.js里面，我们只需要接受返回值即可。



type定义了数据传输方式是post还是get。

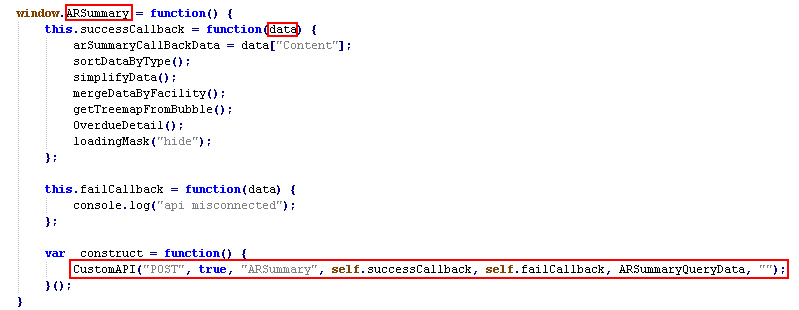
Headers拼接了头部部分，url是数据获取的地址。

dataType定义数据是json格式，async是异步获取。

success成功回调函数，error失败回调函数。

### 回调函数

总共有5个API，第一个ARSummary对应的是第一页的气泡图和矩形图，第二个是AraUserAuthority主要用来管理用户权限，包括第一页的气泡图和第二页的facility菜单，每个用户会根据自身的权限显示对应的facility信息。第三个是OverdueDetail对应第二页tab1，第四个是OutstandDetail对应第二页tab2，第五个是CreditExpiredSoon对应第二页tab3。



ARSummary就是我们的API名称，data就是成功以后得到的返回值，

CustomAPI就是公函里面的方法，参数就是对应前面ajax需要的值。

## 数据处理

### JSON

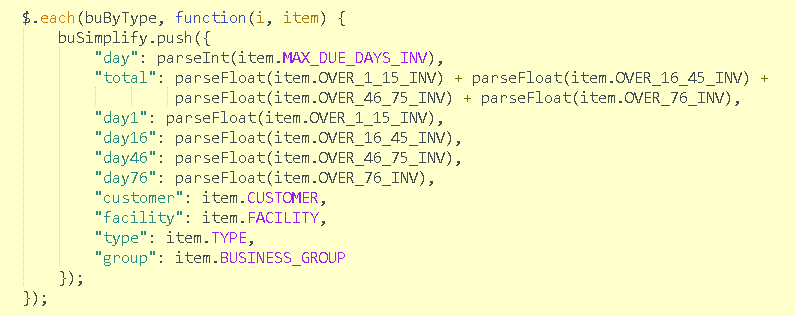
数据获取到了之后，因为数据量较多，但有些可能是我们暂时不需要的，所以就需要对数据进行处理，然后返回我们想要的数据内容及格式。

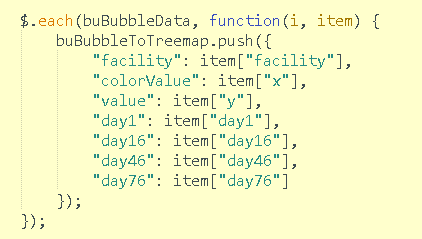
我们在调用API成功以后，success回调函数会返回一个data，它一般都是json格式。[JSON](https://baike.baidu.com/item/JSON)是一种轻量级的数据交换格式。它基于 [ECMAScript](https://baike.baidu.com/item/ECMAScript)的一个子集，采用完全独立于编程语言的文本格式来存储和表示数据。简洁和清晰的层次结构使得 JSON 成为理想的数据交换语言。易于人阅读和编写，同时也易于机器解析和生成，并有效地提升网络传输效率。任何支持的类型都可以通过 JSON 来表示，例如字符串、数字、对象、数组等。JSON有2个内置方法：

1. JSON.stringify( )，将JSON转换成string字符串，在用ajax异步请求的时候，JSON数据只能通过string字符串类型进行传输，所以在传输前要把JSON转换成string字符串格式。
2. JSON.parse( )，将string字符串转换成JSON，与JSON.stringify( )方法作用相反。一般两者都是结合使用

### $.each

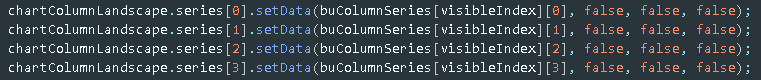
jQuery给我们提供了$.each( )遍历数组的方法。它可以遍历一维数组、多维数组、DOM、JSON 等等，功能强大。





因为JSON对象采用的是键值对的方式保存，所以我们可以通过obj.属性名或者obj[“属性名”]这两种方式来获取对应属性的值。我们遍历数组，然后获取需要的对象的属性值，再将处理过后的数据同样使用对象（键值对）的方式push到新的数组当中。

数据处理我把它分成两个方向，一个是Highcharts图表数据，一个是html页面显示数据。图表数据我们使用setData( )方法添加。



页面数据是将html标签利用字符串拼接的方式生成页面元素，最后利用append( )方法再添加到页面当中。



### localStorage

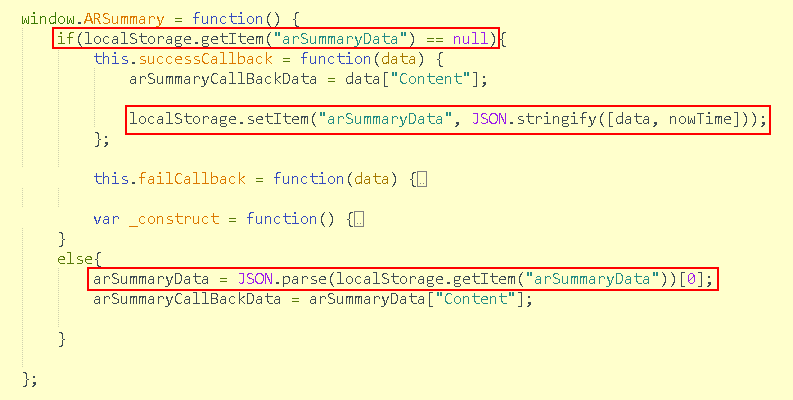
localStorage是H5新特性，可以把JSON存储在浏览器客户端，不手动删除不会自动失效。优点是可以减轻服务器压力，本地获取方便，减少响应时间。

localStorage有3个方法：

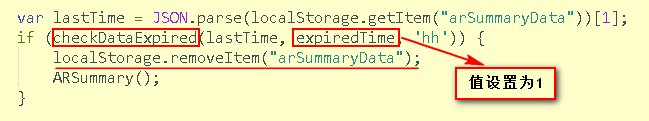
1. localStorage.getItem( )，获取
2. localStorage.setItem( )，设置
3. localStorage.remove( )，删除

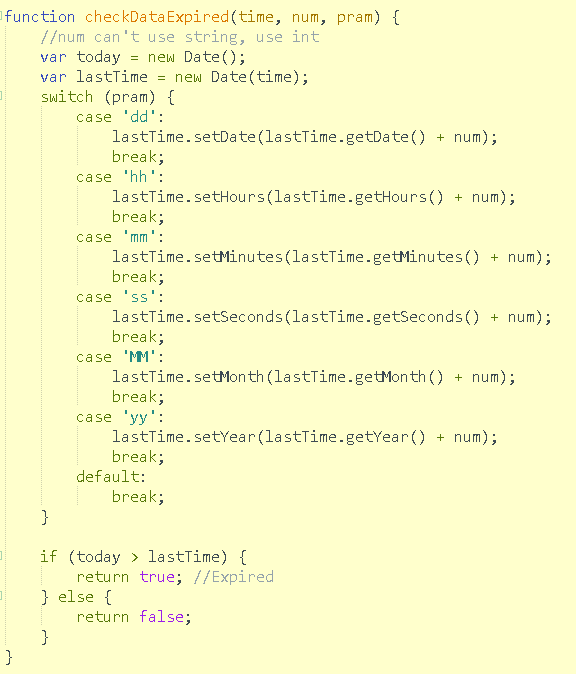
需要注意的地方是当我们在设置localStorage的时候，要把JSON转换成string字符串，在获取之后要把string字符串转换成JSON。这就要结合上面讲到JSON的2个方法一起使用。

localStorage的使用逻辑，首先需要判断localStorage是否为空，如果有值直接使用localStorage.getItem( )方法获取，如果为空则使用localStorage.setItem( )方法为其赋值。



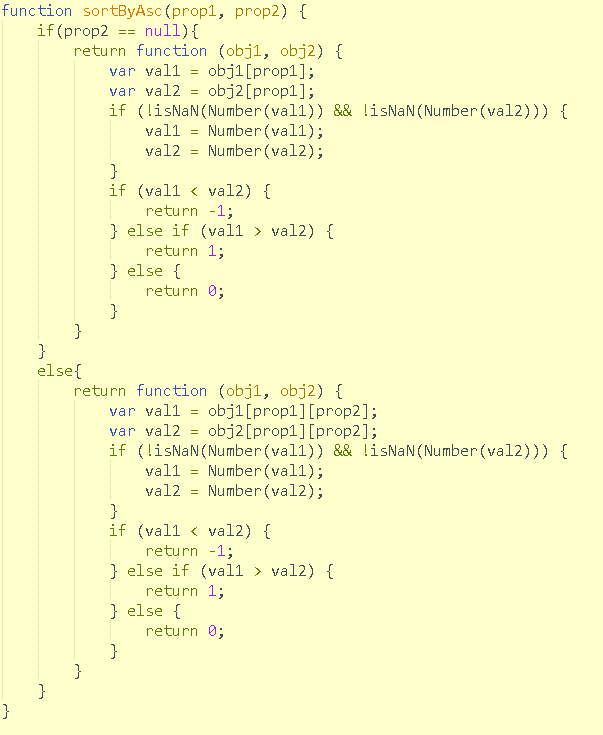
因为数据不是实时更新，所以我们可以使用localStorage把数据存储在客户端。当数据更新，这个时候就需要把localStorage清除。QisdaEIS我们设置的localStorage实效是1h，逻辑是本次登录距离上次登录时间超过1h则清除localStorage，再根据当前新时间重新存在本地；如果本次登录距离上次登录时间没有超过1h，则不用清除。





### sort

数据排序功能，根据需求数据可以通过CUSTOMER升序或降序，还可以通过TOTAL升序或者降序，以及TIME升序降序。不管怎么排它们的逻辑都是一样的，只是参数上的差别。



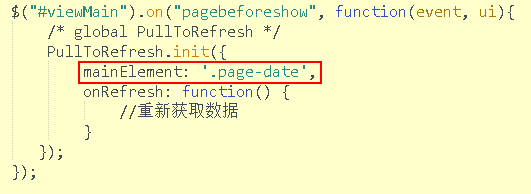


### Pull Refresh

Pull refresh是下拉更新数据操作，首先需要引入pulltorefresh.js，路径EnterpriseAPPPlatform\APP\component\lib\pulltorefresh.js，拷贝到我们项目www\js\lib下即可。

然后在html页面定义一个div元素，比如<div class=” page-date”></div>

接着套用PullToRefresh.init({… })初始化方法，mainElement就是我们class=”page-data”的div元素，onRefresh就是我们更新数据的function，其操作逻辑就是先清除localStorage，再调用API即可。

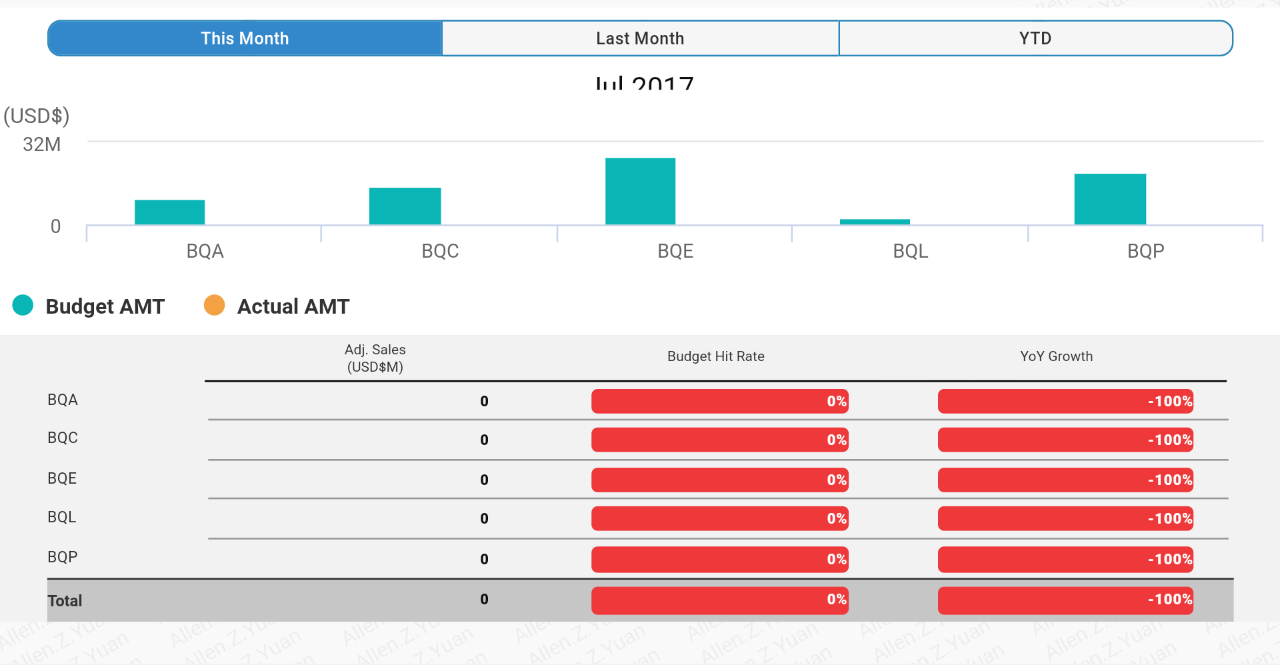


# 问题及解决办法

在开发过程当中会遇到许多问题，有一些典型的错误，还有一些疑难点，在这里罗列出来。

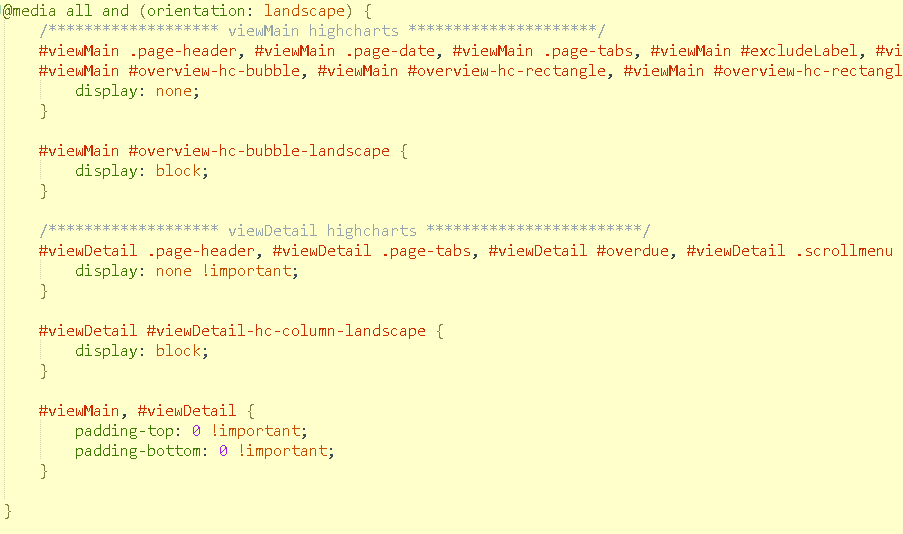
## iOS屏幕翻转

在移动设备上我们容器和字体使用的单位是vw和vh，是根据屏幕宽高等比例缩放，从而实现响应式。当横屏状态下，屏幕宽度变成了竖屏时的高度，图表容器的宽高也随之发生变化。如下图：

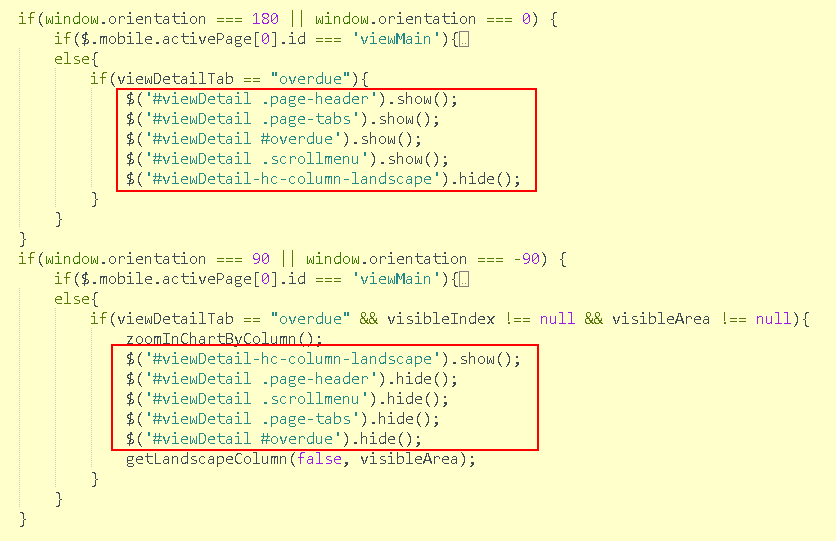


错误现象，在iOS设备上，横屏转竖屏后，但堆叠柱状图(column)还是保持横屏下的宽和高，导致在竖屏状态下，column只能显示一部分。

我在设置div宽高的时候都是使用的vw单位，在竖屏下是没有问题的。我最先怀疑的地方是在横屏时，如果div的高度还是使用vw单位，因为横屏时宽度变大，div可能会变形严重，所以我将所有div的高度换成对应的vh单位。但在iOS下，还是会出现显示不全的问题。所以又到翻转屏幕事件里面去找原因。当屏幕翻转的时候，在index.css里面有层的显示和隐藏的操作，使用的是媒体查询@media。



我们也可以使用js在翻转时来对DOM进行show( )或hide( )的操作。



我当时使用的是js方法来控制显示和隐藏，后来改成只用css来控制，

iOS显示不全的问题就没有了。

可以将问题延伸一下，如果css和js同时控制某种样式，那js的优先级肯定是要高于css的，但如果给样式添加！important提高css优先级的话，那即使你使用了js也不会有效果的，因为现在css优先级是要高于js的。

结论，能够用css解决的问题就使用css，而不是js。

## 加载时间过长

在将数据呈现到页面上，并添加图表数据时，发现页面加载时间长达20秒左右。后来找到原因，是因为一次加载了所有的图表，图表渲染时间就长达18秒。因为当时BU下有150个折线图和150个柱形图，而CSD下有50个折线图和50个柱形图，总共一起有400张图表一起加载。因为柱形图原本是隐藏的，只有展开后才能看到。所以我就先把柱形图拿掉，加载时间变成10秒。后来又将图表绘制时的动画效果animation设置为false，加载时间变成7秒。因为一页一次最多只能显示11条数据，所以我又利用页面可视区域对数据进行分页显示，每次加载12条数据，如果页面滚动到第11条的时候，就加载13-24的数据，依次类推，这样就将每次加载的图表数量降到最低。一次只加载12个图表，跟原来400张图表相比性能得到了巨大的提升，加载时间也只有3秒。这个问题也得到了解决，但我只加载了折线图，柱形图还没有加载。因为它需要展开才能看到，所以我的办法是点开某个按钮，再将图表数据设置进去，这样点击一次加载一个，对性能也没有影响。我在生成html时，给元素添加了一个自定义属性data-index=“i”，用来记录每条数据在数组当中的索引位置，这样通过索引i找到数据，再将数据添加到图表当中。

展开所有柱形图的方法也是将柱形图分页，因为展开后最多只能显示2条数据，所以我一次只加载4条，当屏幕发生滚动时触发onscroll事件，监听数据是否在可视区域内，当在可视区域内时再加载后4条。这些对于可视区域的利用，也让我认识到在移动开发中的重要性，之前不太了解，也绕了不少弯，对于可视区域的利用还不止这些，在今后的工作中我也会加强这方面的运用。

## Landscape Highcharts Tooltip

在横屏状态下，只有一个div用来显示堆叠柱状图(column)，但是在switch开关关闭状态下，column只有4组数据，tooltip显示正常。但switch开关打开，增加显示负值，column数据也由4组变为8组。

在绑定数据的时候，还有2个功能，一个是点击scroll menu筛选facility，一个switch开关来显示正负值。当时在想解决方案的时候，一直犹豫是一次把所有数据按照facility分组后全部加载出来，然后再根据facility只显示我所需的数据，其他数据隐藏。还是说我把数据放在一个数组当中，然后每次的筛选之前都清空数组和html，就不需要隐藏和显示的操作。第二种方法的坏处是每次点击facility时都需要运算，然后添加数据，渲染图表，所以每次点击时都有不到1秒的加载时间。但这种方法的好处是，第一次加载的时间就不会受到影响（本来第一次加载的时间就已经到极限3秒钟），不然第一次加载时的运算和图表都会相应的增加，时间也会再次突破3秒钟，所以我选择了第二种方法。而switch开关显示正负值的操作，我也采用了相同的方法。当我做完再回头想想以后，发现其实这2种方法都还有可以讨论的空间，也有可以优化的地方，下次再遇到类似问题的时候，我会更加仔细地斟酌一下。

## 总结

这次QisdaEIS开发对于我来说是一次非常宝贵的经验，让我了解了整个QPlay APP的开发流程，以及APP背后的运作方式，包括如何使用github进行版本控制和团队合作，如何使用jenkins进行daily build自动生成对应版本，如何使用gulp合成文件或者压缩文件，如何在浏览器上debug手机APP等等。也使我认识到自己的不足，包括代码的可维护性、方法的封装、循环的使用、性能的优化等。争取在下次开发中做得更好。