### 前端重构思路

* 1. 搞清楚被重构项目的配置。如：
     1. jQuery：DOM操作，含UI插件
     2. Bootstrap：UI风格
     3. AWF：企业级组件库
     4. Gulp：打包工具
     5. 单页面应用（SPA）
  2. 确定重构方案。如：
     1. Require.js（text+css插件）：模块化加载（我理解是懒加载）
     2. Mockjs：模拟ajax调用
     3. Director.js：路由（vue里面自带vue-router）
     4. Underscore：模块化
     5. jQuery：DOM操作，含UI及插件
     6. Bootstrap：UI风格（建议常用的element-UI）
     7. AWF：企业级组件库（建议常用的element-UI）
     8. 打包：gulp+r.js

### **如何在Vue项目中使用vw实现移动端适配**

* 1. <https://www.jianshu.com/p/1f1b23f8348f>

### **CSS3 的视口单位vw、vh实现自适应（带有px，em，rem的简单介绍）**

* 1. px：像素单位，即图像的基本采样单位。对不同的设备，它的图像基本单位是不同的，如：显示器和打印机。通常来说，显示器分辨率是指的是桌面设定的分辨率，不是物理分辨率。但是为了最佳显示效果，一般显示器分辨率就等于物理分辨率。由此导致：**使用px会严重受到桌面分辨率的影响，需要做自适应**
  2. em：相对长度单位。相对于当前对象内本文的字体尺寸（**即受父元素的影响，**如果父元素没有，会一直往上找，直到body默认字体尺寸，也就是16px，则此时换算就是1em=16px）。如果使用em，建议将body的fon-size设置成62.5%，即16\*62.5%=10px，此时1em=10px，方便后面计算。
  3. rem：相对长度单位，但是**rem始终都是相对html的根元素**，不会受到对象内文本字体尺寸的影响，只需要改变根元素就能改变所有字体大小。兼容IE8以上即其他所有浏览器。
  4. vw、vh：视口（viewport units）单位，即**根据浏览器窗口大小的单位**，不收显示器分辨率的影响。vw（可视窗口的宽度）vh（可视窗口的高度）类似百分比，1vw=可视窗口的百分之一。如：窗口宽度大小是1800px，那么1vw=18px。但是**vw始终相对于可视窗口的宽度，而百分比受到父元素的影响。**vmin和vmax是指选择vm和vh中最小的那个，而vmax是选择最大的那个。**兼容性较差。**vm和vh由于受浏览器窗口影响，所以很轻松完成响应式布局。

5.em举例：

<style>

body {

font-size: 10px;

}

div {

font-size: 3em;

margin-top: 3em;

}

p {

font-size: 2em;

}

span {

font-size: 3em;

}

</style>

</head>

<body>body

<div>div

<p>p<br/>

<span>span</span>

</p>

</div>

</body>结果：

（理解：div元素的font-size =3em \* 10 = 30px margin-top=3em \* 10px=30px

p元素的font-size =2em \* 30px=60px; span元素的font-size=3em \* 60px=180px）

6.rem举例：

<style>

body {

font-size: 10px;

}

div {

font-size: 3rem;

}

p {

font-size: 2rem;

}

span {

font-size: 3rem;

}

</style>

</head>

<body>body

<div>div

<p>p<br/>

<span>span</span>

</p>

</div>

</body>

结果：

（理解：其换算结果都只跟body的10px相关）

1. **JS类**
   1. addEventListener：
      1. 定义：addEventListener()方法用于指向指定元素添加事件句柄（提示：使用removeEventListener()方法来移除addEventListener()方法添加的事件句柄）。
      2. 语法：element.addEventListener(event, function, useCapture)
2. event：必须。字符串，指定事件名（**注意：不要使用“on”前缀。例如：写“click”而不是“onclick”**）
3. function：必须。指定要触发的函数。（提示：当事件对象会作为第一个参数传入函数，事件对象的类型取决于特定的事件。例如，“click”事件属于mouseEvent（鼠标事件）对象）
   * 1. **useCapture：可选，布尔值，指定事件是否在捕获或冒泡阶段执行**
   1. **true——事件句柄在捕获阶段执行**
   2. **false——默认，事件句柄在冒泡阶段执行**
      1. useCapture的作用：
4. 假设当前有一个HTML内容：div、p、span元素都有click事件的响应逻辑

<div>

<p>

<span></span>

</p>

</div>

如果此时的第三个参数为false的时候：点击span依次触发——span>p>div

如果此时的第三个参数为true的时候：点击span依次触发——div>p>span

* + 1. **如何阻止冒泡：**

fuction(evt) {

evt.[stopPropagation()](https://www.w3school.com.cn/jsref/event_stoppropagation.asp); // 不会阻止本dom本事件的其他事件处理函数的执行

evt.stopImmediatePropagation(); // 会导致后续绑定在该dom的其他事件处理函数不再执行；

**解析：当我给一个span标签添加两个addEventListener，当我在第一个addEventListener里面添加evt.[stopPropagation()](https://www.w3school.com.cn/jsref/event_stoppropagation.asp)时，停止冒泡，但是第二个addEventListener仍然会继续；而evt.stopImmediatePropagation()则连第二个addEventListener都不会执行了**

}

* + 1. 与on的区别：

以click为例：

1. **on事件会被后面的on的事件覆盖，addEventListener则会全部打印**：

obj.onclick(function() {

alert(‘aaa’);

});

obj.onclick(function() {

alert(‘bbb’);

});

打印结果:‘bbb’

obj.addEventListener(“click”, fn(){

alert(‘aaa’);

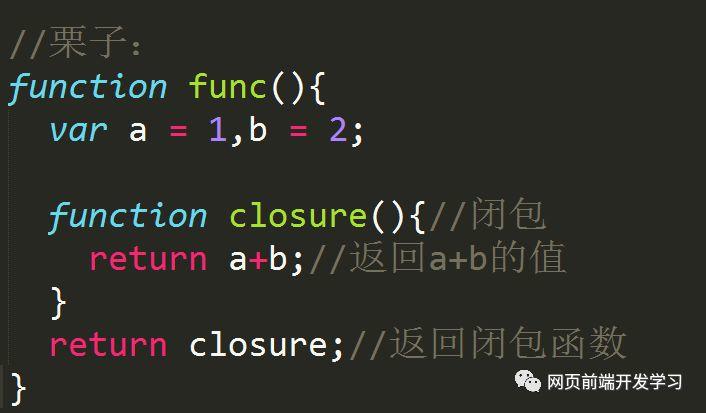
});

obj.addEventListener(“click”, fn(){

alert(‘bbb’);

});

打印结果:‘aaa’‘bbb’

* 1. 闭包：
     1. 定义：简单来说就是有权方位另一个函数作用域中变量的函数，MDN上面这么说：闭包是一种特殊的对象。
     2. 理解：**外部函数调用之后，其变量对象本应该被销毁，但是闭包的存在就使得我们仍然可以访问外部函数的变量对象。**
     3. 举例： 
     4. 产生一个闭包：创建一个闭包最常见的方式就是一个函数内部创建另一个函数，右图例子中的closure就是一个闭包：
     5. 作用域：闭包的作用域包含着他自己的作用域，以及包含他的函数的作用域和全局作用域。



* + 1. 注意事项：通常函数的作用域及其所有变量都会在函数执行结束后被销毁。但是，在创建了一个闭包后，这个函数的作用域就会一直保存在闭包不存在为止。举例：

右图所示代码可以看到add5和add10都是闭包。他们共享相同的函数定义，但是保存了不同的环境。在add5的环境中，x为5。而在add10中，x为10。最后通过null释放了add5和add10对闭包的引用。在JavaScript中，如果一个对象不再被引用，那么这个对象就会被垃圾回收机制回收；如果两个对象相互引用，而不再被第三者引用，那么这两个相互引用的对象也会被回收。

* + 1. **闭包只能取得包含函数中的任何变量中的最后一个值**

如右图：arr数组中包含了10个匿名函数，每个匿名函数都能访问到外部函数的变量i。

当arr()执行完毕之后，其作用域被销毁，但他的变量对象仍然保存在内存中，得意被匿名访问，这时候的i值为10。

要想保存在循环过程中的每一个i值，需要在匿名函数外部再套一个匿名函数，在这个匿名函数中定义另一个变量并且立即执行来保存i值。如右图的brr()。



* + 1. this对象

如右图所示：在上面这段代码中，obj1.getName()()实际上是在全局作用域中调用了匿名函数，this指向了window（可以理解为：这个匿名函数其实仅仅只是写在obj1里面而已。我把这个匿名函数拿到打印这里执行，其实此时是跟obj1没有任何关系了，这时候的this指向肯定指的是全局的name1而不是obj1里面的name1了）。

这里要清楚：**函数名与函数功能是分割开的，不要认为函数在哪里，其内部的this就指向哪里。**

相比obj2.getName()()，执行函数内部的that是父函数保存的this（因为闭包本身可以保存函数外的变量嘛），所以可以指向obj内部。

obj3.getName()是在对象obj3的环境中执行的，所以this指向obj。而(obj3.getName = obj3.getName)赋值语句返回的是等号右边的值，而右边的值是一个函数且此时的函数还没有执行就已经在全局作用域中返回，所以(obj3.getName = obj3.getName)();的this指向全局。要把函数名和函数功能分隔开。

* + 1. 内存泄漏

闭包会引用包含函数的整个变量对象，如果必报的作用域中保存着一个HTML元素，那么就意味着钙元素无法被销毁。我们有必要在对这个元素的操作完之后主动销毁。

* + 1. 函数内部定时器

当函数内部的定时器引用了外部函数变量对象时，该对象不会被销毁

* + 1. 闭包的应用

应用闭包的主要场合是：**设计私有的方法和变量。**任何在函数中定义的变量，都可以认为是私有变量，因为不能在函数外部访问这些变量，私有变量包括函数的参数、局部变量和函数内定义的其他函数。

把有权访问私有变量的公有方法成为**特权方法**

在这里，需要理解两个概念：

**模块模式：**为单例创建私有变量和方法。

**单例：**指的是只有一个实例的对象。JavaScript一般以对象字面量的方法来创建一个单例对象。

**匿名函数最大的用途就是创建闭包，并且还可以构建命名空间，以减少全局变量的使用。从而使用闭包模块化代码，减少全局变量的污染。**

下面这段代码中的addEvent和removeEvent都是局部变量，但我们可以通过全局变量objEvent使用它，这就大大减少了全局变量的使用，**增强网页的安全性。**

* + 1. **运用闭包的关键**

**闭包引用外部函数变量对象中的值；**

**在外部函数的外部调用闭包。**

* + 1. **闭包的缺陷**

闭包的缺点就是常驻内存会增大内存使用量，并且使用不当很容易造成内存泄漏。

如果不是因为某些特殊任务而需要闭包，在没有必要的情况下，在其他函数中创建函数是不明智的，因为闭包对脚本性能具有负面影响，包括处理速度和内存消耗。

* + 1. 面试题



* 1. Promise：
     1. 含义：本质上，一个promise是某个函数返回的对象，你可以把回调函数绑定在这个对象上，而不是把回调函数当做参数传进函数
     2. 执行顺序原理：**同步优先，异步靠边（setTimeout可算异步）**
     3. 规范：

1. 一个promise可能有三种状态：等待（pending）、已完成（resolved/fulfilled）、已拒绝（rejected）
2. 一个promise的状态只可能从“等待”转到“完成”或“拒绝”。不能逆向转换，同时“完成”和“拒绝”两种状态不能相互转换
3. Promise必须实现then方法（可以说，then就是promise的核心），而且then必须返回一个promise，同一个promise的then可以调用多次，并且回调的执行顺序跟着它们被定义时的顺序一致
4. Then方法接收两个参数，第一个是成功时候的回调（在resolved状态调用），另一个是失败时候的回调（在rejected状态调用）。
   * 1. promise.resolve()和promise.reject() 两者都是new Promise()的快捷方式。
     2. Promise.all()和Promise.race()

1）promise.all()：接收一个promise对象的数组作为参数

* + 1. 关于catch的提问：catch后面还能不能继续catch到错误？Catch后还能不能继续then？then后面的catch还能不能catch到then的错误？

1. 如果异步操作抛出错误，状态就会变成rejected，就会调用catch方法指定的回调函数处理错误。
2. Then方法指定的回调函数在运行中报错也会被catch捕获到。
3. 在resolve()后面抛出的错误会被忽略（如果前面已经是执行了resolve，那么后面throw出来的error就不会被catch）
4. Catch返回的也是一个promise，因此后面可继续then（不管是then方法还是catch方法返回的都是一个新的promise实例，这意味着promise可以**链式调用**then和catch，每一个方法的返回值作为下一个方法的参数）
5. 在异步函数中抛出的错误不会被catch捕获到（例如使用了setTimeOut之类的，抛出来的错误不会被catch到）
6. 有多个catch连写，如果在catch中继续throw出异常，那么后面的catch就会一直执行，直到不再抛出异常。（catch其实是then的语法糖）