HTML/CSS/JS编码规范

一、HTML编码规范

1. img标签要写alt属性

根据W3C标准，img标签要写alt属性，如果没有就写一个空的。但是一般要写一个有内容的，根据图片想要表达的意思，因为alt是在图片无法加载时显示的文字。

如下不太好的写法：

```
<img src="company-logo.svg" alt="ABC Company Logo">
```

更好的写法：

```
<img src="chime-logo.svg" alt="ABC Company">
```

这里就不用告诉用户它是一个Logo了，直接告诉它是ABC Compay就好了。

再如：

```
<img src="user-head.jpg" alt="User Portrait">
```

可改成：

```
<img src="user-head.jpg" alt="Bolynga Team">
```

如果图片显示不出来，就直接显示用户的名字。

有些人偷懒就直接写个空的alt那也可以，但是在一些重要的地方还是要写一下，毕竟它还是有利于SEO.

2. 单标签不要写闭合标签

为什么？因为写了也没用，还显得你不懂html规范，我们不是写XHTML。常见的单标签有img、link、input、hr、br，如下不好的写法：

```
<img src="test.jpg"></img><br/>
<input type="email" value=""/>
```

应改成：

```
<img src="test.jpg"><br>
<input type="email" value="">
```

如果你用React写jsx模板，它就要求每个标签都要闭合，但是它始终不是原生html.

3. 自定义属性要以data-开头

自己添加的非标准的属性要以data-开头，否则w3c validator会认为是不规范的，如下不好的写法：

```
<div count="5"></div>
```

应改成：

```
<div data-count="5"></div>
```

4. td要在tr里面，li要在ul/ol里面

如下不好的写法：

```
<div>
    <li></li>
    <li></li>
</div>
```

更常见的是td没有写在tr里面：

```
<table>
    <td></td>
    <td></td>
</table>
```

如果你写得不规范，有些浏览器会帮你矫正，但是有些可能就没有那么幸运。因为标准并没有说如果写得不规范应该怎么处理，各家浏览器可能有自己的处理方式。

5. ul/ol的直接子元素只能是li

有时候可能会直接在ul里面写一个div或者span，以为这样没关系：

```
<ol>
    <span>123</span>
    <li>a</li>
    <li>b</li>
</ol>
```

这样写也是不规范的，不能直接在ol里面写span，ol是一个列表，它的子元素应该都是display: list-item的，突然冒出来个span，你让浏览器如何自处。

所以写得不规范就会导致在不同的浏览器会有不同的表现。

同样，tr的直接子元素都应该是td，你在td里面写tr那就乱了。

6. section里面要有标题标签

如果你用了section/aside/article/nav这种标签的话，需要在里面写一个h1/h2/h3之类的标题标签，因为这四个标签可以划分章节，它们都是独立的章节，需要有标题，如果UI里面根本就没有标题呢?

那你可以写一个隐藏的标题标签，如果出于SEO的目的，你不能直接display: none，而要用一些特殊的处理方式，如下套一个hidden-text的类：

```
<style>.hidden-text{position: absolute; left: -9999px; right: -9999px}
</style>
<section>
    <h1 class="hidden-text">Listing Detail</h1>
</section>
```

7. 使用section标签增强SEO

使用section的好处是可以划分章节，如下代码：

```
<body>
<h1>Listing Detail</h1>
<section>
    <h1>House Infomation</h1>
    <section>
       <h1>LOCATION</h1>
       <p></p>
    </section>
    <section>
        <h1>BUILDING</h1>
        <p></p>
    </section>
</section>
<section>
    <h1>Listing Picture</h1>
</section>
</body>
```

就会被outline成这样的大纲：

```
Listing Detail
House Infomation
LOCATION
BUILDING
Listing Picture
```

可以使用html5 outliner进行实验，可以看到，我们很任性地使用了多个h1标签，这个在html4里面是不合法的。

8. 行内元素里面不可使用块级元素

例如下面的写法是不合法的：

```
<a href="/listing?id=123">
    <div></div>
</a>
```

a标签是一个行内元素，行内元素里面套了一个div的标签，这样可能会导致a标签无法正常点击。再或者是span里面套了div，这种情况下需要把inline元素显式地设置display为block，如下代码：

```
<a href="/listing?id=123" style="display: block">
    <div></div>
</a>
```

这样就正常了。

9. 每个页面要写<!DOCType html>

设置页面的渲染模式为标准模式，如果忘记写了会怎么样？忘记写了会变成怪异模式，怪异模式下很多东西渲染会有所不同，怪异模式下input/textarea的默认盒模型会变成border-box，文档高度会变成可视窗口的高度，获取window的高度时就不是期望的文档高度。

还有一个区别，父容器行高在怪异模式下将不会影响子元素，如下代码：

```
<div><img src="test.jpg" style="height:100px"></div>
```

在标准模式下div下方会留点空白，而在怪异模式下会。这个就提醒我们在写邮件模板时需要在顶部加上<!DOCType html>，因为在本地开发邮件模板时是写html片段，没有这个的话就会变成怪异模式。

10. 要用table布局写邮件模板

由于邮件客户端多种多样，你不知道用户是使用什么看的邮件，有可能是用的网页邮箱，也有可能用的gmail/outlook/网易邮箱大师等客户端。

这些客户端多种多样，对html/css的支持也不一，所以我们不能使用高级的布局和排版，例如flex/float/absolute定位，使用较初级的table布局能够达到兼容性最好的效果，并且还有伸缩的效果。

另外邮件模板里面不能写媒体查询，不能写script，不能写外联样式，这些都会被邮件客户端过滤掉，样式都得用内联style，你可以先写成外联，然后再用一些工具帮你生成内联html。

写完后要实际测一下，可以用QQ邮箱发送，它支持发送html格式文本，发完后在不同的客户端打开看一下，看有没有问题，如手机的客户端，电脑的客户端，以及浏览器。

由于你不知道用户是用手机打开还是电脑打开，所以你不能把邮件内容的宽度写死，但是完全100%也不好，在PC大屏幕上看起来可能会太大，所以一般可以这样写：

```
<table style="border-collapse:collapse;font-family: Helvetica
Neue,Helvetica,Arial;font-size:14px;width:100%;height:100%">
    <tr><td align="center" valign="top"><table style="border:1px solid
#ececec;border-top:none; max-width:600px;border-collapse:collapse">
    <tr><td>内容1</td></tr>
    <tr><td>内容2</td></tr>
</table></td></tr></table>
```

最外面的table宽度100%，里面的table有一个max-width:600px，相对于外面的table居中。这样在PC上最大宽度就为600px，而在手机客户端上宽度就为100%。

但是有些客户端如比较老的outlook无法识别max-width的属性，导致在PC上太宽。但是这个没有办法，因为我们不能直接把宽度写死不然在手机上就要左右滑了，也不能写script判断ua之类的方法。所以无法兼容较老版本outlook.

11. html要保持简洁，不要套太多层

需要套很多层的，一般有两种情况，一种是切图不太会，需要套很多层来维持排版，第二种是会切图，但是把UI拆解得太细。像以下布局：

我会这么写：

```
<ul>
    <li>
        <div class="img-container">
            <img>
        </div>
        <div class="text"></div>
    </li>
</ul>
```

因为它是一个列表，所以外面用ul/li作为容器，里面就两个div，一个图片的float: left，另外一个文字的容器，这样就可以了。不用套很多层，但是有一些是必要的，你写得太简单的话，扩展性不好，例如它是一个列表那你就应该使用ul/ol，如果你要清除浮动，那你可能需要套一个clearfix的容器。

如果有一块是整体，它整体向右排版，那么这一块也要套一个容器，有时候为了实现一些效果，可能也要多套一个容器，例如让外面的容器float，而里面的容器display: table-cell。但是你套这些容器应该都是有价值，如果你只是为了在功能上看起来整齐划一，区分明显好像没太大的意义。

12. 特殊情况下才在html里面写script和style

通常来说，在html里面直接写script和style是一种不好的实践，这样把样式、结构和逻辑都掺杂在一起了。但是有时候为了避免闪屏的问题，可能会直接在相应的html后面跟上调整的script，这种script看起来有点丑陋，但是很实用，是没有办法的办法。

13. 样式要写在head标签里

样式不能写在body里，写在body里会导致渲染两次，特别是写得越靠后，可能会出现闪屏的情况，例如上面的已经渲染好了，突然遇到一个style标签，导致它要重新渲染，这样就闪了一下，不管是从码农的追求还是用户的体验，在body里面写style终究是一种下策。

同样地script不要写在head标签里面，会阻碍页面加载。

而CSS也推荐写成style标签直接嵌在页面上，因为如果搞个外链，浏览器需要先做域名解析，然后再建立连接，接着才是下载，这一套下来可能已经过了0.5s/1s，甚至2~3秒。而写在页面的CSS虽然无法缓存，但是本身它也不会很大，再加gzip压缩，基本上在50k以内。

14. html要加上lang的属性

如下，如果是英文的网页，应该这么写：

```
<html lang="en">
<html lang="en-US">
```

第一种表示它是英文的网页，第二种表示它是美国英语的网页，加上这个的好处是有利于SEO和屏幕阅读器使用者，他可以快速地知道这个网页是什么语言的，如果是中文可以这么写：

```
<html lang="zh-CN">
```

15. 要在head标签靠前位置写上charset的meta标签

如下，一般charset的meta标签要写在head标签后的第一个标签：

```
<head>
   <meta charset="utf-8">
</head>
```

一个原因是避免网页显示unicode符号时乱码，写在前面是因为w3c有规定，语言编码要在html文档的前1024个字节。如果不写的话在老的浏览器会有utf-7攻击的隐患，具体可以自行查阅资料，只是现在的浏览器基本都去掉了对utf-7编码的支持了。

charset的标签写成html5的这种比较简洁的写法就行了，不需要写成html4这种长长的：

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

据我所查，就算是IE6也支持那种简短的写法，虽然它不是一个html5浏览器。

16. 特殊符号使用html实体

不要直接把Unicode的特殊符号直接拷到html文档里面，要使用它对应的实体Entity，常用的如下表所示：

符号 实体编码

```
©   &copy;
¥   &yen;
®   &reg;
>   &gt;
<   &lt;
&   &amp;
```

特别是像©这种符号，不要从UI里面直接拷一个unicode的字符过去，如果直接拷过去会比较丑，它取的是用的字体里面的符号。

17. img空src的问题

有时候可能你需要在写一个空的img标签，然后在JS里面动态地给它赋src，所以你可能会这么写：

```
<img src="" alt>
```

但是这样写会有问题，如果你写了一个空的src，会导致浏览器认为src就是当前页面链接，然后会再一次请求当前页面，就跟你写一个a标签的href为空类似。

如果是background-image也会有类似的问题。

这个时候怎么办呢？如果你随便写一个不存在的url，浏览器会报404的错误。

我知道的有两种解决方法，第一种是把src写成about:blank，如下：

```
<img src="about:blank" alt>
```

这样它会去加载一个空白页面，这个没有兼容问题，不会加载当前页面，也不会报错。

第二种办法是写一个1px的透明像素的base64，如下代码所示：

```
<img
src="
AAIBRAA7">
```

第二种可能比较符合规范，但是第一种比较简单，并且没有兼容性问题。

18. 关于行内元素空格和换行的影响

有时候换行可能会引入空格，如下代码：

```
<form>
    <label>Email: </label>
    <input type="email">
</form>
```

在label和input中间会有一个空格，这样可能会导致设置lable的width和input 的width两者的和等于form的时候会导致input换行了，有时候你检查半天没查出原因，最后可能发现，原来是多了一个空格，而这个空格是换行引起的。

这个时候你可能会有一个问题，为什么<form>和<label>之间以及<input>和</form>之间的换行为什么没引入空格？这是因为块级元素开始的空白文本将会被忽略，如下Chrome源码的说明：

```
// Whitespace at the start of a block just goes away. Don't even

// make a layout object for this text.
```

并且，块级元素后面的空白文本结点将不会参与渲染，也就是说像这种：

```
<div></div>
<div></div>
```

两个div之间有textNode的文本节点，但是不会参与渲染。

要注意的是注释标签也是正常的页面标签，也会给它创建一个相应的节点，只是它不参与渲染。

19. 类的命名使用小写字母加中划线连接

如下使用-连接，不要使用驼峰式：

```
<div class="hello-world"></div>
```

20. 不推荐使用自定义标签

是否可以使用自定义标签，像angular那样都是用的自定义标签，如下代码：

```
<my-cotnainer></my-container>
```

一般不推荐使用自定义标签，angular也有开关可以控制是否要使用自定义标签。虽然使用自定义标签也是合法的，只要你给他display: block，它就像一个div一样了，但是不管是从SEO还是规范化的角度，自定义标签还是有点另类，虽然可能你会觉得它的语义化更好。

21. 重复杂id和重复属性

我们知道，如果在页面写了两个一模一样的id，那么查DOM的时候只会取第一个，同理重复的属性也会只取第一个，如下：

```
<input class="books" type="text" name="books" class="valid">
```

第二个class将会被忽略，className重复了又会怎么样？重复的也是无效的，这里主要是注意如果你直接操作原生className要注意避免className重复，如下代码：

```
var input = form.books;
input.className += " valid";
```

如果重复执行的话，className将会有重复的valid类。

22. 不推荐使用属性设置样式

例如，如果你要设置一个图片的宽高，可能这么写：

```
<img src="test.jpg" alt width="400" height="300">
```

这个在ios的safari上面是不支持的，可以自行实验。

或者table也有一些可以设置：

```
<table border="1"></table>
```

但是这种能够用CSS设置的就用CSS，但是有一个例外就是canvas的宽高需要写在html上，如下代码：

```
<canvas width="800" height="600"></canvas>
```

如果你用CSS设置的话它会变成拉伸，变得比较模糊。

23. 使用适合的标签

标签使用上不要太单调：

如果内容是表格就使用table，table有自适应的优点；如果是一个列表就使用ol/ul标签，扩展性比较好

如果是输入框就使用input，而不是写一个p标签，然后设置contenteditable=true，因为这个在IOS Safari上光标定位容易出现问题。如果需要做特殊效果除外

如果是粗体就使用b/strong，而不是自己设置font-weight

如果是表单就使用form标签，注意form里面不能套form

如果是跳链就使用a标签，而不是自己写onclick跳转。a标签里面不能套a标签

使用html5语义化标签，如导航使用nav，侧边栏使用aside，顶部和尾部使用header/footer，页面比较独立的部分可以使用article，如用户的评论。

如果是按钮就应该写一个button或者

```
<input type="button">
```

而不是写一个a标签设置样式，因为使用button可以设置disabled，然后使用CSS的:disabled，还有:active等伪类使用，例如在:active的时候设置按钮被按下去的感觉

如果是标题就应该使用标题标签h1/h2/h3，而不是自己写一个

```
<p class="title"></p>
```

相反如果内容不是标题就不要使用标题标签了

在手机上使用select标签，会有原生的下拉控件，手机上原生select的下拉效果体验往往比较好，不管是IOS还是android，而使用

```
<input type="tel">
```

在手机上会弹一个电话号码的键盘，

```
<input type="number">
<input type="email">
```

都会弹相应的键盘

如果是分隔线就使用hr标签，而不是自己写一个border-bottom的样式，使用hr容易进行检查

如果是换行文本就应该使用p标签，而不是写br，因为p标签可以用margin设置行间距，但是如果是长文本的话使用div，因为p标签里面不能有p标签，特别是当数据是后端给的，可能会带有p标签，所以这时容器不能使用p标签。

24. 不要在https的链接里写http的图片

只要https的网页请求了一张http的图片，就会导致浏览器地址栏左边的小锁没有了，一般不要写死，写成根据当前域名的协议去加载，用//开头：

```
<img src="//static.chimeroi.com/hello-world.jpg">
```

二、CSS编码规范

1. 文件名规范

文件名建议用小写字母加中横线的方式。为什么呢？因为这样可读性比较强，看起来比较清爽，像链接也是用这样的方式，例如stackoverflow的url:

```
https://stackoverflow.com/questions/25704650/disable-blue-highlight-when-touch-press-object-with-cursorpointer
```

或者是github的地址：

```
https://github.com/wangjeaf/ckstyle-node
```

那为什么变量名不用小写字母加小划线的方式，如：family_tree，而是推荐用驼峰式的familyTree？C语言就喜欢用这种方式命名变量，但是由于因为下划线比较难敲(shift + -)，所以一般用驼峰式命名变量的居多。

引入CSS文件的link可以不用带type="text/css"，如下代码：

```
<link rel="stylesheet" href="test.css">
```

因为link里面最重要的是rel这个属性，可以不要type，但是不能没有rel。

JS也是同样道理，可以不用type，如下代码：

```
<script src="test.js"></script>
```

没有兼容性问题。

2. 属性书写顺序

属性的书写顺序对于浏览器来说没有区别，除了优先级覆盖之外。但是如果顺序保持一致的话，扫一眼可以很快地知道这个选择器有什么类型的属性影响了它，所以一般要把比较重要的属性放前面。比较建议的顺序是这样的：

你可能会觉得我平时差不多就是这么写的，那么说明你有一个比较好的素养。并且我觉得规则不是死，有时候可以灵活，就像你可能会用transform写居中，然后把left/top/transform挨在一起写了，我觉得这也是无可厚非的，因为这样可以让人一眼看出你要干嘛。

3. 不要使用样式特点命名

有些人可能喜欢用样式的特点命名，例如：

```
.red-font{
    color: red;
}
.p1{
    font-size: 18px;
}
.p2{
    font-size: 16px;
}
```

然后你在它的html里面就会看到套了大量的p1/p2/bold-font/right-wrap之类的类名，这种是不可取的，假设你搞了个red-font，下次UI要改颜色，那你写的这个类名就没用了，或者是在响应式里面在右边的排版在小屏的时候就会跑到下面去，那你取个right就没用了。

有些人先把UI整体瞄了一下，发现UI大概用了3种字号18px/16px/14px，于是写3个类p1/p2/p3，不同的字号就套不同的类。这乍一看，好像写得挺通用，但是当你看他的html时，你就疯掉了，这些p1/p2/p3的类加起写了有二三十个，密密麻麻的。我觉得如果要这样写的话还不如借助标题标签如下：

```
.house-info h2{
    font-size: 18px;
}
.house-info h3{
    font-size: 16px;
}
```

因为把它的字号加大了，很可能是一个标题，所以为什么不直接用标题标签，不能仅仅担心因为标题标签会有默认样式。

类的命名应当使用它所表示的逻辑意义，如signup-success-toast、request-demo、agent-portrait、 company-logo等等。

如果有些样式你觉得真的特别通用，那可以把它当作一个类，如clearfix，或者有些动画效果，有几个地方都要用到，我觉得这种较为复杂并且通用的可以单独作为一个类。但是还是趋向于使用意义命名。

4. 不要使用hack

有些人在写CSS的时候使用一些hack的方法注释，如下：

```
.agent-name{
    float: left;
    _margin: 10px;
    //padding: 20px;
}
```

这种方法的原理是由于//或者_开头的CSS属性浏览器不认识，于是就被忽略，分号是属性终止符，从//到分号的内容都被浏览器忽略，但是这种注释是不提倡的，要么把.css文件改成.less或者.scss文件，这样就可以愉快地用//注释了。

还有一些专门针对特定浏览器的hack，如*开头的属性是对IE6的hack。不管怎么样都不要使用hack。

5. 选择器的性能

选择器一般不要写超过3个，有些人写sass或者less喜欢套很多层，如下：

```
.listings-list{
    ul{
        li{
            .bed-bath{
                p{
                    color: #505050;
                }
            }
        }
    }
}
```

一个容器就套一层，一层一层地套下来，最底层套了七八层，这么长的选择器的性能比较差，因为Chrome里面是用递归从最后一个选择器一直匹配到第一个，选择器越多，匹配的时间就越长，所以时间会比较长，并且代码的可读性也比较差，为看到最里面的p标签的样式是哪个的我得一层层地往上看，看是哪里的p。代码里面缩进了7、8层看起来也比较累。

一般只要写两三个比较重要的选择器就好了，不用每个容器都写进去，重要的目标元素套上class或者id。

最后一个选择器的标签的应该少用，因为如果你写个.container div{}的话，那么页面上所有的div第一次都匹配中，因为它是从右往左匹配的，这样的写的好处是html不用套很多的类，但是扩展性不好，所以不要轻易这样用，如果要用需要仔细考虑，如果合适才使用，最起码不能滥用。

6. 避免选择器误选

有时候会出现自己的样式受到其他人样式的影响，或者自己的样式不小心影响了别人，有可能是因为类的命名和别人一样，还有可能是选择器写的范围太广，例如有人在他自己的页面写了：

```
* {
    box-sizing: border-box;
}
```

结果导致在他个页面的公用弹框样式挂了。一方面不要写*全局匹配选择器，不管从性能还是影响范围来说都太大了，例如在一个有3个子选择器的选择器：

```
.house-info .key-detail .location{}
```

在三个容器里面，*都是适用的，并且有些属性是会继承的，像font-size，会导致这三个容器都有font-size，然后一层层地覆盖。

还有一种情况是滥用了:first-child、:nth-of-type这种选择器，使用这种选择器的后果是扩展性不好，只要html改了，就会导致样式不管用了，或者影响到了其它无关元素。但是并不是说这种选择器就不能用，只要用得好还是挺方便的，例如说在所有的li里面要让最后一个li的margin-left小一点，那么可以这么写：

```
.listing-list li:last-child{
    margin-left: 10px;
}
```

这可能比你直接套一个类强。但是不管怎么样，不能滥用，合适的时候才使用，而不是仅仅为了少写类名。

7. 减少覆盖
   覆盖是一种常用的策略，也是一种不太优雅的方式，如下代码，为了让每个house中间的20px的间距，但是第一个house不要有间距：

```
.house{
    margin-top: 20px;
}
.house:first-child{
    margin-top: 0;
}
```

其实可以改成这样：

```
.house + .house{
    margin-top: 20px;
}
```

只有前面有.house的.house才能命中这个选择器，由于第一个.house前面没有，所以命不中，这样看起来代码就简洁多了。

还有这种情况，如下代码所示：

```
.request-demo input{
    border: 1px solid #282828;
}
.request-demo input[type=submit]{
    border: none;
}
```

其实可以借助一个:not选择器：

```
.request-demo input:not([type=sbumit]){
    border: 1px solid #282828;
}
```

这样看起来代码也优雅了很多。

有一种覆盖是值得的，那就是响应式里面小屏的样式覆盖大屏，如下：

```
.container{
    width: 1080px;
    margin: 0 auto;
}
@media (min-width: 1024px){
    .container{
        width: auto;
        margin: 0 40px;
    }
}
```

大屏的样式也可以写成：

```
@media (min-width: 1025px){
     .container{
         width: 1080px;
         margin: 0 auto;
    }
}
```

我一开始是就是这么写的，为了遵循减少覆盖原则，但是后来发现这种实践不好，代码容易乱，写成覆盖的好处在于可以在浏览器明显地看到，小屏的样式是覆盖了哪个大屏的样式，这个在大屏的时候又是怎么样的。

8. 使用CSS3的选择器完成一些高级的功能

上面提到:not可以让代码简洁，还有其它一些很好用的。例如说只有两个的时候一个占比50%,而有3个的时候一个占比33%，这个用CSS就可以实现，如下：

```
.listing-list li{
    width: 33%;
}
.listing-list li:first-child:nth-last-child(2),
.listing-list li:first-child:nth-last-child(2) ~ li{
     width: 50%;
}
```

当li是第一个元素并且是倒数第二个元素的时候以及和它相邻的li被第二组选择器命中，它的宽度是50%，也就是只有两个li的时候才能满足这个条件。

另外还可以借用:hover/:focus/:invalid/:disabled等伪类选择器完成一些简单的交互。

9. 少用!important

important用来覆盖属性，特别是在CSS里面用来覆盖style里的属性，但是important还是少用为妙。

有时候你为了偷懒直接写个!important，以为这个的优先级是最高的了，往往螳螂捕蝉，黄雀在后，很可能过不了多久又要再写一个优先级更高的覆盖掉，这样就略显尴尬了。

所以能少用还是少用。如果要覆盖还是先通过增加添加选择器权重的方式。

10. 多写注释

"程序猿最烦两件事，第一件事是别人要他给自己的代码写文档，第二件呢？是别人的程序没有留下文档"。注释也是同样道理，当看到很多绿色的注释代码神经会比较放松，而当看到揉成一团还没有注释的代码是比较压抑的。CSS的注释可包括：

（1）文件顶部的注释

```
/*
 * @description整个列表页样式入口文件
 * @author yincheng.li
 */
```

（2）模块的注释

```
/*详情页贷款计算器*/
```

（3）简单注释

```
/*为了去除输入框和表单点击时的灰色背景*/
input,
form{
    -webkit-tap-highlight-color:  rgba(255, 255, 255, 0);
    }
```

（4）TODO的注释 有时候你看源码的时候你会看到一些TODO的注释：

```
/* TODO(littledan): Computed properties don't work yet in nosnap.
   Rephrase when they do.
*/
```

表示这些代码还有待完善，或者有些缺陷需要以后修复。而这种TODO的注释一般编辑器会把TODO高亮。

注意不要写一些错误的误导的注释或者比较废话的注释，这种还不如不写，如下：

```
/* 标题的字号要大一点 */
.listings h2{
    font-size: 20px;
}
```

11. 排版规范

不管是JS/CSS，缩进都调成4个空格，如果你用的sublime，在软件的右下角有一个Tab Size，选择Tab Size 4，然后再把最上面的Indent Using Spaces勾上。

这样，当你打一个tab键缩进的时候就会自动转换成4个空格。如果你使用vim，新增或者编辑~/.vimrc文件新增一行：

```
:set tabstop=4
```

也会自动把缩进改成4个空格，其它编辑器自行查找设置方法。因为\t在不同的编辑器上显示长度不一样，而改成空格可以在不同人的电脑上格式保持一致。

多个选择器共用一个样式集，每个选择器要各占一行，如下：

```
.landing-pop,
.samll-pop-outer,
.signup-success{
    display: none;
}
```

每个属性名字冒号后面要带个空格，~、>、+选择器的前后也要带一个空格：

```
.listings > li{
    float: left;
}
```

12. 属性值规范

（1）如果值是0，通常都不用带单位 例如：

```
.list{
    border: 1px solid 0px;
    margin: 0px;
}
```

应改成：

```
.list{
    border: 1px solid 0;
    margin: 0;
}
```

但是有个特例，就是和时间有关的时间单位都要带上秒s，如下两个都是不合法的：

```
transition-duration: 0;
transition: transform 0 linear;
```

（2）色值用十六进制，少用rgb 如下：

```
color: rgb(80, 80, 80);
```

应改成：

```
color: #505050;
```

因为使用rgb是一个函数，它还要计算一下转换。如果是带有透明度的再用rgba.

如果色值的六个数字一样，那么写3个就好：

```
color: #ccc;
```

（3）注意border none和0的区别 如下两个意思一样：

```
border: 0;
border-width: 0;
```

而下面这两个一样：

```
border: none;
border-style: none;
```

所以用0和none都可以去掉边框。

你可能会说打包工具其实最后会帮我处理，但自己要保持一个良好的书写习惯还是很重要的。

13. font-family的设置

注意使用系统字体的对应的font-family名称，如SFUIText Font这个字体，在Safari是-apple-system，而在Chrome是BlinkMacSystemFont，所以font-family可以这么写：

```
font-family{
    font-family: -apple-system, BlinkMacSystemFont, sans-serif;
}
```

再如微软雅黑，很多中文网站都用这个字体，要写成：

```
font-family{
    font-family: Microsoft YaHei;
}
```

另外font-family不能在代码任意设置，如果使用了自定义字体。如下代码：

```
.title{
    font-family: Lato Bold;
}
```

因为如果你在代码里面写了好多个font-family，到时候要整体替换网页的字体就很麻烦了，正确的做法应该是这样的：

```
h1,
strong,
b{
    font-family: Lato Bold;
    font-weight: normal;
}
```

如果需要加粗就用标题标签，或者b/strong标签，并且要把font-weight调回来，因为那个字体本身就有加粗效果了。

如果font-weight再是粗体的话浏览器会用自己的算法继续加粗。如果是细体怎么办，一方面一般细体用得比较少，另一方面没有细体的标签，可以通过套类的方式。

14. 不要设置太大的z-index

有些人喜欢设置z-index很大：

```
z-index: 99999;
```

以为他是老大了，不会有人再比他高了，但是螳螂捕蝉，黄雀在后，很快得再写一个：

```
z-index: 999999;
```

通常自己页面的业务逻辑的z-index应该保持在个位数就好了。

15. 合并属性

一般的说法是说为了提高性能，属性要合并，但其实Chrome每个属性都是单独的，就算你合在一起，它也会帮你拆出来。

如把margin拆成left/right/top/bottom，但是我们还是推荐写成合的，因为它可以让代码看起来更简洁，代码量更少，如下代码：

```
.container{
    margin-top: 20px;
    margin-left: 10px;
    margin-right: 10px;
}
```

可以写成：

```
.container{
    margin: 20px 10px 0;
}
```

但是合在一起写了，要注意别覆盖了其它的设置，如上面把margin-bottom设置成了0.

再如：

```
.banner{
    background-image: url(/test.jpg);
    background-position: 0 0;
    background-repeat: no-repeat;
}
```

可以改成：

```
.banner{
    background: url(test.jpg) 0 0 no-repeat;
}
```

16. 注意float/absolute/fixed定位会强制设置成block
    如下代码：

```
a.btn {
    float: left;
    display: block;
    width: 100px;
    height: 30px;
}
```

第二行的display: block其实是没用的，因为如果你浮动了，目标元素就会具有块级盒模型的特性，即使你

```
display: table-cell或者inline也不管用。如果你是display: flex，那么float将会被忽略。
```

同样地，absolute定位和fixed定位也有同样的效果，会把行内元素变成块级的。

17. 清除浮动

清除浮动有多种方法，一般用clearfix大法，虽然这个方法有缺陷，但是它比较简单且能够适用绝大多数的场景，一个兼容IE8及以上的clearfix的写法：

```
.clearfix:after{
    content: "";
    display: table;
    clear: both;
}
```

就不要在末尾添加一个多余元素的方法清除浮动了，虽然也可行，但是比较low.

18. 引号的使用

- font-family

一般来说font-family不需要添加引号，即使字体名称带有空格也没关系，但是有一种情况是一定要加上引号的，就是字体名称刚好是关键词，如下字体都需要加关键词：

```
font-family: "inherit", "serif",
"sans-serif", "monospace", "fantasy", and "cursive"
```

- background的url

```
background-url: url("//cdn.test.me/test.jpg");
```

你不加也可以，但是有一种一定要加，那就是url里面带有特殊字符没有转义，如下：

```
background-url: url(//cdn.test.me/hello world.jpg)
```

上面浏览器会去加载//cdn.test.me/hello，然后报404。这种情况通常是图片是用户上传的，图片的名字带有空格，后端给的url没有对特殊字符做处理，就会有问题，所以当url是可变的时候，最好还是带上引号：

```
background-url: url('//cdn.test.me/hello world.jpg');
```

这样浏览器就能正常加载图片了。这种情况最好的还是从源头上避免，但我们也可以做个兼容。

- 单引号还是双引号

这两个都是合法的，只是统一一下比较好，不能一下子单引号，一下子双引号的，比较普遍的推荐是html使用双引号，css使用单引号。

19. CSS动画规范

（1）不要使用all属性做动画

使用transition做动画的时候不要使用all所有属性，在有一些浏览器上面可能会有一些问题，如下：

```
transition: all 2s linear;
```

在Safari上面可能会有一些奇怪的抖动，正确的做法是要用哪个属性做动画就写哪个，如果有多个就用隔开，如下代码所示：

```
transition: transform 2s linear,
             opacity 2s linear;
```

（2）使用transform替代position做动画 如果能用transform做动画的，就不会使用left/top/margin等。

因为transform不会造成重绘，性能要比position那些高很多，特别是在移动端的时候效果比较明显。基本上位移的动画都能用transform完成，不需要使用CSS2的属性，如一个框从右到左弹出。

（3）偏向于使用CSS动画替代JS动画 例如把一个框，从下到上弹出，可以用jQuery的slideUp函数，或者自己写setInterval函数处理。

但是这些没有比用CSS来得好。使用CSS，初始状态可以把框translate移动屏幕外，然后点击的时候加上一个类，这个类的transform值为0，然后再用transition做动画就好了。

20. 不要断词

英文的单词或者数字如果当前行排不下会自动切到下一行，这样就导致每行长短不一，有时候可能不太美观，但是不能使用word-break: break-all把一个单词拆成两行，还有一种是使用：

```
hyphens: auto;
```

它会把单词拆成用-连接的形式，看起来好像挺合理，但是由于它断词断得不够彻底，有些单词断不了，长短不一的现象看起来也比较明显，有些单词还被拆成了两行，所以还不如不加。因此，不要使用断词。

21. 不要设置图标字体font-family

这个和上面提到的font-family设置是一样的，不要在代码里面手动设置font-family，如下：

```
.icon-up:before{
    content: "\e950";
    font-family: "icon-font";
}
```

正确的做法是给.icon-up的元素再套一个.icon的类，font-family等对图标字体的相关设置都统一在这个类里面：

```
.icon{
    font-family: "icon-font";
    /* Better Font Rendering =========== */
    -webkit-font-smoothing: antialiased;
    -moz-osx-font-smoothing: grayscale;
}
```

因为我们可能会添加其它一些设置，有个.icon的类统一处理比较好。就不要手动一个个去设置font-family了。

22. 设置常见样式reset

由于每个浏览器都有自己的UA样式，并且这些样式还不太统一，所以需要做样式reset，常见的reset有以下：

```
/* IE浏览器对输入控件有自己的font-family，需要统一 */
input,
textarea,
button{
    font-family: inherit;
}

/* Chrome浏览器会在输入控制聚集的时候添加一个蓝色的outline*/
input:focus,
textarea:focus,
select:focus{
    outline: none;
}

/* 去掉textarea的可拉大小功能*/
textarea{
    resize: none;
}

/* IOS Safari在横屏的时候会放大字体，第二个属性让滑动更流畅 */
html{
    -webkit-text-size-adjust: 100%;
    -webkit-overflow-scrolling : touch;
}

/* 统一标签的margin值和p标签的line-height*/
body, p, h1, h2, ul, ol, figure, li{
    padding: 0;
    margin: 0;
}

h1, h2, p{
    line-height: 150%;
```

```
}

/* 去掉select的默认样式 */
select{
    -webkit-appearance: none;
}
/* 如果有输入内容IE会给输入框右边加一个大大的x */
input::-ms-clear{
    display: none;
    width: 0;
    height: 0;
}

/* 去掉number输入框右边点击上下的小三角 */
input::-webkit-inner-spin-button{
    -webkit-appearance: none;
}

input::-webki-outer-spin-button{
    -webki-appearance: none;
}
```

23. 图片压缩

不管是UI直接给的图片还是自己从UI图里切出来的图片，都需要把图片压缩一下，建议使用tinypng，它可以在保持图片质量减少较低的情况下，把图片压得很厉害，比直接在PS里面设置压缩质量要强。

如果是色彩比较丰富的图片要使用jpg格式，不能使用png格式，png会大得多，如果是logo那种矢量图片，直接使用svg格式即可。一般来说要把图片控制在300k以内，特别是banner头图，图片的大小也要控制住。

24. 正确使用background和img

显示一张图片有两种方式，可以通过设置CSS的background-image，或者是使用img标签，究竟什么时候用哪种呢?

如果是头图等直接展示的图片还是要img标签，如果是做为背景图就使用background，因为使用img可以写个alt属性增强SEO，而背景图那种本身不需要SEO。

虽然background有一个一个background-position: center center很好，但是头图那种还是使用img吧，自己去居中吧，不然做不了SEO。

25. 响应式的规范

响应式开发最不好不要杂合使用rem，文字字号要么全部使用rem，要么不要用，也不要使用transform: scale去缩小。

因为被缩小的字号看起来会有点奇怪，别人都是14px，而你变成了13.231px，小了一点。响应式的原则一般是保持中间或者两边间距不变，然后缩小主体内容的宽度。

26. 适当使用:before/:after

:before和:after可以用来画页面的一些视觉上的辅助性元素，如三角形、短的分隔线、短竖线等，可以减少页面上没有用的标签。但是页面上正常的文本等元素还是不要用before/after画了。

27. 少用absolute定位

首先absolute定位的元素渲染性能会比较高，因为它独立出来了，计算量会少，用得好还是可以的。

但是如果你页面的主要布局是使用absolute的那肯定是不可取的，因为absolute定位的可扩展性很差，你把每个元素的位置都定死了就变不了了，可以多用float。

虽然float的性能相对较差，但是不管是实用性还是兼容性都是挺好的。

28. 少用inline-block布局

有些人喜欢用inline-block，特别是刚开始学切图的人，因为block会换行，而inline-block不会换行还具有盒模型，因此inline-block用得很顺手，而float比较复杂，还要处理清除浮动之类的问题。如下布局：

应该写li，然后让li float，如果你让li display:inline-block也可以达到目的。

但是inline-block用得多了可能会有一些奇怪的问题，你通常要在一个inline-block的元素里面套block的元素，inline-block是行内元素，而block是块级元素，这两者终究有点差别。

这种应该用float/flex会更自然，如果你float用顺手了你会发现比inline-block好多了，并且更加专业。

如果你没怎么用过flex ，那你可以尝试换一下使用flex，如果你没怎么用过float，可以尝试用一下。只有你的切图方式多样化了，你切起图来才能比较灵活。

29. 图片的居中和宽高设定

一般来说，UI给的图片展示宽高是固定的，但是实际的图片长宽是不固定，大部分图片是长是比宽大，小部分图片是宽比长大。所以需要居中裁剪展示，如下图所示：

中间黑色的框是展示区域，图片的短边和窗器的边一样大，另一边按图片的原始比例拉伸，然后居中显示。这个得借助JS，因为图片未加载好之前，不知道是长边比较大还是宽比较大。如下代码：

```
<div class="img-container">
    <img src="test.jpg" alt onload="resizeImg(this, '400px', '300px'">
</div>
```

借助一个resizeImg函数，在onload函数里面做处理。然后居中用CSS：

```
.img-container{
    position: relative;
    width: 400px;
    height: 300px;
}
.img-container img{
    position: absolute;
    left: -9999px;
    right: -9999px;
    top: -9999px;
    bottom: -9999px;
    margin: auto;
}
```

上面代码用了一个margin: auto做居中。

30. 移动端提高可点区域范围

移动端的的一些图标如X，可能会设计得比较小，所以点起来会不太好点，因此要提高可点区域范围，可通过增加padding。

如下代码：

```
.icon-close{
    position: abosulte;
    right: 0;
    top: 0;
    padding: 20px;
}
```

这样区域就增加了一圈，点起来就容易多了。

31. 不要设置input的line-height

如果设置input的line-height，如下代码，你可能要做垂直居中：

```
.request-demo input{
    height: 40px;
    line-height: 40px;
}
```

设置了line-height为一个很高的值，这样会导致Safari浏览器的输入光标|变得巨大，所以如果你要居中的话，使用padding吧。

32. 移动端弹框要禁止body滑动

因为IOS Safari切换输入框的时候会页面会弹闪得很厉害，因为你在切的时候它会先把键盘收起来，然后再弹出来。

这个时间很短，给人感觉页面弹闪了一下，但如果把body禁止滑动了就不会有这个问题。

这有两个解决办法，第一种是把body fixed住，第二种设置body overflow: hidden，相对来说第二种比较简单一点。IOS10完全不会闪，IOS9以下还是会闪。

33. 对于渐变的处理

有时候UI里面会有一些渐变的效果，无法复制CSS出来，这个时候可以用一个在线的工具，生成渐变的CSS：

www.cssmatic.com/gradient-ge...，但是这个需要自己手动调一个和UI一模一样的效果，或者可以直接给UI调一个它理想的效果，它会生成兼容性很强的CSS：

```
background: #fff;
background: -moz-linear-gradient(left, #fff 0%, #d2d2d2 43%, #d1d1d1 58%,
#fefefe 100%);
background: -webkit-gradient(left top, right top, color-stop(0%, #fff),
color-stop(43%, #d2d2d2), color-stop(58%, #d1d1d1), color-stop(100%,
#fefefe));
background: -webkit-linear-gradient(left, #fff 0%, #d2d2d2 43%, #d1d1d1
58%, #fefefe 100%);
background: -o-linear-gradient(left, #fff 0%, #d2d2d2 43%, #d1d1d1 58%,
#fefefe 100%);
background: -ms-linear-gradient(left, #fff 0%, #d2d2d2 43%, #d1d1d1 58%,
#fefefe 100%);
background: linear-gradient(to right, #fff 0%, #d2d2d2 43%, #d1d1d1 58%,
#fefefe 100%);
filter: progid:DXImageTransform.Microsoft.gradient( startColorstr='#fff',
endColorstr='#fefefe', GradientType=1 );
```

34. 行内元素可以直接设置

```
margin-left/margin-right
```

如下有些人为了把span撑开，设置span display: inline-block：

```
span.phone-numer{
    display: inline-block;
    margin-left: 10px;
}
```

其实行内元素可直接margin的左右，能够把它撑开，不需要设置inline-block：

```
span.phone-numer{
    margin-left: 10px;
}
```

另外需要注意的是img/input/textarea/button默认就是inline-block，也不用再设置。

三、JS编码规范

1. 变量命名

《代码大全》这本书里面有一章是专门讲变量命名的，这里结合这本书的建议做说明。总地来说，变量名要准确完整地描述该变量所表述的事物，具体来说：

（1）变量名不应以短巧为荣 如以下好的变量名和不好的变量名：

不好的变量名 好的变量名

```
inp input, priceInput
day1, day2, param1  today, tomorrow
id  userId, orderId
obj orderData, houseInfos
tId removeMsgTimerId
handler submitHandler, searchHandler
```

左边的变量名都不太清楚，代码的扩展性不好，一旦代码需要加功能的话，就容易出现obj1、obj2、obj3这种很抽象的命名方式。所以一开始就要把变量的名字起得真实有意义，不要搞一些很短很通用的名字。

当然变量名取得太长也不好，如maximumNumberOfTeamMembers.

（2）变量名不要使用计算机术语 变量名应直指问题领域，来源于现实世界，而不是计算机世界，例如取了texareaData之类的名字，应该取一个和业务相关的名字，如leaveMsg.

（3）变量名的对仗要明确 如up/down、begin/end、opened/closed、visible/invisible、scource/target，对仗明确可以让人很清楚地知道两个变量的意义和用途。

（4）警惕临时变量 有些喜欢取temp和obj之类的变量，如果这种临时变量在两行代码内就用完了，接下来的代码就不会再用了，还是可以接受的，如交换数组的两个元素。

但是有些人取了个temp，接下来十几行代码都用到了这个temp，这个就让人很困惑了。所以应该尽量少用temp类的变量。

如下代码：

```
var temp = 10;
var leftPosition = currentPosition + temp,
    topPosition = currentPosition - temp;
```

应改成：

```
var adjustSpace = 10;
var leftPosition = currentPosition + adjustSpace,
     topPosition = currentPosition - adjustSpace;
```

（5）bool变量 《代码大全》这本书建议布尔变量不用以is/do之类的开头，如：

```
var isMobile = true,
    isError = true,
    doUpdate = false;
```

可改成：

```
var mobile = true,
    error = true,
    updated = false;
```

还有其它一些常用的名称如done/found/successs/ok/available/complete等，结合具体的语境：

```
var ajaxDone = true,
    fileFound = false,
    resourceUpdated = true;
```

另外变量名不要使用否定的名词，如notOk，notReady，因为否定的词取反的时候就会比较奇怪，如if(!notOk). 要使用肯定的布尔变量名。如果是参数的话可结合ES6的默认形参值。

（6）变量名使用正确的语法

不要使用中文拼音，如shijianchuo应改成timestamp，如果是复数的话加s，或者加上List，如orderList、menuItems，而过去式的加上ed，如updated/found等，如果正在进行的加上ing，如calling.

2. 声明变量时要赋值

如下声明三个变量：

```
var registerForm,
     question,
     calculateResult;
```

以上绝对是合法JS语法，但是这三个变量的用途会让人比较困惑，特别是中间第二个question，问题是什么。但是当你把上面的变量赋一个初始值的时候：

```
var registerForm = null,
     question = "",
     calculateResult = 0;
```

就让人豁然开朗了，原来question是一个问题的字符串，而result是一个数字，form是一个对象。

这也有利于JS解释器提前做一些优化处理，不用等到使用的时候才知道这些变量是什么类型的。

3. 函数的返回值类型要确定

如下代码：

```
function calculatePrice(seatCount){
    if (seatCount <= 0) {
        return "";
    } else {
        return seatCount * 79;
    }
}
```

这个代码可能返回整型，也有可能返回字符串，就会让人比较困惑，同时从代码性能来说也是不高的，虽然它是合法的JS语法，一个函数的返回类型要统一。

你可能会说我用上面的函数做为输入框显示的值，如果是负数或者0，那么输入框就不要显示任何东西，所以才会返回空的字符串。

即使是这样的原因也不建议这样写，从长远来看这样写是不利的，你应该用其它的方法组织你的代码。要养成强类型的代码风格，这样不容易出bug，扩展也容易。

另外如果一个变量你把它当成数字使用，下面就不要再把它当成字符串使用了，因为这样也容易让人困惑。

微软的Typescript就是一种强类型的书写语法，很多大型项目会使用typescript写JS，有兴趣的可以继续了解怎么写typescript.

4. 不要给变量赋值undefined

undefined表示一个变量未定义，你定义了一个变量又说它未定义本身就很奇怪。

这可能会造成的问题是使用上的歧义，因为我们经常使用undefined来判断变量有没有定义：

```
if (typeof window.HTMLVideoElement === "undefined")
```

如果要赋值应该要赋空值，如对象赋值为null，数字赋值为0，字符串赋值为空字符，那你可能会说0也是一个正常的数字。

如果赋值为0会导致我误认为它是一个正常的数据，那怎么办呢？如果你的数字都是非负数，那么可以把初始值置为-1，实在不行就置成NaN.

函数的返回值也不要显式地return undefined.

5. 排版规范

一个比较流行的空格和缩进排版如下代码所示：

```
//逗号后面带个空格，) {中间带个空格
function getSeatDiscount(seatCount, currentPrice) {
    //双目运算符左右两边都带个空格
    var originPrice = editOrder.getSeatsPrice(seatCount);
    return Math.round((originPrice - currentPrice) / originPrice * 100);
}
```

一行太长要换行，如V8的源码里面一行最长是70个字符，超过就换行：

```
function ArrayUnshift(arg1) {  // length == 1
//if判断里面进行了换行，并且if (中间带个空格
  if (len > 0 && UseSparseVariant(array, len, IS_ARRAY(array), len) &&
      !%object_is_sealed(array)) {
    SparseMove(array, 0, 0, len, num_arguments);
  } else {
    SimpleMove(array, 0, 0, len, num_arguments);
  }
}
```

一行代码太长了就换行是一种好的习惯，太长让人看起来比较费劲。基本上一行不要超过100个字符，超过就要换行，不管是注释还是代码。

6. 使用===代替==

==会带上类型转换，这和上面一样的，我们要用强类型的风格写代码，所以不要使用==，如果有类型转换自己做类型转换，不要让别人去猜这里面有类型转换，使用==会有一些比较奇怪的结果：

```
null == undefined           //true
'' == '0'                   //false
0  == ''                    //true
0  == '0'                   //true
' \t\r\n ' == 0             //true
new String("abc") == "abc" //true
new Boolean(true) == true  //true
true == 1                   //true
```

7. 减少魔数

对一些比较重要的常量起一个名字，例如下面的代码：

```
const ONE_DATE = 3600 * 24 * 1000;
var tomorrow = today + ONE_DATE;
```

再如下面不好的写法：

```
dialogHandler.showQuestionNaire ("seller", "sell", 5, true) ;
```

上面四个常量会让人看起来比较困惑，如果可以的话给它们起个名字，如果觉得麻烦那就加上注释。

8. 不要让代码暴露在全局作用域下运行

一个原因是在全局作用域下，变量的查找时间会更长，第二个原因是污染全局作用域，有时候会造成一些意想不到的结果，如下：

```
var name = "hi boy";
console.log(window.name);
```

定义了一个变量，但是刚好不巧window.name是本来有这个属性，这个属性通常用来跨域传递数据。如果你设置了name这个变量，就把全局的window.name给覆盖了。

9. let/var/const的使用

ES6新增了let/const定义变量，使用let有一些好处，如：

（1）避免变量重复定义

```
let me = "go";
// Uncaught SyntaxError: Identifier 'me' has already been declared
let me = "go";
```

使用babel loader打包的时候它会做静态检查：

```
Module build failed: Duplicate declaration "me"
```

（2）for循环的变量作用域是独立的

```
for(let i = 0; i <= 4; i++) {
    tasks.push(function(){
        console.log("i is " + i);
    });
}
```

使用let使得i在for循环里面每次运行的作用域都是独立的。并且for里定义的变量在for循环外是不可见的。

babel在转换的时候，会在for循环里面套一个function，然后把i当作函数的参数:

```
var _loop = function _loop(_i) {
    tasks.push(function () {
        console.log("i is " + _i);
    });
};

for (var _i = 0; _i <= 4; _i++) {
    _loop(_i);
}
```

由于let可以避免变量重复定义，就冲着这一点，就使得它很有意义。所以推荐多用let定义变量。所以本规范下面的变量将使用let代替var.

而const适合于给常量起个名字，如上面提到的：

```
const ONE_DAY = 3600 * 24 * 1000;
const adjustSpace = 10;
```

或者是定义其它一些不需要修改的变量，防止不小心被其它代码修改了。

10. 简洁代码

（1）使用三目运算代替简单的if-else 可以写一行就不要写三行，如下：

```
let seatDiscount = 100;
if(seat < 5) {
    seatDiscount = 90;
} else if(seat < 10) {
    seatDiscount = 80;
} else {
    seatDiscount = 70;
}
```

可以改成三目运算符：

```
let seatDiscount = seat < 5 ? 90 :
                           seat < 10 ? 80 : 70;
```

代码从8行减少到了2行。

（2）使用箭头函数取代简单的函数 例如以下代码：

```
setTimeout(function(){
    window.location.reload(true);
}, 2000);
```

可改成：

```
setTimeout(() => window.location.reload(true), 2000);
```

代码从3行变成了1行。

11. 注意避免执行过长时间的JS代码

对于一般的页面的数据量来说，加减乘除等计算不足以造成性能瓶颈。容易造成瓶颈的是DOM操作，特别是大批量的DOM操作，只要一次有几百上千的级别就容易造成页面卡顿。特别是不要在一个for循环里不断地修改DOM，如下代码：

```
for(var i = 0; i < 1000; i++) {
    ul.appendChild(li);
}
```

这种可以先把li拼好了，再一次性append到ul里面，如下代码：

```
var fragment = document.createDocumentFragment();
for(var i = 0; i < 1000; i++) {
    fragment.appendChild(li);
}
ul.appendChild(fragment);
```

如果你用jq的话应该先把模板渲染好，然后再一次性append到dom里面，而不是不断地append到dom里面。现在的浏览器一般也比较智能，它会做一些优化，但是我们不能老是指望浏览器会优化。

但是还是要注意数据量特别大的情况，你可能要使用setTimeout的方式分段处理数据，甚至使用多线程。使用setTimeout可以这样：

```
function sliceWorks(data, finishedCallback) {
    if(!data.length) {
        finishedCallback();
    } else {
        const PIECES = 100;
        process(data.splice(0, PIECES));
        setTimeout(() => sliceWorks(data, finishedCallback), 100);
    }
}
```

我们使用一个递归，把数据分段处理，每段100个，当数据处理完再调完成回调函数。

12. 多写注释

这个和CSS规范类似：

- 文件顶部的注释，包括描述、作者、更新

```
/*
 * @file listing-detail.js
 * @description 房源详情页的JS主文件，处理轮播、房贷计算器、约看房等逻辑
 * @author yincheng.li
 * @update (yincheng.li 2017/8/19)
 */
```

- 函数的注释

```
/*
 * 和搜索界面展示有关的处理逻辑
 * @namespace
 */
```

```
var searchWinHandler = {
    /*
     * 初始化驱动函数
     *
     * @param {bool} realTimeSearch 是否需要进行实时搜索
     * @param {HTMLFormElement} form 搜索表单DOM元素
     *
     */
    init(realTimeSearch, HTMLFormElement){

    }

    /*
     * 搜索条件展示点击x按钮的处理函数
     *
     * @param {object} jquery的点击事件event
     * @trigger 会触发search按钮的点击事件，以触发搜索
     * @returns 无返回
     *
     * TODO 这里临时使用了一个全局变量的flag，这种实现方式不太好
     * 虽然比较方便
     */
    closeFilterSpan(event){

    }

};
```

上面的@auhor @return都是注释标签，其它常用的注释标签还有：

```
/*
@class 表示一个类
@constructor 构造函数
@deprecated 被弃用
@global 全局的变量
@namespace 具有命名空间作用的object，如$.fn.remove，$.fn.append，$和fn就是一个
namespace，而fn是$的子命名空间
@this 这里的this指向哪里
@throws 在这个函数里面可能会抛出什么异常
@version 当前版本
*/
```

（3）变量定义和代码的注释 对一些比较重要的变量加注释，标明它是什么用途，以及对一些核心代码逻辑加上注释，或者比较复杂的业务逻辑，写了5个case，每个case分别代表什么。

为了改某个bug而加入的代码，说明下为了解决什么问题；还有某些易混的判断，为什么if判断条件写了四个，为什么代码到这个if判断不通过就直接return了。

一些常量的注释，为什么会突然冒出来100这个数字；改动了别人的代码，为什么要改动；等等。

如：

```
 var requestData = {
        listingId: listingData.listingId,
        page: 1,
        //把200改成5，点击More的时候是重新刷新页面的，也没有其他地方用到，
        //没必要请求那么多，严重影响性能
        pageSize: 5//200
};
```

总之多写注释还是好的，只要不是废话：

//定义了一个number的变量

```
let number = 5;
```

或者是和逻辑不符合的错误注释。

还有一种排版的注释，右括号的对应关系：

```
            } //function ajax
        } //switch(b)
    } //if(a)
}  //searchHandler
```

主要是为了方便在后面加代码，例如我要在switch(b)后面加代码，当我看到这个注释我就很清楚地知道需要在哪里按回车。不过一般不推荐嵌套很深的代码，或者写得很长，一个函数几百行。

13. 代码不要嵌套太深

有些人的代码经常会套个七八层，以jq代码为例，如下：

```
var orderHandler = {
    bindEvent: function(){
        $(".update-order").on("click", function(){
            if(orderStatus === "active"){
                ajax({
                    url: "/update-order",
                    success: function(data){
                        for(let i = 0; i < data.orders.length; i++){
                            dom.append();
                        }
                    }
                });
            } else {
                ajax({
                    url: "/create-order",
                    success: function(data){

                    }
                });
            }
        });
    }
};
```

上面的代码最深的一层缩进了八层，你可能会觉得这样逻辑挺清晰的啊，但是这种写法同时也有点面条式。以上代码如果让我写，我会这么组织：

```
var orderHandler = {
    sendUpdateOrderReq: function(requestUrl, successCallback){
        ajax({
            url: requestUrl,
            success: successCallback;
        });
    },
    updateOrder: function(event){
        let requestUrl = orderStatus === "active" ? "/update-order"
                                : "create-order";
        //更新订单回调函数
        let activeUpdateCallback = function(data){
            for(var i = 0; i < data.orders.length; i++){
                console.log(data.orders[i].id);
```

```
            }
        };
        //创建订单回调函数
        let inactiveUpdateCallback = function(data){

        };

        let successCallback = {
            active: activeUpdateCallback,
            inactive: inactiveUpdateCallback
        };
        //发请求处理订单
        searchHandler.sendUpdateOrderReq(requestUrl,

  successCallback[orderStatus]);
    },
    bindEvent: function(){
        $(".update-order").on("click", searchHandler.updateOrder);
    }

};
```

首先把绑定的匿名函数改成有名的函数，这样有个好处，当你想要off掉的时候随时可off掉，然后可以减少一层缩进，接着把根据orderStatus不同的回调先用变量判断好，而不是同时积压到后面再一起处理。

再把发送请求的函数再单独抽出来做为一个函数，这样可以减少两层缩进。上面最深的缩进为4层，减少了一半。并且你会发现这样写代码逻辑会更加清晰，我在bindEvent里面扫一眼就可以知道哪些DOM绑了哪些事件，然后我对如对哪个DOM的事件感兴趣再跳到相应的回调函数去看，而不用拉了一两页才在bindEvent里面找到目标DOM。

并且把updateOrder单独做为一个独立的函数，其它地方如果需要也可以使用，例如可能还有一个组合功能的操作可能会用到。

另外把ajax再做一层抽象主要是这个东西实在是太常用，让人一眼就知道要干嘛，把它分离到另外一个地方可以让具体的业务代码更加简单，例如上面发请求，我把回调函数准备好之后，只要执行一行代码就好了。

你缩进太多层，一行就被空格占掉了三、四十个字符，感观上就不是很好，还会出现上面提到的，最后面要写好多个右括号收尾的情况，并且一个函数动不动就两、三百行。

14. jQuery编码规范

如果你使用了jQuery。

（1）使用closest代替parent 尽量不要使用parent去获取DOM元素，如下代码：

```
var $activeRows = $this.parent().parent().children(".active");
```

这样的代码扩展性不好，一旦DOM结构发生改变，这里的逻辑分分钟会挂，如某天你可能会套了个div用来清除浮动，但是没想到导致有个按钮点不了了就坑爹了。

应该用closest，如：

```
var $activeRows = $this.closest(".order-list").find(".active");
```

直接定位和目标元素的最近共同祖先节点，然后find一下目标元素就好了，这样就不会出现上面的问题，只要容器的类没有变。如果你需要处理非自己的相邻元素，可以这么搞：

```
$this.closest("li").siblings("li.active").removeClass("active");
$this.addClass("active");
```

有时候你可以先把所有的li都置成某个类，然后再把自己改回去也是可取的，因为浏览器会进行优化，不会一见到DOM操作就立刻执行。

会先排成一个队列，然后再一起处理，所以实际的DOM操作对自己先加一个类然后再去掉的正负相抵操作很可能是不会执行的。

（2）选择器的性能问题 如下代码：

```
$(".page ul").addClass("shown");
$(".page .page-number").text(number);
$(".page .page-next").removeClass("active");
```

上面的代码做了三个全局查找，其实可以优化一下：

```
var $page = $(".page");
$page.find("ul").addClass("shown");
$page.find(".page-number").text(number);
$page.find(".page-next").removeClass("active");
```

先做一个全局查找，后续的查DOM都缩小到$page的范围，$page的节点只有几十个，在几个里面找就比在document几百几千个节点里面查找要快多了。jQuery的查DOM也是用的querySelectorAll，这个函数除了用在document之外，可用在其它DOM结点。

（3）on事件之前需要的时候才off 有些人喜欢在绑事件之前先off掉，这样感觉可以确保万无一失，但是如果你绑的事件是匿名的，你很可能会把其它JS文件绑的一起off掉了，并且这样不容易暴露问题，有时候你的问题可能是重复绑定事件。

如点一次按钮就绑一次就导致了绑多次，所以根本原因在这里。你应该要确保事件只被绑一次，而不是确保每次写之前都先off掉。如果你的事件容易出现绑多次的情况说明你的代码组织有问题，这个在开发的时候应该是能够暴露出来的。

（4）对DOM节点较少的不要使用委托 例如说一个表单只有几个input元素，然后你给input加了个委托到form上面，甚至有时候是body上面，由于事件冒泡导致在form上或者在页面上的所有操作都会冒泡到form/body上。

即使操作的不是目标元素，这样jQuery就会收到在body上的事件，然后再判断处理所有的操作的目标元素是不是你指定的那个，如果是再触发你绑的回调函数。

特别是像mousemove触发得频繁的事件都需要执行。所以如果元素比较少或者不需要动态增删的那种就不要使用冒泡了，直接绑在对应的多个元素就好了。

（5）有时候使用原生更简单 例如获取表单的input的和它的value：

```
let email = form.email.value.trim();
```

如果form里面有一个input[name=email]的输入框，就可以这么用。

再如，改变一个button的状态，下面两个其实差不多，但是如果获取不到dom元素的话第一个会挂：

```
$("#update-order")[0].disabled = true;
$("#update-order").prop("disabled", true);
```

设置一个元素的display为block：

```
div.style.display = "block";
```

但是绝大多数的情况下还是要使用jq的API以确保兼容性，如下获取scrollTop：

```
//在Firefox永远返回0
let _scrollTop = document.body.scrollTop();
//正确方法
let scrollTop = $(window).scrollTop();
```

因为在firefox里面需要使用：

```
document.documentElement.scrollTop
```

而这个在Chrome永远返回0。再如window.innerWidth在某些低版本的安卓手机会有问题。

所以当你不确定兼容性的时候，就不要使用原生API，不然你得经过小心验证后再使用。

你可以不用，但不是说不要去了解原生API，多去了解原生DOM操作还是挺有帮助的。

15. 对于常用的属性进行缓存

如下代码，频繁地使用了window.location这个属性：

```
let webLink = window.location.protocol + window.location.hostname;
if(openType === "needtoTouch"){
    webLink += "/admin/lead/list/page" +
             window.location.search.replace(/openType=needToTouch(&?)/, "")
+
             window.location.hash;
}
```

可以先把它缓存一下，加快变量作用域查找：

```
let location = window.location;
let webLink = location.protocol + location.hostname;
if(openType === "needtoTouch"){
    webLink += "/admin/lead/list/page" +
                location.search.replace(/openType=needToTouch(&?)/, "") +

                location.hash;
}
```

当把location变成一个局部变量之后，它的查找时间将明显快于全局变量。你可能会说就算再快这点时间对于用户来说还是没有区别的吧，但是这是做为一名程序员的追求，以及可以让代码更简洁。

16. 尽量不要在JS里面写CSS

如下代码，如果是非选中状态就把颜色置灰：

```
$menuItem.css("color", "#ccc");
```

反之颜色恢复正常：

```
$menuItem.css("color", "#000");
```

这样的代码有问题，如果以后颜色改了，那么你需要改两个地方，一个是CSS里设置，另一个是JS里面设置，而JS写的样式特别容易被忽略，查起来也不好定位。好的做法应该是通过添加删除类的方法：

```
//变成选中态
$menuItem.addClass("selected");
//变成非选中态
$menuItem.removeClass("selected");
```

然后再通过CSS给selected的类添加样式。如果是button之类的控件可以结合:disabled、:checked、:valid等伪类，连类都不用添加

但是有一种是一定要用JS控制的，就是需要先计算然后动态地改变position或者transform的值，如果用CSS3的transition实现不了.

17. 在必要的地方添加非空判断

添加非空判断可以提高代码的稳健性，如下代码：

```
//弹框时显示other monthly charge
showOtherMonthlyCharge: function(otherCharges, $dialog){
    if(!otherCharges || !otherCharges.length){
        return;
    }
}
```

如果传的为空就不用处理，有时候你可能要抛个异常，告诉调用者。对一些比较重要的地方可能还要添加类型检验。

后端传的数据要确保会有那个属性，如果不确定也要添加非空判断。如果调了第三方的API，添加出错处理也很重要，因为你不能确保第三方API一定能正常工作。

在一些你觉得可能会挂的地方做处理，如请求可能会超时，或者返回了undefined的异常结果，这种多使用一般能够发现。

18. 不要用for in循环数组

如下代码：

```
let a = [9, 3, 5];
for(let i in a){
  console.log(a[i])
}
```

正常情况下将会输出数组的元素，但是很不幸的是，如果有人给数组原型添加了一个函数：

```
Array.prototype.add = function(){};
```

循环里的i将会有4个值：0, 1, 2, "add"，这样就导致你的遍历出现问题，所以数组遍历应该使用length属性或者数组的forEach/map方法。

19. 分号规范

JS里面的表达式是可以不用分号结尾，例如Zepto的源码几乎没看到一个分号，但是我们还是提倡要每个句子后面都要加上分号，这样不容易出错。

20. 使用location跳转需要先转义

对于那些根据用户输入内容做跳转，需要先把用户内容做转义，如下有问题的代码：

```
let searchContent = form.search.value.trim();
window.location.href = `/search?key=${searchContent}`;
```

如果用户输入了一个#号如门牌号，将会导致#后面的内容当作锚点了，或者用户可能会输入一个空格。所以如果不确定内容的东西需要先encode一下。

如下代码：

```
let searchContent = encodeURIComponent(form.search.value.trim());
window.location.href = `/search?key=${searchContent}`;
```

这样跳转就没有问题了。

21. 点击跳转尽量不要使用onclick跳转

点击一个容器的时候做跳转，有些人喜欢这么写：

```
<div onclick="window.locatioin.href='/listing/detail?id={{listingId}}'">
    <img>
    <div></div>
</div>
```

其实这样写不好，不利于SEO，如果是一个跳转应该用a标签。

如下：

```
<a href="window.locatioin.href='/listing/detail?id={{listingId}}'">
    <img>
    <div></div>
</a>
```

同时把a标签变成块级。就算你不用做SEO，也应当尽量使用这种方式，因为用这种方式比较自然，还可以控制是否要新开页，如果在移动端也不用考虑click事件是否有延迟的问题。

22. 不要直接使用localStorage

由于Safari的隐身模式下本地存储会被禁用，如果你尝试往localStorage写数据的话，会报超出使用限制的错误：

```
QuotaExceededError (DOM Exception 22): The quota has been exceeded.
```

而Chrome的隐身窗口不会禁用。而使用Safari的用户可能会开隐身窗口，特别是手机上的。这样就导致代码抛异常了，所以为了兼容Safari，不能直接使用localStorage，要做个兼容：

```
Data.hasLocalStorage = true;
try{
```

```
        window.localStorage.trySetData = 1;
    }catch(e){
        Data.hasLocalStorage = false;
    }

    setLocalData: function(key, value){
        if(Data.hasLocalStorage){
            window.localStorage[key] = value;
        }
        else{
            util.setCookie("_LOCAL_DATA_" + key, value, 1000);
        }
    },

    getLocalData: function(key){
        if(Data.hasLocalStorage){
            return window.localStorage[key];
        }
        else{
            return util.getCookie("_LOCAL_DATA_" + key);
        }
    }
```

上面代码做了个兼容，如果不支持localStorage就使用cookie。要注意cookie一个域名最多只能有4kB，50个key，而本地存储限制为5Mb.

23. 使用简便的转换

- 把字符串转整型可以使用+号

```
let maxPrice = +form.maxPrice.value;
```

+号相当于Number：

```
let maxPrice = Number(form.maxPrice.value);
```

parseInt和Number有一个很大的区别是parseInt("10px")结果为10，而Number("10px")是NaN，parseInt会更加自然，其它编程语言也有类似的转换。

但是Number还是能适用很多的场景。

1)把小数去掉尾数转成整型，可以使用 >> 0 如果计算某个数字在第几排：

```
let _row = Math.floor(index / columns);
let row = parseInt(index / columns);
```

都可改成：

```
let row = index / columns >> 0;
```

这个用位运算的效率会明显高于上面两个。

2)转成boolean值用!!

如下代码：

```
let mobile = !!ua.match(/iPhone|iPad|Android|iPod|Windows Phone/)
```

24. 注意返回false的变量

有几个值在if判断里面都返回false：0、false、""、undefined、null、NaN都是false，所以判断一个数组有没有元素可以这么写：

```
if (array.length) {}
```

而不用写成：

```
if (array.length !== 0) {}
```

判断一个字符串是不是空可以写成：

```
if (str) {}
```

但是判断一个变量有没有定义还是要写成：

```
if (typeof foo !== "undefined") {}
```

因为如果直接if变量的话，上面的几个可能取值都将认为是没定义。

25. 使用Object.assgin简化数据赋值

如下代码，在发请求之前，经常需要获取表单的值，然后去修改和添加老数据提交：

```
var orderData = {
    id: 123,
    price: 500
}
orderData.price = 600;
orderData.discount = 15;
orderData.manageFee = 100;
```

其实有一种更优雅的方式那就是使用Object.assign：

```
var setOrderData = {
    price: 600,
    discount: 15,
    manageFee: 100
}

Object.assgin(orderData, setOrderData);
```

使用这个的好处是可以弄一个setOrderData的Object，写成大括号的形式，而不用一个个去赋值，写起来和看起来都比较累。最后再assign一下赋值给原先的Object就可以了。

26. 调试完去掉无关的console

调试完就把console.log之类的打印信息去掉，别想着等一下做完了再删，等一下就忘了。

另外，不要使用alert调试，console/debugger上线了都没事，一般用户也不会开一个控制台，但是alert上线了就完蛋了，特别是有些人喜欢用alert(“fuck”)之类的看下代码有没有运行到这里。

这种调试技巧还是比较初级，要是真上线了可能得卷铺盖走人了。这也可以通过代码检查工具做静态检查。

27. 注意this的指向

如下代码：

```
let searchHandler = {
    search() {
        console.log(this);
        this.ajax();
    },
    ajax() {

    }
};
$searchBtn.on("click", searchHandler.search);
```

当触发searchBtn的点击事件时，search函数里的this已经指向 searchBtn了，因为它是click的回调函数：

```
searchHandler.search.call(btn, event);
```

所以函数运行环境就变成了btn了，因此这种单例的Object最好不要使用this，应直接使用当前命名空间的变量名：

```
let searchHandler = {
    search() {
        console.log(this);
        searchHandler.ajax();
    },
    ajax() {

    }
};
$searchBtn.on("click", searchHandler.search);
```

这样就没问题了。

28. 使用正则表达式做字符串处理

正则表达式可以很方便地处理字符串，通常只要一行代码就搞定了。例如去掉全局的某一个字符，如去掉电话号码的-连接符：

```
phoneNumer = phoneNumber.replace(/\-/g, “”);
```

或者反过来，把电话号码改成3-3-4的形式：

```
phoneNumber = phoneNumber.replace(/^(\d{3})(\d{3})(\d{4})$/, “$1-$2-$3”);
```

熟练掌握正则表达式是每个前端的基本技能。

29. 保持复用模块的观念

当你一个函数要写得很长的时候，例如两、三百行，这个时候你考虑把这个大函数给拆了，拆成几个小函数，然后让主函数的逻辑变得清晰简洁，而每个小函数的功能单一独立，使用者只需要管输入输出，而不需要关心内部是怎么运行的。

如下在地图里面处理用户点击的处理函数：

```
handleMouseClick(latLng, ajax = true) {
    var path = this.path;
    // 这里调了一个closeToFirstPoint的函数判断点击位置是否接近第一个点
    if(path.length >= 2 && ajax && this.closeToFirstPoint(latLng)){
        // 如果是的话调closePath关闭路径
        this.closePath(ajax);
        return;
    }
    path.push({lat: latLng.lat(), lng: latLng.lng()});
    // 调画点的函数
    this.drawPoint(latLng);
    // 调画线的函数
    this.drawSolidLine();
```

```
    // 调画多边形背景的函数
    this.drawPolygonMask();
    this.lastMoveLine && this.lastMoveLine.setMap(null);
    this.$drawTip.hide();
}
```

上面拆成了很多个小函数，如画点的drawPoint函数，使用这个函数只需要关心给它一个当前点的经纬度就可以了，它就帮你画一个点。

在函数之上又可以继续抽象，如把这个画图功能的模块写成一个DrawTool的类，这个类负责整个画图的功能，使用者只需要实例化一个对象，然后调一下init，传一些参数就好了。

先抽成不同的函数，每个函数负责一小块，相似的函数聚集在一起形成一个模块，几个模块的相互调用又形成一个插件。

30. 注意label事件会触发两次

如果label里面有input，监听label的事件会触发两次，如下代码：

```
<form id="choose-fruit">
    <label>
        <input type="radio" name="fruit" value="apple">
        <span>apple</span>
    </label>
    <label>
        <input type="radio" name="fruit" value="pear">
        <span>pear</span>
    </label>
<form>
```

```
<script>
{
    let $form = $("#choose-fruit");
    $form.find("label").on("click", function(event){
        console.log(event.target);
    });
}
</script>
```

当点到span的时候，click事件会触发两次，如果label里面没有input的话，就只会触发一次。这是为什么呢?

因为在label容器内，点到span文字的时候会下发一次click事件给input，input事件又会冒泡到label，因此label会触发两次。

因此如果你直接监听label事件要注意注意触发两次的情况。