# 优化价值

## 1.1 一个例子

如下图1所示，当网络速率200K/S,页面出现布局混乱、显示不全等问题，图2则反映了浏览器发出的图片请求，并未得到响应，超时报错。



图1



依上述所得，当用户并发请求到达一定数量时，且网络中存在一定时延的情况下，portal认证页面是有可能出现上述的情况，因此，页面的优化是必要的。

谷歌浏览器请求限制，自动化压缩、合并，压缩目的128K(包括图片)

## 1.2 优化目的

如下图3所示，web页面呈现的时延与用户心理的感受；当用户等待页面的时间大于1S以上时，用户体验指数开始下降；当时间大于8S，用户对产品的印象分大打折扣，甚至可能投诉产品的使用。



图3

因此，优化不仅只是要解决上述的问题，而且要改善用户体验，提升用户对产品的好评度，提高产品的销量、口碑。

## 1.3 portal存在的优化点

基于上述的认识，总结了以下几点目前superportal前端存在的优化点：

* Html\Js\css压缩
* 图片优化
* 浏览器缓存问题
* 代码冗余、重用率低
* 代码性能优化

另外，除了上述的性能优化点，还有些工程化方面的优化点，但在这里不详述。

# 优化手段

优化手段可从浏览器绘制页面流程、页面内容、渲染等方面展开。

## 2.1 优化绘制页面流程

浏览器绘制页面的整体流程如下：

1. 处理 HTML 标记并构建 DOM 树。
2. 处理 CSS 标记并构建 CSSOM 树。
3. 将 DOM 与 CSSOM 合并成一个渲染树。
4. 根据渲染树来布局，以计算每个节点的几何信息。
5. 将各个节点绘制到屏幕上。

### 2.1.1构建DOM树

如下图4所示，构建DOM树有以下的迭代流程：

1. **转换**： 浏览器从磁盘或网络读取 HTML 的原始字节，并根据文件的指定编码（例如 UTF-8）将它们转换成各个字符。
2. **标签化**： 浏览器将字符串转换成 W3C HTML5 标准规定的各种标签，例如，“<html>”、“<body>”，以及其他尖括号内的字符串。每个标签都具有特殊含义和一组规则。
3. **词法分析**： 发出的标签转换成定义其属性和规则的“对象”。
4. **DOM 构建**： 最后，由于 HTML 标记定义不同标记之间的关系（一些标记包含在其他标记内），创建的对象链接在一个树数据结构内，此结构也会捕获原始标记中定义的父项-子项关系：HTML 对象是 body 对象的父项，body 是 paragraph 对象的父项，依此类推。

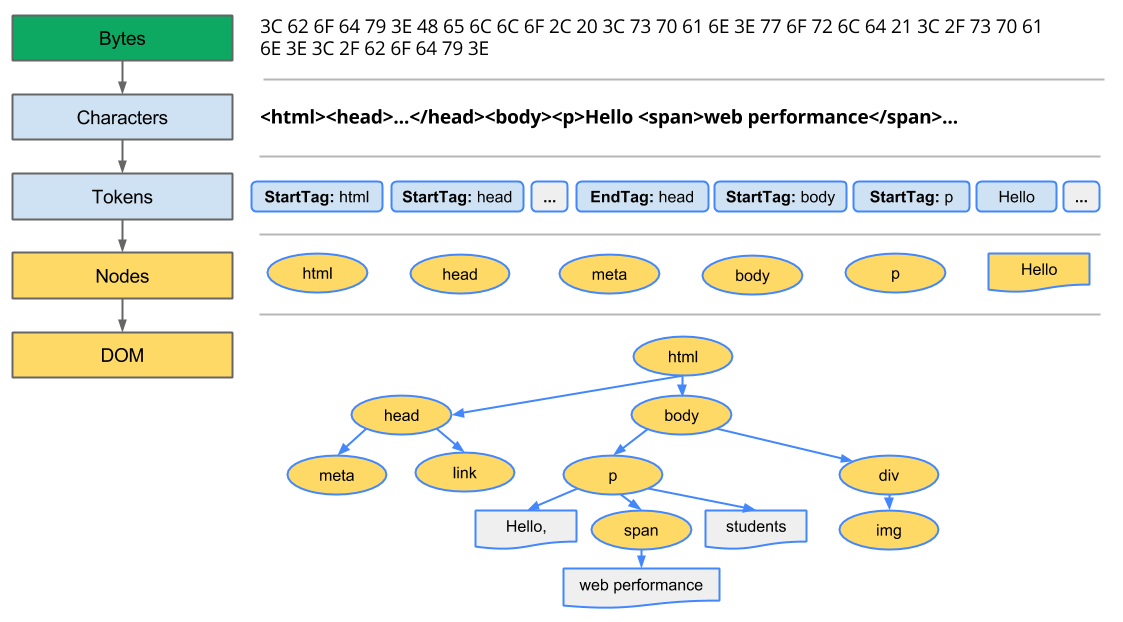


图4

可以使用chrome浏览器提供的调试工具，查看构建DOM树的花费时间。

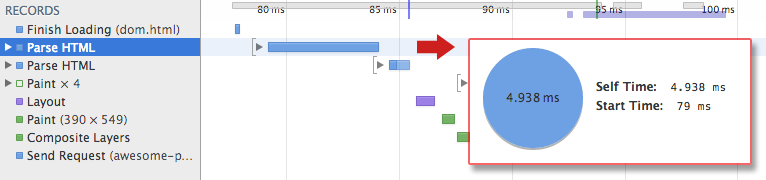


图5

### 2.1.2 构建CSSOM树

当浏览器构建html DOM 时，在文档的 head 部分遇到了一个 引用外部css样式的link 标记时，就会立即发出对该资源的请求。而构建cssom树的流程如html流程，最终得到如下图6所示的cssom树结构。

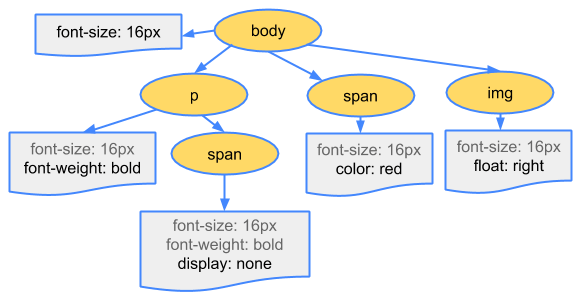


图6

而构建cssom树所花费的时间，仍可以从chrome浏览器提供的调试工具看到。

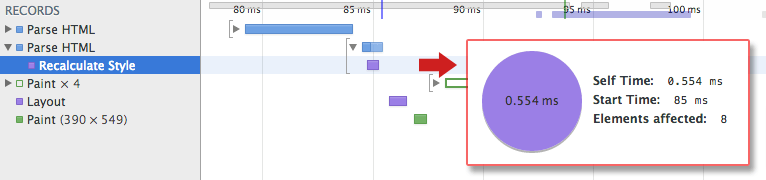


图7

### 2.1.3 构建渲染树、布局、绘制

当浏览器完成了构建dom树、cssom树过程，紧接着是构建渲染树的流程，如下图8所示。该过程主要是通过与cssmo树结合，计算每个dom树节点的样式与内容。

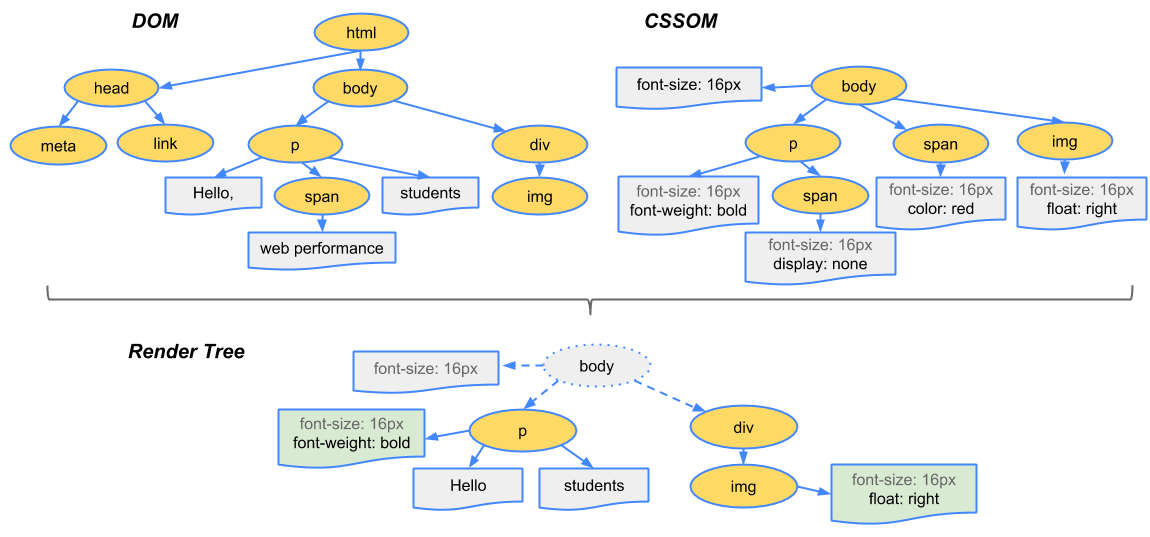


图8

当构建渲染树完成后，进入布局阶段。该阶段主要是计算dom节点的位置与布局，换算成真实屏幕大小的位置像素。而后，将元素实际像素绘制到浏览器屏幕上，呈现出来。

这三个阶段，可从chrome浏览器提供的调试工具看到，如下图9所示。

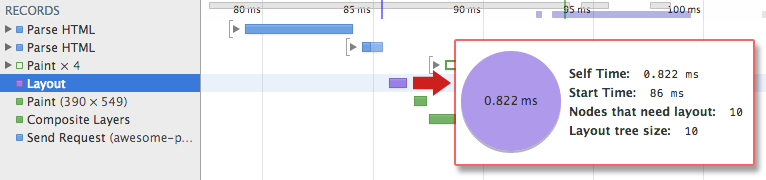


图9

### 2.1.4 回顾及应用

基于上述的绘制流程，可知页面的呈现时间与这5个阶段流程息息相关。可得到以下几点优化规则：

* css文件尽早引用。(防止阻塞后面阶段的执行)
* 评估js、css对dom树节点的操作，若无操作dom节点，则使用异步下载js、css文件。
* 首次渲染的js应避免长时间的运算

这些优化规则应用于superportal系统中，则有以下几点需要考虑优化的：

1. js、css文件的引用，如下图10所示：

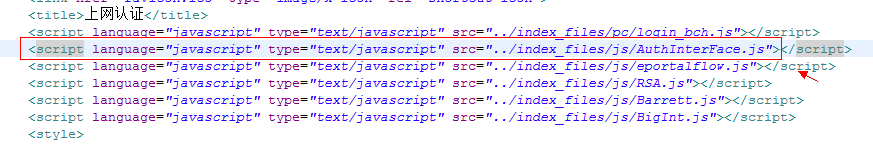
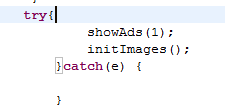


图10

AuthInterFace.js、RSA.js、Barrett.js、BigInt.js等辅助js文件，引用方式可改写为<script src=" ../index\_files/js/AuthInterFace.js " async></script>

1. js代码处理错误、或者运行时间较长，需要考虑优化。



## 2.2 优化页面内容

优化页面内容，主要思想是保证页面功能正常前提下，减少页面大小、页面请求次数；分为以下压缩文件、图片优化、http缓存、请求优化等方面展开。

### 2.2.1 压缩html、js、css

压缩html、js、css文件，主要是减少页面大小，分为源码压缩、gzip传输压缩两种。

**源码压缩，**主要是去除注释、空格、制表符、替换变量名。如下图11所示，代码压缩前片段，图12则是压缩后代码片段；可得出，网页字符数从 406 缩减到 150，缩减率高达 63%，虽然丢失了网页源码可读性。另一例子是JQuery 内容库未压缩开发版本的大小现已接近大约 300KB。而压缩（移除注解等内容）后同一内容库的大小仅为原来的大约 1/3：大约 100KB。

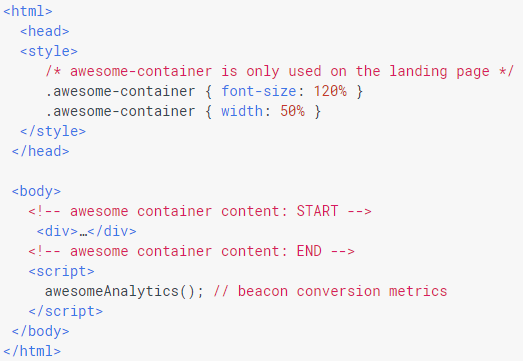


图11

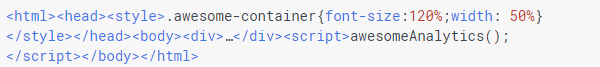
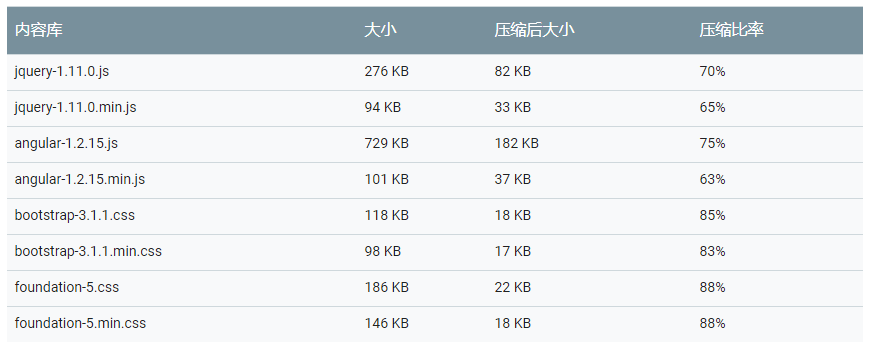


图12

**gzip传输压缩**，采用文本压缩算法，压缩html、js、css文件的字符，一般可实现可实现高达 70-90% 的压缩率。但gzip压缩，首先确保在服务器开启，而浏览器一般都支持解压gzip的http请求。

结合两种压缩方式，可实现下图所示的压缩率。因此，压缩html、js、css文件，大大减少了页面大小，提高了传输效率。



### 2.2.2 图片优化

图片优化，主要是减少图片资源请求次数、图片的大小，分为替换图片、压缩图片、自适应图片等方面。

**替换图片**，指的是用css样式替换图片，例如按钮的渐变、阴影、小图标，之前若是采用图片实现的，可采用css样式替换。

**压缩图片**，WEB页面使用图片，一般分为gif、jpg、png三种主流格式图片；不同格式适用不同场景，gif格式图片用于动图、png格式用于**高分辨率精细的图片**、jpg格式用于可接受有损图片的展示。这几种图片皆有对应的优化工具支持，如下图所示。



**自适应图片**，一般指的是可缩放、自适应页面大小布局的图片；由于图片大小是固定的，但在不同分辨率的屏幕上显示，传统的做法是制作适配各种分辨下的图片，这样无疑增加了图片资源；基于上述的认识，实现自适应图片目前有使用svg格式的矢量图片、css样式设置等方法。

方式1，如下图所示，使用svg格式的矢量图片缩放的效果，光栅图片一般指的是gif、png、jpg格式图片。

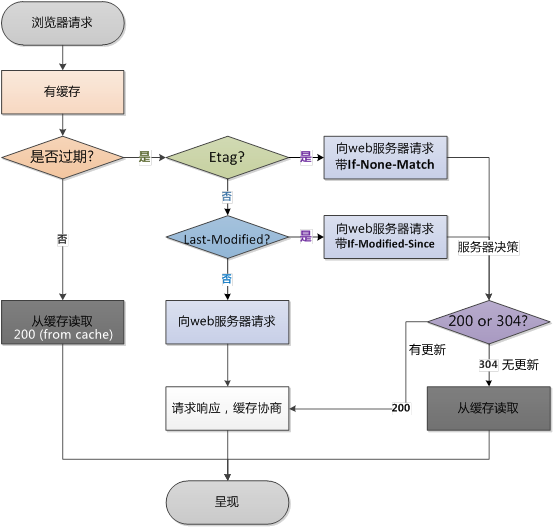


方式2，如下图所示，使用css样式设置改变图片的呈现区域的大小。



### 2.2.3 http缓存

利用浏览器缓存机制，可减少重复内容的请求次数。它分为HTML Meta标签、HTTP头信息两种实现方式，但HTML Meta标签方式，浏览器支持较少，这里主要讲解1.1版本http协议头部缓存控制的方式。如下图所示，浏览器请求某个页面的流程图：



在http协议中Cache-Control头部定义了缓存策略，重点关注“no-cache”、 “no-store”、 “max-age”三种策略。“no-cache”含义是浏览器根据Etag判断是否从缓存上读取页面内容，“no- store”含义是不缓存该页面，“max-age”含义是该页面缓存的时长。

从上述的缓存原理，存在个问题：当服务器突然要更新页面，而浏览器的缓存时长未失效，用户无法请求到最新的资源页面。解决该问题，事实上只需要对每个资源文件加入指纹或标识即可，每个版本中的资源文件指纹均是不同的，当用户请求页面时，浏览器自然会去下载新的资源文件。

### 2.2.4 请求优化

请求优化，主要是重新考量页面中js、css、图片文件的请求次数与内容的大小的关系。一般地，可以合并多个js\css文件减少请求，增加了文件的大小、提高传输效率；但某些页面并不是需要用到被下载的资源文件，因此该方法可能让用户体验到页面首次加载时比较慢。其实，最好的策略是按需加载资源。这可被应用图片加载、大数据量等需要耗时时间长的操作中，例如：当用户点击到某些页面时，才加载该页面的资源；当用户查看下一张图片时，才去请求该图片资源；当用户点击下一列数据时，才去请求下一列的数据。

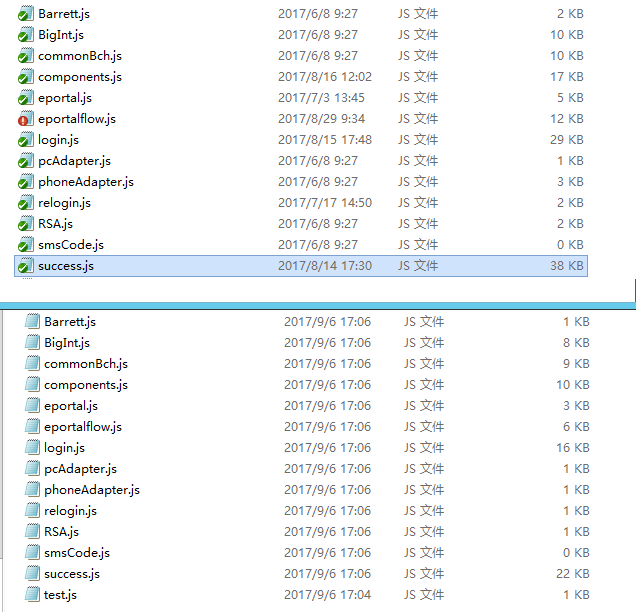
### 2.2.5 回顾及应用

从优化页面内容方面，可总结出以下几点优化项：

1. 压缩html、css、js文件
2. 服务端开启gzip配置
3. 压缩图片
4. 优先选择svg格式图片
5. 考虑使用自适应图片
6. 开启服务端的缓存策略
7. 考虑每个页面的缓存策略
8. 资源文件应携带版本指纹
9. 合并css、js、图片资源文件
10. 按需加载资源、图片文件

…

基于上述的优化项，结合superportal系统的前端框架，只有第10条受于框架限制，不能优化，其余皆可尝试优化，如下图所示。



## 2.3 优化渲染

渲染在页面交互过程中，无处不在。例如：往页面添加ajax请求后的数据展示，鼠标经过超链接的样式变化，用js改变了某个元素的长度等等影响页面重新绘制的操作，如下图所示：



### 2.3.1 渲染过程

渲染过程经过JavaScript、样式计算、布局、绘制、合成过程，如下图所示：



**JavaScript**。一般来说，我们会使用 JavaScript 来实现一些视觉变化的效果。比如用 jQuery 的 animate 函数做一个动画、对一个数据集进行排序或者往页面里添加一些 DOM 元素等。当然，除了 JavaScript，还有其他一些常用方法也可以实现视觉变化效果，比如：CSS Animations、Transitions 和 Web Animation API。

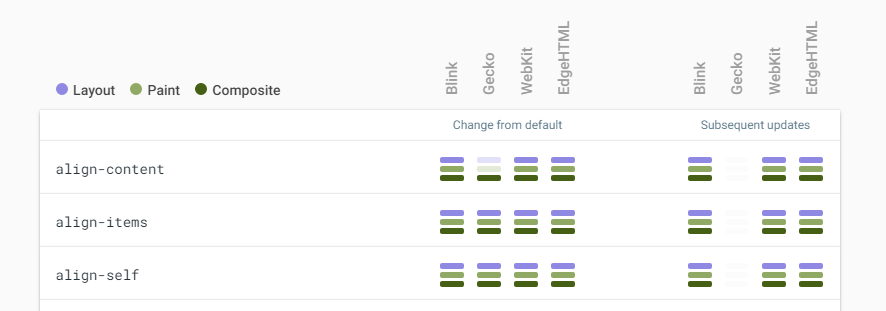
**样式计算**。此过程是根据匹配选择器（例如 .headline 或 .nav > .nav\_\_item）计算出哪些元素应用哪些 CSS 规则的过程。从中知道规则之后，将应用规则并计算每个元素的最终样式。

**布局**。在知道对一个元素应用哪些规则之后，浏览器即可开始计算它要占据的空间大小及其在屏幕的位置。网页的布局模式意味着一个元素可能影响其他元素，例如 <body> 元素的宽度一般会影响其子元素的宽度以及树中各处的节点，因此对于浏览器来说，布局过程是经常发生的。

**绘制**。绘制是填充像素的过程。它涉及绘出文本、颜色、图像、边框和阴影，基本上包括元素的每个可视部分。绘制一般是在多个表面（通常称为层）上完成的。

**合成**。由于页面的各部分可能被绘制到多层，由此它们需要按正确顺序绘制到屏幕上，以便正确渲染页面。对于与另一元素重叠的元素来说，这点特别重要，因为一个错误可能使一个元素错误地出现在另一个元素的上层。

但需要注意的是，并不是所有的操作需要经历上述的5个阶段；如果修改元素的“layout”属性，也就是改变了元素的几何属性（例如宽度、高度、左侧或顶部位置等），那么浏览器将必须检查所有其他元素，然后“自动重排”页面。任何受影响的部分都需要重新绘制，而且最终绘制的元素需进行合成。如果修改“paint only”属性（例如背景图片、文字颜色或阴影等），即不会影响页面布局的属性，则浏览器会跳过布局，但仍将执行绘制。如果更改一个既不要布局也不要绘制的属性，则浏览器将跳到只执行合成。[CSS 触发器](https://csstriggers.com/)描述了修改了dom节点的哪些样式引起了哪个阶段的变化，如下图所示。



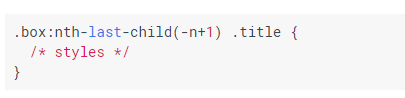
### 2.3.2 渲染过程优化

**JavaScript阶段优化**

如上述渲染过程所述，首当其冲的是JavaScript阶段，而js代码一般运行在主线程上，当js执行复杂或长时间运算时，应当考虑使用worker线程，再将返回结果传递给主线程，再渲染结果。

**样式计算阶段优化**

该阶段是根据给定的元素选择器获取dom节点，再对齐设置样式；因此，对元素的获取应尽量的简单高效，避免诸如下图所示的复杂选择器。.class-xxx-xxx



**布局阶段优化**

该阶段的优化就是尽量不执行该阶段的工作，因为该阶段的执行，一般涉及修改了整个页面的布局。

**绘制阶段优化**

绘制是填充像素的过程，像素最终合成到用户的屏幕上。 它往往是管道中运行时间最长的任务，应尽可能避免此任务。因此，尽量使用 transform 或 opacity 属性，避免其他属性的使用而引发绘制阶段。

**合成阶段优化**

合成是将页面的已绘制部分放在一起以在屏幕上显示的过程。尽量使用 transform 和 opacity 属性更改来实现动画，使用 will-change 或 translateZ 提升移动的元素。

### 2.3.3 回顾及应用

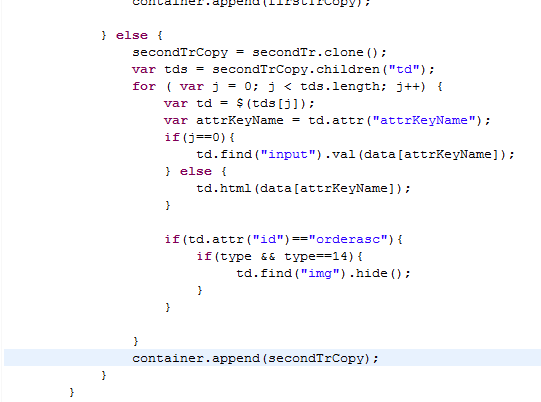
从渲染过程方面，可总结出以下几点优化项：

1. 长时间的js运行，应该放到worker线程中，或者设法优化算法
2. css的选择器，尽量简单，最好是以类来表示
3. 参考[CSS 触发器](https://csstriggers.com/)，尽量避免设置引起重新布局的属性
4. 考虑替换引起重新布局的属性
5. 考虑元素的布局，尽量不引起全页面的布局计算
6. 引入必要的图层，避免重叠绘制或过渡的绘制

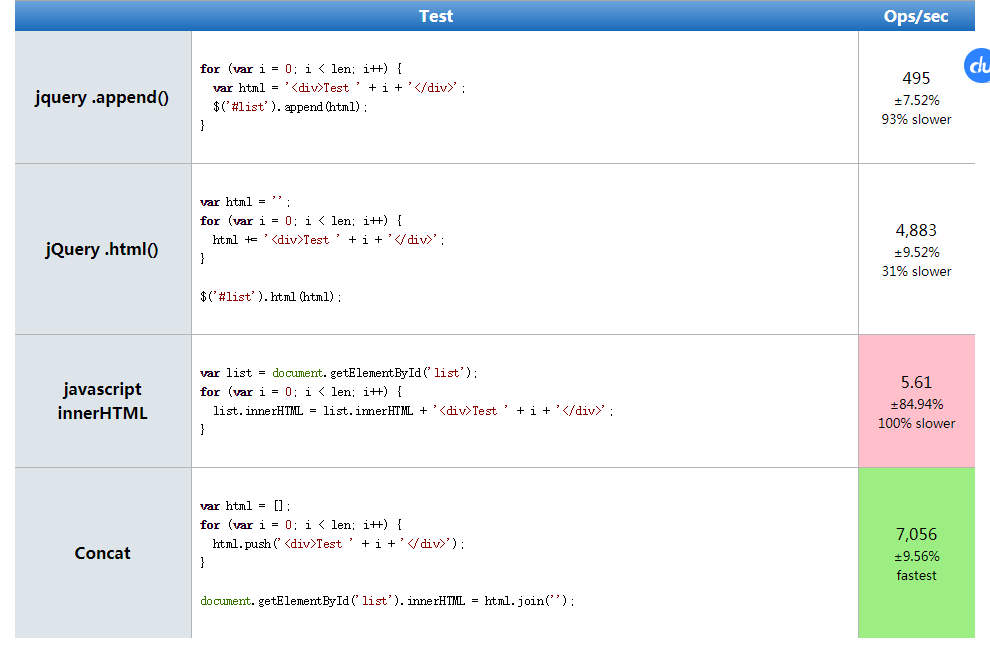
…

基于上述的优化项，结合superportal系统代码，有以下几点待优化：

1. 表格数据的绘制，在eportalAjax文件的214行开始，如下图所示：



经过测试，事实上使用concat效率是最高的，如下图所示。



# 3.参考链接

<https://developers.google.cn/web/fundamentals/performance/?hl=zh-cn>

<http://www.cnblogs.com/520yang/articles/4807408.html>

<https://jsperf.com/jquery-append-vs-html-list-performance/24>