# 浏览器重要线程

1. javaScript引擎线程；---执行浏览器JS处理的线程
2. GUI渲染线程；---进行浏览器渲染工作线程
3. 浏览器事件触发线程；---监听浏览器事件触发线程
4. Http请求线程；----异步请求下载线程
5. 定时线程；---计算定时的触发线程

# 网页加载流程

1. 用户请求URL，浏览器下载html文件；

2. 解析html文件，发现引用外部CSS文件；

3. 浏览器请求外部css文件，下载完进行解析成CSSOM；

4. 载入html中body内容，结合已经解析完CSS文件，进行页面渲染；

5. 在渲染过程中遇到图片文件，继续渲染，同时下载图片资源；

6. 服务器返回图片文件，图片大小对布局会有影响，因此浏览器需要重新渲染这部分代码；渲染的时机点是在渲染空闲时候；

7. 浏览器发现Javascript代码，下载并立即执行；

8. 在JS运行过程中发现有dom节点修改样式如display:none。浏览器会重新渲染这部分代码；

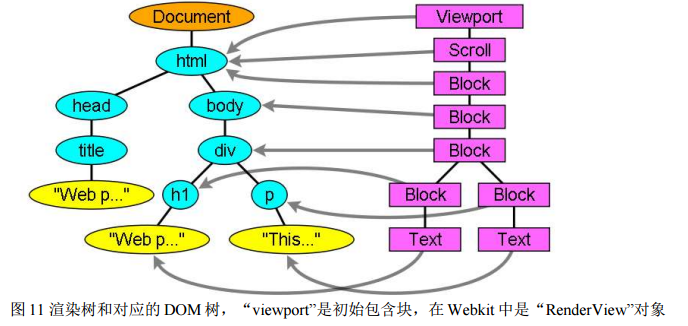
9. 终于等到了＜/html>结束

# Dom tree，Render tree

Dom树：html所有节点的树，包括head，display：none的节点。

Render树：渲染引擎工作需要用到的树，是有可见元素与样式结合组成的一颗树。

两者关系：渲染对象和 DOM 元素是相一致的，但是并不是一对一的关系，非可见元素不会被插入到渲染树 上，有个例子是“head”元素，还有 display 属性为 none 的元素不会出现在树。



# CSS 样式表与渲染关系

渲染树需要有样式表与dom树结合生成。

CSS是并行下载的。下载结束进行加载解析，加载的过程中也是阻塞渲染的。

默认情况下，CSS 被视为阻塞渲染的资源，这意味着在 CSSOM 构建完成前，浏览器会暂停渲染任何已处理的内容。确保精简你的 CSS，尽快传送它，并使用媒体类型与媒体查询来解除阻塞。

如果我们在 CSS 不阻塞的情况下尝试渲染一个普通页面会怎样？

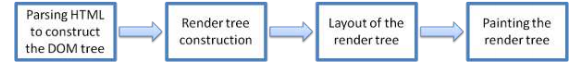


上面的例子，显示了纽约时报有 CSS 与没有 CSS 的情况，这证实了为什么要在 CSS 可用之前阻塞渲染 - 没有 CSS 的页面基本不可用。实际上，右侧的情况通常称为「内容样式短暂失效」(FOUC)。因此，在同时拥有 DOM 和 CSSOM 前，浏览器会阻塞渲染。

项目中为了避免空白页面显示，尽量在head加载必要的css样式。

# 渲染过程

渲染文档是每次8K分块的去解析显示。渲染过程包括



# Javascript引擎与预编译

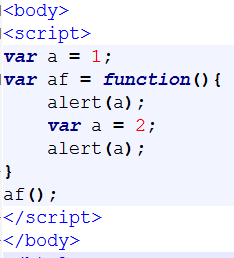
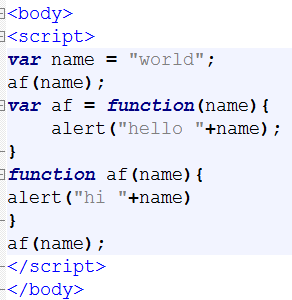
Javascript的运行有两大特性：

1）载入后马上执行，

2）执行时会阻塞页面后续的内容（包括页面的渲染、其它资源的下载）。

参看例子。1.html。2.html。

1. Html 2.html

1. **html结果：undefined 和2**
2. **html结果：hi world 和hello world**

**预编译**

JavaScript预编译听起来高大上，但其实还是比较简单的。

首先，对一段JS代码，JS引擎并不是读一句执行一句，而是读取一段、解释执行一段。而一段一段执行，JS会对读取的这段JS代码整体有个预处理，这个预处理就是所谓的预编译。

最佳实践：变量定义尽量放在函数体或者模块体前面，尽量遵守单一var原则。

# 重绘与重排

重排即reflow，生成布局。

重绘repainter，重新绘制。

需要注意的是，"重绘"不一定需要"重排"，比如改变某个网页元素的颜色，就只会触发"重绘"，不会触发"重排"，因为布局没有改变。但是，"重排"必然导致"重绘"，比如改变一个网页元素的位置，就会同时触发"重排"和"重绘"，因为布局改变了。

在样式的写操作之后，如果有下面这些属性的读操作，都会引发浏览器立即重新渲染。

 offsetTop/offsetLeft/offsetWidth/offsetHeight

 scrollTop/scrollLeft/scrollWidth/scrollHeight

 clientTop/clientLeft/clientWidth/clientHeight

 getComputedStyle()

从性能角度考虑，尽量不要把读操作和写操作，放在一个语句里面。

例如：

// bad

div.style.left = div.offsetLeft + 10 + "px";

div.style.top = div.offsetTop + 10 + "px";

// good

var left = div.offsetLeft;

var top = div.offsetTop;

div.style.left = left + 10 + "px";

div.style.top = top + 10 + "px";

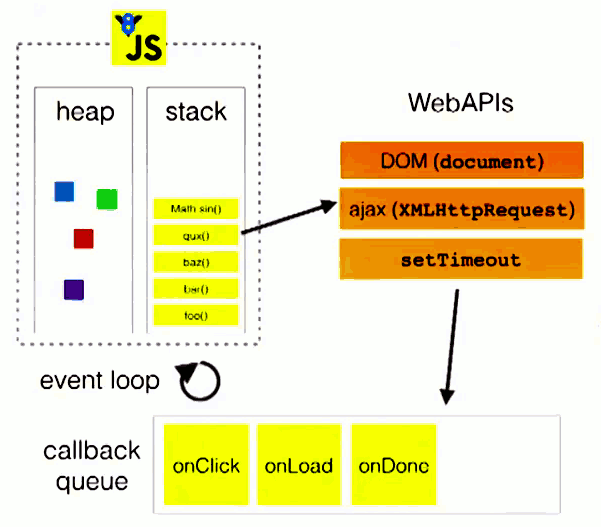
# javascript运行机制---Event Loop

（1）所有同步任务都在主线程上执行，形成一个[执行栈](http://www.ruanyifeng.com/blog/2013/11/stack.html" \t "_blank)（execution context stack）。

（2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。

（3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。

（4）主线程不断重复上面的第三步。



上图中，主线程运行的时候，产生堆（heap）和栈（stack），栈中的代码调用各种外部API，它们在"任务队列"中加入各种事件（click，load，done）。只要栈中的代码执行完毕，主线程就会去读取"任务队列"，依次执行那些事件所对应的回调函数。

**定时器**

在项目中经常发现定时器的任务执行时间有偏后的现象原因是：

setTimeout()只是将事件插入了"任务队列"，必须等到当前代码（执行栈）执行完，主线程才会去执行它指定的回调函数。要是当前代码耗时很长，有可能要等很久，所以并没有办法保证，回调函数一定会在setTimeout()指定的时间执行。

参考链接：

CSS阻塞渲染：

<https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-blocking-css?hl=zh-cn>

how browser worker:

<http://taligarsiel.com/Projects/howbrowserswork1.htm#The_rendering_engine>

JS预编译：

<http://creeperyang.github.io/2014/11/javascript-base-about-scoping-and-precompile/>

渲染白屏出现情况：

<http://www.cnblogs.com/hustskyking/archive/2013/11/04/white-screen-in-chrome.html>

网页性能管理详解

<http://www.ruanyifeng.com/blog/2015/09/web-page-performance-in-depth.html>

浏览器的JavaScript引擎

<http://javascript.ruanyifeng.com/bom/engine.html>

浏览器渲染引擎

<http://www.jianshu.com/p/22b11c917dee>