浏览器加载和渲染html的顺序  
1. IE下载的顺序是从上到下，渲染（就是把请求的内容显示到浏览器屏幕上）的顺序也是从上到下，下载和渲染是同时进行的。   
2. 在渲染到页面的某一部分时，其上面的所有部分都已经下载完成（并不是说所有相关联的元素都已经下载完）。   
3. 如果遇到语义解释性的标签嵌入文件（JS脚本，CSS样式），那么此时IE的下载过程会启用单独连接进行下载。   
4. 样式表在下载完成后，将和以前下载的所有样式表一起进行解析，解析完成后，将对此前所有元素（含以前已经渲染的）重新进行渲染。   
5. JS、CSS中如有重定义，后定义函数将覆盖前定义函数。   
  
JS的加载  
1. 不能并行下载和解析（阻塞下载）。   
2. 当引用了JS的时候，浏览器发送1个js request就会一直等待该request的返回。因为浏览器需要1个稳定的DOM树结构，而JS中很有可能有   
代码直接改变了DOM树结构，比如使用 document.write 或 appendChild,甚至是直接使用的location.href进行跳转，浏览器为了防止出现JS修   
改DOM树，需要重新构建DOM树的情况，所以 就会阻塞其他的下载和呈现.   
如何加快HTML页面加载速度  
1. 页面减肥：   
a. 页面的肥瘦是影响加载速度最重要的因素。   
b. 删除不必要的空格、注释。   
c. 将inline的script和css移到外部文件。   
d. 可以使用HTML Tidy来给HTML减肥，还可以使用一些压缩工具来给JavaScript减肥。   
2. 减少文件数量：   
a. 减少页面上引用的文件数量可以减少HTTP连接数。   
b. 许多JavaScript、CSS文件可以合并最好合并，零点IDC论坛都把自己的JavaScript. functions和Prototype.js合并到一个base.js文件里去了。   
3. 减少域名查询：   
a. DNS查询和解析域名也是消耗时间的，所以要减少对外部JavaScript、CSS、图片等资源的引用，不同域名的使用越少越好。   
4. 缓存重用数据：   
a. 对重复使用的数据进行缓存。   
5. 优化页面元素加载顺序：   
a. 首先加载页面最初显示的内容和与之相关的JavaScript和CSS，然后加载HTML相关的东西，像什么不是最初显示相关的图片、flash、视频等很肥的资源就最后加载。   
6. 减少inline JavaScript的数量：   
a. 浏览器parser会假设inline JavaScript会改变页面结构，所以使用inline JavaScript开销较大。   
b. 不要使用document.write()这种输出内容的方法，使用现代W3C DOM方法来为现代浏览器处理页面内容。   
7. 使用现代CSS和合法的标签：   
a. 使用现代CSS来减少标签和图像，例如使用现代CSS+文字完全可以替代一些只有文字的图片。   
b. 使用合法的标签避免浏览器解析HTML时做“error correction”等操作，还可以被HTML Tidy来给HTML减肥。   
8. Chunk your content：   
a. 不要使用嵌套table，而使用非嵌套table或者div。将基于大块嵌套的table的layout分解成多个小table，这样就不需要等到整个页面（或大table）内容全部加载完才显示。   
9. 指定图像和table的大小：   
a. 如果浏览器可以立即决定图像或table的大小，那么它就可以马上显示页面而不要重新做一些布局安排的工作。   
b. 这不仅加快了页面的显示，也预防了页面完成加载后布局的一些不当的改变。   
c. image使用height和width。   
  
HTML页面加载和解析流程  
1. 用户输入网址（假设是个html页面，并且是第一次访问），浏览器向服务器发出请求，服务器返回html文件。   
2. 浏览器开始载入html代码，发现<head>标签内有一个<link>标签引用外部CSS文件。   
3. 浏览器又发出CSS文件的请求，服务器返回这个CSS文件。   
4. 浏览器继续载入html中<body>部分的代码，并且CSS文件已经拿到手了，可以开始渲染页面了。   
5. 浏览器在代码中发现一个<img>标签引用了一张图片，向服务器发出请求。此时浏览器不会等到图片下载完，而是继续渲染后面的代码。   
6. 服务器返回图片文件，由于图片占用了一定面积，影响了后面段落的排布，因此浏览器需要回过头来重新渲染这部分代码。   
7. 浏览器发现了一个包含一行Javascript代码的<script>标签，赶快运行它。   
8. Javascript脚本执行了这条语句，它命令浏览器隐藏掉代码中的某个<style>（style.display=”none”）。杯具啊，突然就少了这么一个元素，浏览器不得不重新渲染这部分代码。   
9. 终于等到了</html>的到来，浏览器泪流满面……   
10. 等等，还没完，用户点了一下界面中的“换肤”按钮，Javascript让浏览器换了一下＜link＞标签的CSS路径。   
11. 浏览器召集了在座的各位<div><span><ul><li>们，“大伙儿收拾收拾行李，咱得重新来过……”，浏览器向服务器请求了新的CSS文件，重新渲染页面  
静态 HTML页面加载和解析的显示流程