性能优化

1. 图片优化

1.1 样式代替图片

例如：半透明、圆角、阴影、高光、渐变等。这些效果主流的浏览器都能够完美支持，

1.2 精灵图

CSS Sprites，将同类型的图标或按钮等背景图合到一张大图中，减少页面请求。

1.3 字体图标

Icon Font，将图标做成字体文件。优点是图标支持多个尺寸，兼容所有浏览器，减少页面请求等。美中不足的是只支持纯色的icon。SVG，对于绝大多数图案、图标等，矢量图更小，且可缩放而无需生成多套图。现在主流浏览器都支持SVG了，所以可放心使用

1.4 base64

将图片转化为base64编码格式，资源内嵌于CSS或HTML中，不必单独请求。

1.5图片响应式

通常图片加载都是可以通过lazy加载的形式来的，那么可以在加载的时候来判断屏幕的尺寸来达到加载大图还是小图的目的来达到优化。

2. 资源优化

2.1 CSS顶部, JS底部

2.2 CSS JS文件压缩

2.3 尽量使用图片使用精灵图，字体图标

2.4 图片加载可通过懒加载的方式。

总之就是减少资源体积减少资源请求次数。

3. 代码性能

3.1 css代码性能优化

1>使用CSS缩写，减少代码量；

2>减少查询层级：如.header .logo要好过.header .top .logo；

3>减少查询范围：如.header>li要好过.header li；

4>避免TAG标签与CLASS或ID并存：如a.top、button#submit；

5>删除重复的CSS；

3.2 html代码性能优化

1>正确的闭合标签

1

如避免使用<div/>，浏览器会多一个将它解析成<div\></div\>的过程；

2>减少DOM节点：加速页面渲染；

3>最小化重绘和重排

浏览器下载完成所有的html标记，js，css，图片后，它解析文件并创建两个内部结构，一个事dom树，一个是渲染树，浏览器根据DOM树排列，根据渲染树绘制；

什么情况下发生重排：

(1)添加或者删除可见的dom元素

(2)元素位置可变

(3)元素尺寸改变（边距，填充，边框宽高等）

(4)内容改变

(5)最初的页面渲染

(6)浏览器窗口改变尺寸

最好减少这些属性（布局信息）的查询次数，查询时将它赋值给局部变量，参与计算，当你需要对dom树中的多个元素进行修改时，最好根据以下方式来减少重排和重绘：

(1)从文档中国摘下此元素

(2)对其应用多重改变

(3)将元素带回文档中

3.3 js性能优化

1>缓存变量的值，尽量少的使用全局变量

2>如果需要不断运行的代码，不该使用setTimeout，而是使用setInterval，因setTimeout每一次都会初始化一个定时器，而setInterval只会在开始时初始化一个定时器

3>避免使用with语句，由于js的解析器需要检查with块中的变量是否属于with包含的对象，将使with语句执行速度下降，导致js语句很难优化，为了兼顾速度与代码量，找个折中的方案：

var form = document.forms[];

form.name.value = "lihang";

form.address.value = "beijing";

form.age.value = "23";

4>尽量少的去使用闭包，使用后要记得销毁

5>访问速度与成员嵌套深度有关，嵌套约深，访问速度越慢

例如：local.href的访问速度快于window.localtion.href

6>prop()的执行速度要快于attr()

attr()函数主要依赖的是Element对象的getAttribute()和setAttribute()两个方法。prop()函数主要依赖的则是JS中原生的对象属性获取和设置方式。

7>尽量使用for循环遍历数组，少使用for in循环

8>建议将对象进行缓存处理，特别是DOM访问是比较消耗资源的

//c.length没有缓存，每次迭代都要计算一下数组的长度

function foo1() {

var i, b, c=[1,2,3];

for (i=0;i&lt;c.length;i++) {

b = c[i];

if(b === "2"){

return b;

}

}

}

//性能更好，第一次将数组的长度缓存到变量l中，第二次及后续的循环无需计算数组长度

function foo2() {

var i, b, c=[1,2,3],l;

for (i=0,l=c.length;i&lt;l;i++) {

b = c[i];

if(b === "2")

return b;

}

}

//document.getElementById('info')没有缓存，每次都要遍历DOM

function foo1() {

var e;

document.getElementById('info').innerHTML="call 1";

document.getElementById('info').innerHTML="call 2";

}

//性能更好，第二次无需访问DOM

function foo2() {

var e=document.getElementById('info');

e.innerHTML="call 1";

e.innerHTML="call 2";

}

9>尽量不要在函数内部进行过深的嵌套判断

10>避免循环引用，防止内存泄漏

11>建议避免在函数内返回一个未声明的变量，会污染外部变量