前端性能优化

购买服务器视频网址
http://www.html5train.com/kecheng/detail_894292?f=org_coursecenter
=前端开发性能优化方案=
减少HTTP请求次数和请求大小
代码优化
->有利于SEO
->有利于扩展维护
->有利于减少性能消耗
[JS代码优化的108条建议] [雅虎CSS优化的36条建议]
…
DNS及HTTP通信方式的优化

1.在JS中尽量减少闭包的使用（原因：闭包会产生不释放的栈内存）
A:循环给元素做事件绑定的时候，尽可能的把后期需要的信息（例如索引）存储到元素的自定义属性上，而不是创建闭包存储
B:可以在最外层形成一个闭包，把一些后续需要的公共信息进行存储，而不是每一个方法都创建闭包(例如单例模式)
C:尽可能的手动释放不被占用的内存
…

2.尽量合并CSS和JS文件（把需要引入的CSS合并为一个，JS也是合并为一个），原理是在减少HTTP请求次数，尽可能的把合并后的代码进行压缩，减小HTTP请求资源的大小
A:webpack这种自动化构建工具，可以帮我们实现代码的合并和压缩（工程化开发）
B:在移动开发(或者追求高性能的PC端开发[例如百度首页])，如果CSS或者JS不是需要很多，我们可以选择把css和js编程内嵌式（也就是代码直接写在HTML中）

3.尽量使用字体图标或者SVG图标，来代替传统的PNG等格式的图片（因为字体图标等是矢量图(基于代码编写出来的)，放大不会变形，而且渲染速度快，相对比位图要小一些）

4.减少对DOM的操作（主要是减少DOM的重绘和回流(重排)）
A:关于重排的分离读写
B:使用文档碎片或者字符串拼接做数据绑定(DOM的动态创建)

5.在JS中避免“嵌套循环”（这种会额外增加很多循环次数）和“死循环”（一旦遇到死循环浏览器就卡壳了）

6.采用图片的“懒加载”（延迟加载）
目的是为了减少页面“第一次加载”过程中HTTP的请求次数，让页面打开速度变快
步骤：开始加载页面的时候，所有的真实图片都不去发送HTTP请求加载，而是给一张占位的背景图，当页面加载完，并且图片在可视区域内我们再去做图片加载

7.利用浏览器和服务器端的缓存技术(304缓存)，把一些不经常更新的静态资源文件做缓存处

理（例如：JS、CSS、静态图片等都可以做缓存）
原理是为了减少HTTP请求大小，让获取速度更快

8.尽可能使用事件委托(事件代理)来处理事件绑定的操作，减少DOM的频繁操作，其中包括给每一个DOM元素做事件绑定

9.尽量减少CSS表达式的使用(expression)

```
#myDiv {
position: absolute;
width: 100px;
height: 100px;
left: expression(document.body.offsetWidth - 110 + "px");
top: expression(document.body.offsetHeight - 110 + "px");
background: red;
}
```

10.CSS选择器解析规则是从右向左解析

```
.container .link a{
先找到所有的A，再筛选是在.link样式类中的，再次筛选是在.container样式类中的...
先找到的是所有的A，操作起来是消耗性能的，我们在使用CSS选择器的时候尽可能减少对标签选择器的使用
}
```

11.CSS雪碧图技术(css sprite / css 图片精灵)
把所有相对较小资源图片汇总到一张大图上，后期我们只需要把大图加载下来，用背景定位的方式展示对应的小图即可

```
.bg{
background:url('xxx.png');
}
.box1{
background-position:xx xx;
}
.box2{
background-position:xx xx;
}

<div class='bg box1'></div>
```

13.减少对于COOKIE的使用（最主要的是减少本地COOKIE存储内容的大小），因为客户端操作COOKIE的时候，这些信息总是在客户端和服务器端传来传去
async异步：

https://www.cnblogs.com/jiasm/p/7683930.html

14.页面中的数据获取采用异步编程和延迟分批加载
使用异步获取数据，是为了降低HTTP通道的堵塞，不会因为数据没有请求回来耽误下面信息的渲染，提高页面的打开速度（我们可以这样处理：需要动态绑定数据的区域先隐藏，等数据返回并且绑定完成后在让其显示）
延迟分批加载类似于图片懒加载，是为了减少第一次页面加载时候的HTTP请求次数

15.页面中出现音视频标签，我们不让页面加载的时候就去加载这些资源（要不然页面加载速度会变慢）（方案：只需要设置 preload='none' 即可），等待页面加载完成，音视频播放的时候我们在去加载音视频资源

16.在客户端和服务器端进行信息交互的时候，对于多项数据我们尽可能基于JSON格式来进行传送（JSON格式的数据处理方便，资源偏小）==>相对于XML格式的传输才会有这个优势

17.尽可能实现JS的封装（低耦合高内聚），减少页面中的冗余代码（减少HTTP请求资源的大小）

20.CSS中设置定位后，最好使用Z-INDEX改变盒子的层级，让所有的盒子不在相同的平面上，这样后续处理的时候，性能有那么一丢丢的提高

21.在基于AJAX的GET请求进行数据交互的时候，若请求的数据相同，这样下一次从相同地址获取的数据是上一次缓存的数据，但是项目中一般需要获取最新数据，所以要处理下缓存，给url地址添加一个随机数

22.尽量减少对于filter滤镜属性的使用(这个属性消耗性能较大一些)

23.在CSS导入的时候尽量减少使用@import导入式，因为@import是同步操作，只有把这个对应的CSS导入，才会向下加载，而link是异步操作

24.配置ETag(有点类似于304缓存)

25.使用window.requestAnimationFrame（JS中的帧动画）代替传统的定时器动画

26.减少递归的使用，避免死递归，避免由于递归导致的栈内存嵌套（建议使用尾递归）

27.避免使用iframe（不仅不好管控样式，而且相当于在A页面中加载了其它页面，消耗较大）

28.利用H5中提供的localstorage本地存储或者是manifest离线缓存，做一些信息的本地存储，下一次加载页面的时候直接从本地获取，减少HTTP请求次数

29.基于SCRIPT调取JS的时候，可已使用 defer或者async 来异步加载

重量级优化：做CDN加速（烧钱机器）

=额外技巧=
1.我们一般都把CSS放到BODY上，把JS放到BODY下面（原因：让其先加载CSS在加载

JS，先加载CSS是为了保证页面渲染的过程中，元素是带着样式渲染的，而JS一般都是用来操作DOM元素的，需要等到元素加载完再操作）

2.能用CSS搞定的绝对不用JS，能用原生JS搞定的绝对不用插件，绝对不使用FLASH（除了音视频的低版本浏览器播放）
=>CSS处理动画等功能的性能优于JS，而且CSS中的transform变形还开起了硬件加速

3.JS中尽量减少对EVAL的使用，因为JS合并压缩的时候，可能出现由于符号不完善，导致的代码执行优先级错乱问题，EVAL处理起来消耗的性能也是偏大一点的

4.使用keep-alive实现客户端和服务器端的长连接

5.尽量使用设计模式来管理我们的代码（单例、构造、Promise、发布订阅），方便后期的升级和维护

6.开启服务器端的gzip压缩（这个压缩可以有效减少请求资源文件的大小），其实客户端的图片等资源也是可以进行压缩的（但是对于24位的位图，压缩后可能会变模糊）

7.页面中不要出现无效的链接（利于SEO优化），还有其它技巧：提高关键字曝光率、img需要加alt、设置meta标签、标签语义化…

8.避免使用with语句（非常耗性能）

=========