[Javascript中匿名函数的多种调用方式](http://blog.csdn.net/neil_lee_/article/details/16863797)（掌握）

[JavaScript](http://lib.csdn.net/base/javascript)中定义函数的方式有多种，函数直接量就是其中一种。如var fun = function(){},这里function如果不赋值给fun那么它就是一个匿名函数。好，看看匿名函数的如何被调用。

方式1，调用函数，得到返回值。强制运算符使函数调用执行

<**script**>  
 (**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4));  
</**script**>

方式2，调用函数，得到返回值。强制函数直接量执行再返回一个引用，引用再去调用执行

<**script**>  
 (**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 })(3,4);  
</**script**>

这种方式也是很多库爱用的调用方式，如[jQuery](http://jquery.com/)，[Mootools](http://mootools.net/)。

方式3，使用void

<**script**>  
 void  
 **function**(x)  
 {  
 x  
 = x-1;  
 alert(x);  
 }(9);  
</**script**>

方式4，使用-/+运算符

<**script**>  
 -**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4);  
  
 +**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4);  
  
 --**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4);  
  
 ++**function**(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4);  
</**script**>

方式5，使用波浪符(~)

<**script**>  
 ~**function**(x, y) {  
 alert(x+y);  
 **return** x+y;  
 }(3, 4);  
</**script**>

最后看看错误的调用方式

<**script**>  
 *function*(x,y){  
 alert(x+y);  
 **return** x+y;  
 }(3,4);  
</**script**>

JavaScript操作DOM节点元素重排（面试题，掌握）

JavaScript动态向ul中插入100个li节点，5秒后实现元素的倒排

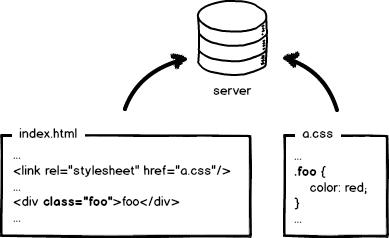
<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**meta charset="utf-8"**>  
 <**script**>  
 *//用JS为id为test的ul下动态创建100个li节点，1秒后将这100个节点实现倒排（节点处理，非innerHTML）* **window**.onload = **function**(){  
 **var** ul = **document**.getElementById(**"test"**);  
 **for**(**var** i=0;i<100;i++){  
 **var** li = **document**.createElement(**"li"**);  
 **var** content = **document**.createTextNode(i);  
 li.appendChild(content);  
 ul.appendChild(li);  
 }  
 *// document.write(ul.firstChild);* setTimeout(**function**(){  
 **var** temp=ul.removeChild(ul.**lastChild**);  
 *//ul.appendChild(temp);* ul.insertBefore(temp,ul.**firstChild**);  
 **for**(**var** j=0;j<99;j++){  
 **var** temp2=ul.removeChild(ul.**lastChild**);  
 ul.insertBefore(temp2,ul.**childNodes**[j+2]);  
 }  
 },5000);  
 }  
 </**script**>  
 </**head**>  
 <**body**>  
 <**ul id="test"**>  
 </**ul**>  
 </**body**>  
</**html**>

BAT大公司里怎样开发和部署web前端代码（了解即可）

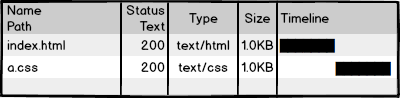
这是一个非常有趣的 非主流前端领域，这个领域要探索的是如何用工程手段解决前端开发和部署优化的综合问题，入行到现在一直在学习和实践中。

在我的印象中，facebook是这个领域的鼻祖，有兴趣、有梯子的同学可以去看看facebook的页面源代码，体会一下什么叫工程化。

接下来，我想从原理展开讲述，多图，较长，希望能有耐心看完。

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/01.png)

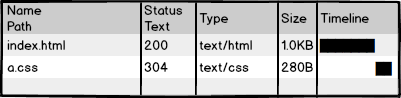
让我们返璞归真，从原始的前端开发讲起。上图是一个“可爱”的index.html页面和它的样式文件a.css，用文本编辑器写代码，无需编译，本地预览，确认OK，丢到服务器，等待用户访问。前端就是这么简单，好好玩啊，门槛好低啊，分分钟学会有木有！

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/02.png)

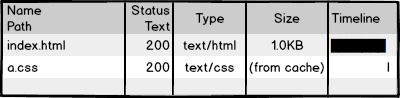
然后我们访问页面，看到效果，再查看一下网络请求，200！不错，太™完美了！那么，研发完成。。。。了么？

等等，这还没完呢！对于大公司来说，那些变态的访问量和性能指标，将会让前端一点也不“好玩”。

看看那个a.css的请求吧，如果每次用户访问页面都要加载，是不是很影响性能，很浪费带宽啊，我们希望最好这样：

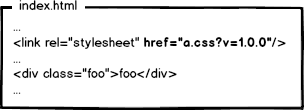
[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/03.png)

利用304，让浏览器使用本地缓存。但，这样也就够了吗？不成！304叫协商缓存，这玩意还是要和服务器通信一次，我们的优化级别是变态级，所以必须彻底灭掉这个请求，变成这样：

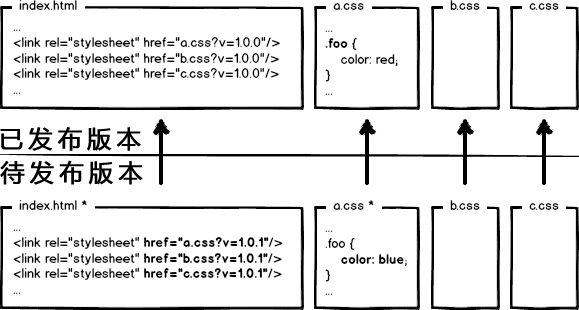
[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/04.png)

强制浏览器使用本地缓存（cache-control/expires），不要和服务器通信。好了，请求方面的优化已经达到变态级别，那问题来了：你都不让浏览器发资源请求了，这缓存咋更新？

很好，相信有人想到了办法：通过更新页面中引用的资源路径，让浏览器主动放弃缓存，加载新资源。好像这样：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/05.png)

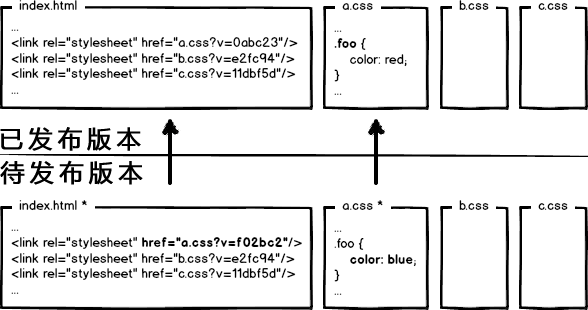
下次上线，把链接地址改成新的版本，就更新资源了不是。OK，问题解决了么？！当然没有！大公司的变态又来了，思考这种情况：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/07.png)

页面引用了3个css，而某次上线只改了其中的a.css，如果所有链接都更新版本，就会导致b.css，c.css的缓存也失效，那岂不是又有浪费了？！

重新开启变态模式，我们不难发现，要解决这种问题，必须让url的修改与文件内容关联，也就是说，只有文件内容变化，才会导致相应url的变更，从而实现文件级别的精确缓存控制。

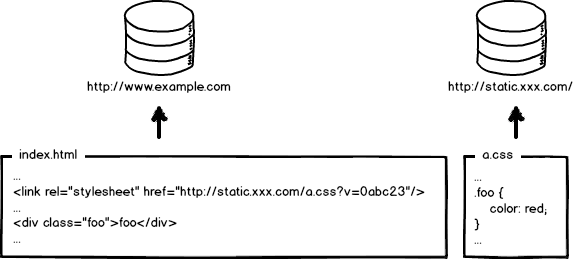
什么东西与文件内容相关呢？我们会很自然的联想到利用 [数据摘要要算法](http://baike.baidu.com/view/10961371.htm) 对文件求摘要信息，摘要信息与文件内容一一对应，就有了一种可以精确到单个文件粒度的缓存控制依据了。好了，我们把url改成带摘要信息的：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/08.png)

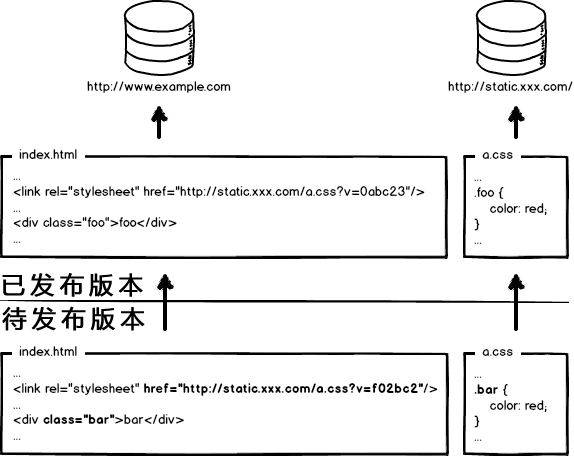
这回再有文件修改，就只更新那个文件对应的url了，想到这里貌似很完美了。你觉得这就够了么？大公司告诉你：图样图森破！

唉~~~~，让我喘口气

现代互联网企业，为了进一步提升网站性能，会把静态资源和动态网页分集群部署，静态资源会被部署到CDN节点上，网页中引用的资源也会变成对应的部署路径：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/09.png)

好了，当我要更新静态资源的时候，同时也会更新html中的引用吧，就好像这样：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/10.png)

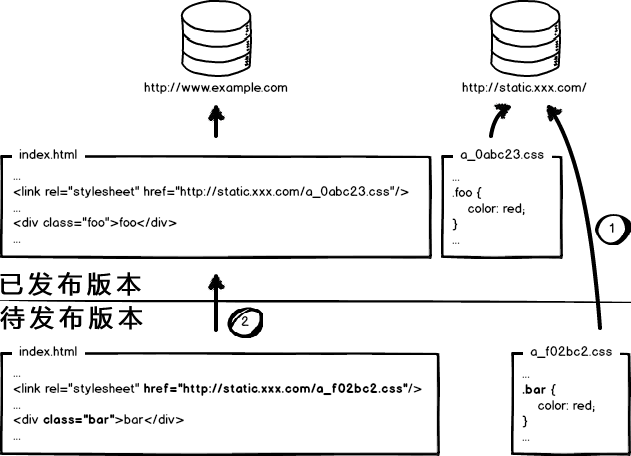
这次发布，同时改了页面结构和样式，也更新了静态资源对应的url地址，现在要发布代码上线，亲爱的前端研发同学，你来告诉我，咱们是先上线页面，还是先上线静态资源？

先部署页面，再部署资源：在二者部署的时间间隔内，如果有用户访问页面，就会在新的页面结构中加载旧的资源，并且把这个旧版本的资源当做新版本缓存起来，其结果就是：用户访问到了一个样式错乱的页面，除非手动刷新，否则在资源缓存过期之前，页面会一直执行错误。

先部署资源，再部署页面：在部署时间间隔之内，有旧版本资源本地缓存的用户访问网站，由于请求的页面是旧版本的，资源引用没有改变，浏览器将直接使用本地缓存，这种情况下页面展现正常；但没有本地缓存或者缓存过期的用户访问网站，就会出现旧版本页面加载新版本资源的情况，导致页面执行错误，但当页面完成部署，这部分用户再次访问页面又会恢复正常了。 好的，上面一坨分析想说的就是：先部署谁都不成！都会导致部署过程中发生页面错乱的问题。所以，访问量不大的项目，可以让研发同学苦逼一把，等到半夜偷偷上线，先上静态资源，再部署页面，看起来问题少一些。

但是，大公司超变态，没有这样的“绝对低峰期”，只有“相对低峰期”。So，为了稳定的服务，还得继续追求极致啊！

这个奇葩问题，起源于资源的 覆盖式发布，用 待发布资源 覆盖 已发布资源，就有这种问题。解决它也好办，就是实现 非覆盖式发布。

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/11.png)

看上图，用文件的摘要信息来对资源文件进行重命名，把摘要信息放到资源文件发布路径中，这样，内容有修改的资源就变成了一个新的文件发布到线上，不会覆盖已有的资源文件。上线过程中，先全量部署静态资源，再灰度部署页面，整个问题就比较完美的解决了。

所以，大公司的静态资源优化方案，基本上要实现这么几个东西：

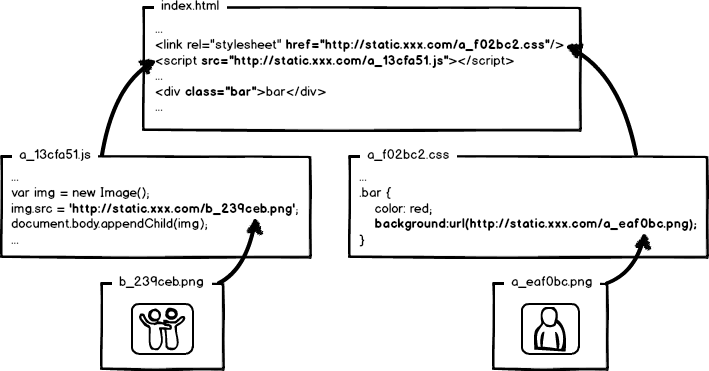
配置超长时间的本地缓存 —— 节省带宽，提高性能

采用内容摘要作为缓存更新依据 —— 精确的缓存控制

静态资源CDN部署 —— 优化网络请求

更资源发布路径实现非覆盖式发布 —— 平滑升级

全套做下来，就是相对比较完整的静态资源缓存控制方案了，而且，还要注意的是，静态资源的缓存控制要求在 前端所有静态资源加载的位置都要做这样的处理 。是的，所有！什么js、css自不必说，还要包括js、css文件中引用的资源路径，由于涉及到摘要信息，引用资源的摘要信息也会引起引用文件本身的内容改变，从而形成级联的摘要变化，大概示意图就是：

[](https://raw.githubusercontent.com/fouber/blog/master/assets/resource/14.png)

好了，目前我们快速的学习了一下前端工程中关于静态资源缓存要面临的优化和部署问题，新的问题又来了：这™让工程师怎么写码啊！！！

要解释优化与工程的结合处理思路，又会扯出一堆有关模块化开发、资源加载、请求合并、前端框架等等的工程问题，以上只是开了个头，解决方案才是精髓，但要说的太多太多，有空再慢慢展开吧。

总之，前端性能优化绝逼是一个工程问题！

以上不是我YY的，可以观察 百度 或者 facebook 的页面以及静态资源源代码，查看它们的资源引用路径处理，以及网络请中静态资源的缓存控制部分。再次赞叹facebook的前端工程建设水平，跪舔了。

建议前端工程师多多关注前端工程领域，也许有人会觉得自己的产品很小，不用这么变态，但很有可能说不定某天你就需要做出这样的改变了。而且，如果我们能把事情做得更极致，为什么不去做呢？

另外，也不要觉得这些是运维或者后端工程师要解决的问题。如果由其他角色来解决，大家总是把自己不关心的问题丢给别人，那么前端工程师的开发过程将受到极大的限制，这种情况甚至在某些大公司都不少见！

一个JS图片放大镜，实现动态记录的图片放大，避免内存泄露，解决IE6下无法遮盖select控件问题（自主选择，建议掌握，太难就做了解）

<**script**>  
 */\*\*  
 qsoft.PopBigImage  
  
 version：0.35  
 author： kimmking@163.com  
 date： 2009年3月25日3:54:50  
  
 version：1.1  
 author： dingherry@163.com  
 date： 2011-8-22 20:05:42  
  
 功能描述：  
 一般页面的的图片为了布局考虑，显示大小都小于实际大小。  
 鼠标在图片上移动时，在旁边展示一个跟此图片显示大小一样的层。  
 并讲鼠标附近区域对应的原始图片的区域按原始大小显示在这个层中。  
  
 v0.1：实现了IE下的鼠标滑动小图动态展示对应的放大局部图的功能。  
 v0.2：实现了Firefox的兼容支持，修改了IE下传递偏移0,0时有空隙的对不齐问题。  
 v0.3：实现了鼠标第一次进入图片时动态创建显示层。提供了一个静态创建方法。  
 v0.35：加入了对google chrome浏览器的支持。  
 v1.0：实现了IE下针对动态绑定图片的控件，避免IE内存泄露。  
 v1.1：修正了IE6下select控件无法遮盖的问题。  
  
 参数描述：  
 origImageId： 要绑定的img对象的id  
 dx：展示大图相对于绑定的img对象右方的x轴偏移量  
 dy：展示大图相对于绑定的img对象上方的y轴偏移量  
 mx：展示层的宽  
 mx在0到1之间时，取大图的宽\*mx的值与小图的宽中的较大者  
 mx在1到10之间时，取小图的宽\*mx的值与大图的宽中的较小者  
 mx大于10时，确保mx在大小图的宽之间，超出的话，取边界值  
 my：展示层的高  
 参照mx的值  
 bflag：create方法中渲染完后是否将展示层显示出来，  
 在onmouseover事件中使用true参数  
 在页面加载时初始化的话使用false参数  
  
 用法：  
 1、页面加载后统一预先加载，在页面上添加JavaScript脚本  
 window.onload = function(){  
 new qsoft.PopBigImage("orig",20,0,2,2).render();  
 //或是 qsoft.PopBigImage.create("orig",20,0,2,2,false).render();  
 }  
  
 或是  
 2、鼠标第一次进入图片时才加载本图片的显示层，在img标签中添加  
 onmouseover="qsoft.PopBigImage.create(this,20,0,2,2,true);"  
  
 \*\*/  
 /\*\*  
 \* 放大镜开关  
 \*/* **var *imgIsSelected*** = **false**;  
  
 */\*\*  
 \* 每次点击一个记录，都将图像保存给一个固定的对象(picMaxObject) 目的：防止IE内存泄露  
 \*/* **var *picMaxObject***;  
  
 */\*\*  
 \* 图像对象，作用类似于picMaxObject  
 \*/* **var *image*** = **new Image**();  
  
 */\*\*  
 \* LOGO  
 \*/* **var *qsoft*** = {  
 **version** : 1.0,  
 **isIE** : **document**.**all** ? **true** : **false**,  
 **prefx** : **'qsoft'**,  
 **\_\_id** : 0,  
 nextId : **function**() {  
 **return this**.**prefx** + **this**.**\_\_id**++;  
 }  
 }  
  
 */\*\*  
 \* 图像放大镜算法主程序  
 \*  
 \** ***@param*** *origImage  
 \* 图像标签<img id="" />的id  
 \** ***@param*** *dx  
 \* 放大镜左上角水平坐标相对于<img id="" />标签右上角水平坐标的偏移量，正数在<img id="" />右边，负数在左边  
 \** ***@param*** *dy  
 \* 放大镜左上角垂直坐标相对于<img id="" />标签右上角垂直坐标的偏移量，正数在<img id="" />下边，负数在上边  
 \** ***@param*** *mx  
 \* 放大倍率 默认设置1即可  
 \** ***@param*** *my  
 \* 放大倍率 默认设置1即可  
 \*/* ***qsoft***.PopBigImage = **function**(origImage, dx, dy, mx, my) {  
  
 **var** type = typeof (origImage);  
  
 **if** (type.toLowerCase() == **"string"**)  
 **this**.**oim** = **document**.getElementById(origImage);  
 **else  
 this**.**oim** = origImage;  
  
 **if** (typeof (**this**.**oim**.pbi) != **"undefined"**)  
 **return this**.**oim**.pbi;  
  
 **this**.**id** = ***qsoft***.nextId();  
 **this**.**oim**.**\_\_maskid** = **this**.**id**;  
 **this**.**oim**.**style**.**cursor** = **"crosshair"**;  
  
 **this**.**ow** = **this**.**oim**.**width**;  
 **this**.**oh** = **this**.**oim**.**height**;  
  
 **this**.**detaX** = (typeof (dx) == **"undefined"**) ? 30 : dx;  
 **this**.**detaY** = (typeof (dy) == **"undefined"**) ? 0 : dy;  
  
 **var** *getPos* = **function**(o) *// for chrome* {  
 **var** x = 0, y = 0;  
 **do** {  
 x += o.**offsetLeft**;  
 y += o.**offsetTop**;  
 } **while** ((o = o.offsetParent));  
 **return** {  
 **left** : x,  
 **top** : y  
 };  
 }  
  
 **this**.getPosition = **function**(o) {  
 **return document**.**documentElement**.getBoundingClientRect  
 && o.getBoundingClientRect() || *getPos*(o);  
 }  
  
 **var** rect = **this**.getPosition(**this**.**oim**);  
 **this**.**ol** = rect.**left** + **this**.**detaX** + **this**.**ow** - (***qsoft***.**isIE** ? 2 : 0);  
 **this**.**ot** = rect.**top** + **this**.**detaY** - (***qsoft***.**isIE** ? 2 : 0);  
  
 **this**.**src** = **this**.**oim**.**src**;  
  
 **this**.getImageSize = **function**(img) {  
 **var** size = {};  
 ***image***.**src** = img.**src**;  
  
 size.**width** = ***image***.**width**;  
 size.**height** = ***image***.**height**;  
 size.**filesize** = ***image***.fileSize;  
  
 **return** size;  
 }  
  
 **var** rsize = **this**.getImageSize(**this**.**oim**);  
  
 **this**.**w** = rsize.**width**;  
 **this**.**h** = rsize.**height**;  
  
 **this**.**maskX** = (typeof (mx) == **"undefined"**) ? **this**.**ow** : mx;  
 **this**.**maskY** = (typeof (my) == **"undefined"**) ? **this**.**oh** : my;  
 **if** (**this**.**maskX** < 1)  
 **this**.**maskX** = **Math**.ceil(**this**.**w** \* **this**.**maskX**);  
 **else if** (**this**.**maskX** < 10)  
 **this**.**maskX** = **Math**.ceil(**this**.**ow** \* **this**.**maskX**);  
 **if** (**this**.**maskY** < 1)  
 **this**.**maskY** = **Math**.ceil(**this**.**h** \* **this**.**maskY**);  
 **else if** (**this**.**maskY** < 10)  
 **this**.**maskY** = **Math**.ceil(**this**.**oh** \* **this**.**maskY**);  
 **this**.**maskX** = **this**.**maskX** < **this**.**ow** ? **this**.**ow** : **this**.**maskX**;  
 **this**.**maskY** = **this**.**maskY** < **this**.**oh** ? **this**.**oh** : **this**.**maskY**;  
 **this**.**maskX** = **this**.**maskX** > **this**.**w** ? **this**.**w** : **this**.**maskX**;  
 **this**.**maskY** = **this**.**maskY** > **this**.**h** ? **this**.**h** : **this**.**maskY**;  
  
 **var** qObj = **this**;  
 **this**.createMask = **function**() {  
 **if** (typeof (**this**.**mask**) == **"undefined"**) {  
 **this**.**mask** = **document**.createElement(**"div"**);  
 **this**.**mask**.**id** = **this**.**oim**.**\_\_maskid** + **"\_mask"**;  
 **this**.**mask**.**style**.**position** = **"absolute"**;  
 **this**.**mask**.**style**.**width** = **this**.**maskX** + **"px"**;  
 **this**.**mask**.**style**.**height** = **this**.**maskY** + **"px"**;  
 **this**.**mask**.**style**.**left** = **this**.**ol** + **"px"**;  
 **this**.**mask**.**style**.**top** = **this**.**ot** + **"px"**;  
 **this**.**mask**.**style**.**backgroundImage** = **"url("** + **this**.**src** + **")"**;  
 **this**.**mask**.**style**.**backgroundRepeat** = **"no-repeat"**;  
 **this**.**mask**.**style**.**display** = **"none"**;  
 **this**.**mask**.**style**.**zIndex** = 1000;  
 *// IE6下遮盖其他控件* **this**.**mask**.**innerHTML** = **'<iframe style=\'width:100%;height:100%;position:absolute;z-index:-1;filter:alpha(opacity=0);border:none\' frameborder=\'no\' border=\'0\' ></iframe>'**;  
 **document**.**body**.appendChild(**this**.**mask**);  
 }  
 }  
  
 **this**.regEvent = **function**() {  
 **this**.**oim**.**onmousemove** = **function**() {  
 **var** e = arguments[0] || **window**.**event** || **top**.**event**;  
 ;  
 **var** ct = e.**target** || e.**srcElement**;  
 **var** sz = qObj.getPosition(ct);  
 **var** ox = ***qsoft***.**isIE** ? e.**offsetX** : (e.**pageX** - sz.**left**);  
 **var** oy = ***qsoft***.**isIE** ? e.**offsetY** : (e.**pageY** - sz.**top**);  
 **var** x = **Math**.ceil(ox \* qObj.**w** / qObj.**ow**) - qObj.**maskX** / 2;  
 **var** y = **Math**.ceil(oy \* qObj.**h** / qObj.**oh**) - qObj.**maskY** / 2;  
  
 **if** (x < 0)  
 x = 0;  
 **if** (y < 0)  
 y = 0;  
 **var** maxx = **Math**.ceil((qObj.**w** - qObj.**maskX**));  
 **var** maxy = **Math**.ceil((qObj.**h** - qObj.**maskY**));  
 **if** (x > maxx)  
 x = maxx;  
 **if** (y > maxy)  
 y = maxy;  
 qObj.**mask**.**style**.**backgroundPosition** = -x + **"px "** + -y + **"px"**;  
 sz = **null**;  
 delete dz;  
 }  
  
 **this**.**oim**.**onmouseout** = **function**() {  
 qObj.**mask**.**style**.**display** = **"none"**;  
 }  
  
 **this**.**oim**.**onmouseover** = **function**() {  
 **if** (!***imgIsSelected***) {  
 qObj.**mask**.**style**.**display** = **"none"**;  
 } **else** {  
 qObj.**mask**.**style**.**display** = **"block"**;  
 }  
 }  
  
 }  
  
 **this**.render = **function**() {  
 **this**.createMask();  
 **this**.regEvent();  
 }  
  
 **this**.resize = **function**(origImage) {  
 **this**.**oim** = **document**.getElementById(origImage);  
  
 **if** (**this**.**src** == **this**.**oim**.**src**) {  
 **return**;  
 }  
  
 **if** (**this**.**oim**.**readyState** == **"uninitialized"**) {  
 **while** (**this**.**oim**.**readyState** == **"complete"**) {  
 **break**;  
 }  
 }  
 **if** (typeof (**this**.**oim**.pbi) != **"undefined"**)  
 **return this**.**oim**.pbi;  
  
 **this**.**oim**.**style**.**cursor** = **"crosshair"**;  
  
 **this**.**ow** = **this**.**oim**.**width**;  
 **this**.**oh** = **this**.**oim**.**height**;  
  
 **this**.**detaX** = (typeof (dx) == **"undefined"**) ? 30 : dx;  
 **this**.**detaY** = (typeof (dy) == **"undefined"**) ? 0 : dy;  
  
 **var** rect = **this**.getPosition(**this**.**oim**);  
 **this**.**ol** = rect.**left** + **this**.**detaX** + **this**.**ow** - (***qsoft***.**isIE** ? 2 : 0);  
 **this**.**ot** = rect.**top** + **this**.**detaY** - (***qsoft***.**isIE** ? 2 : 0);  
  
 **this**.**src** = **this**.**oim**.**src**;  
 **var** rsize = **this**.getImageSize(**this**.**oim**);  
  
 **this**.**w** = rsize.**width**;  
 **this**.**h** = rsize.**height**;  
  
 **this**.**maskX** = (typeof (mx) == **"undefined"**) ? **this**.**ow** : mx;  
 **this**.**maskY** = (typeof (my) == **"undefined"**) ? **this**.**oh** : my;  
 **if** (**this**.**maskX** < 1)  
 **this**.**maskX** = **Math**.ceil(**this**.**w** \* **this**.**maskX**);  
 **else if** (**this**.**maskX** < 10)  
 **this**.**maskX** = **Math**.ceil(**this**.**ow** \* **this**.**maskX**);  
 **if** (**this**.**maskY** < 1)  
 **this**.**maskY** = **Math**.ceil(**this**.**h** \* **this**.**maskY**);  
 **else if** (**this**.**maskY** < 10)  
 **this**.**maskY** = **Math**.ceil(**this**.**oh** \* **this**.**maskY**);  
 **this**.**maskX** = **this**.**maskX** < **this**.**ow** ? **this**.**ow** : **this**.**maskX**;  
 **this**.**maskY** = **this**.**maskY** < **this**.**oh** ? **this**.**oh** : **this**.**maskY**;  
 **this**.**maskX** = **this**.**maskX** > **this**.**w** ? **this**.**w** : **this**.**maskX**;  
 **this**.**maskY** = **this**.**maskY** > **this**.**h** ? **this**.**h** : **this**.**maskY**;  
  
 **this**.**mask**.**style**.**width** = **this**.**maskX** + **"px"**;  
 **this**.**mask**.**style**.**height** = **this**.**maskY** + **"px"**;  
 **this**.**mask**.**style**.**left** = **this**.**ol** + **"px"**;  
 **this**.**mask**.**style**.**top** = **this**.**ot** + **"px"**;  
 **this**.**mask**.**style**.**backgroundImage** = **"url("** + **this**.**src** + **")"**;  
 **this**.**mask**.**style**.**backgroundRepeat** = **"no-repeat"**;  
 rect = **null**;  
 delete rect;  
 }  
  
 }  
  
 */\*\*  
 \* 图片放大镜主程序  
 \*  
 \** ***@param*** *origImage  
 \* 图像标签<img id="" />的id  
 \** ***@param*** *dx  
 \* 放大镜左上角水平坐标相对于<img id="" />标签右上角水平坐标的偏移量，正数在<img id="" />右边，负数在左边  
 \** ***@param*** *dy  
 \* 放大镜左上角垂直坐标相对于<img id="" />标签右上角垂直坐标的偏移量，正数在<img id="" />下边，负数在上边  
 \** ***@param*** *mx  
 \* 放大倍率 默认设置1即可  
 \** ***@param*** *my  
 \* 放大倍率 默认设置1即可  
 \** ***@param*** *bflag  
 \* 是否打开  
 \** ***@return*** *qsoft.PopBigImage  
 \*/* ***qsoft***.PopBigImage.create = **function**(origImage, dx, dy, mx, my, bflag) {  
 **var** q = **new *qsoft***.PopBigImage(origImage, dx, dy, mx, my);  
 q.render();  
 **if** (bflag)  
 q.**mask**.**style**.**display** = **"block"**;  
  
 ***picMaxObject*** = q;  
 **return** q;  
 }  
</**script**>

解决blur与click冲突(面试题，掌握)

在开发中我们会经常遇到blur和click冲突的情况。下面叙述了开发中常遇到的“下拉框”的问题，并提供了两种解决方案。

一、blur和click事件简述

blur事件：当元素失去焦点时触发blur事件；其为表单事件，blur和focus事件不会冒泡，其他表单事件都可以。

click事件：当点击元素时触发click事件；所有元素都有此事件，会产生冒泡。

示例1：blur事件为表单事件

<**input type="text" id="tel"**>  
<**script**>  
 **document**.addEventListener(**"blur"**, **function**(){  
 **console**.log(**"my document blur"**);  
 });  
  
 **var *ipt*** = **document**.getElementById(**"tel"**);  
 ***ipt***.addEventListener(**"blur"**, **function**(){  
 **console**.log(**"my input blur"**);  
 });  
</**script**>  
// 输出结果：document为非表单元素

示例2：click事件可冒泡

<**input type="text" id="tel"**>  
<**script**>  
 document.addEventListener(**"click"**, **function**(){  
 console.log(**"my document click"**);  
 });  
  
 **var** ipt = document.getElementById(**"tel"**);  
 ipt.addEventListener(**"click"**, **function**(){  
 console.log(**"my input click"**);  
 });  
</**script**>  
// 输出结果：  
my input click  
my document click

示例3：点击某元素导致前一个元素失去焦点，blur事件优先于click事件

<**input type="text" id="ipt"**>  
<**input type="button" id="btn" value="点我"**>  
<**script**>  
 **var *ipt*** = **document**.getElementById(**"ipt"**);  
 ***ipt***.addEventListener(**"blur"**, **function**(){  
 **console**.log(**"my input blur"**);  
 });  
  
 **var *btn*** = **document**.getElementById(**"btn"**);  
 ***btn***.addEventListener(**"click"**, **function**(){  
 **console**.log(**"my button click"**);  
 });  
</**script**>  
// 输出结果：  
my input blur  
my button click

二、下拉框blur和click事件冲突，导致不能正常选择值

实际开发中，我们会经常遇到某一下拉列表框，点击其他元素消失列表框；点击下拉框子元素使其生效的需求。这就会面临冲突问题。

<!DOCTYPE **html**>  
<**html**>  
 <**head**>  
 <**meta charset="utf-8"**>  
 </**head**>  
 <**body**>  
 *<!-- DOM结构示意 -->* <**input type="text" placeholder="请选择姓氏" readonly**>  
 <**div class="search-list" data-status="hide"**>  
 <**ul**>  
 <**li**><**a href="javascript:"**>赵</**a**></**li**>  
 <**li**><**a href="javascript:"**>钱</**a**></**li**>  
 <**li**><**a href="javascript:"**>孙</**a**></**li**>  
 <**li**><**a href="javascript:"**>李</**a**></**li**>  
 </**ul**>  
 </**div**>  
 <**script**>  
 */\*\* 说明：  
 \* 目前通过ul外层div自定义属性“data-status”控制其是否显示  
 \*/* (**function**($){  
 $(**"input"**).focus(**function**(){  
 *// input框获取焦点，展示下拉框* $(**".search-list"**).attr(**"data-status"**, **"show"**);  
 }).blur(**function**(){  
 *// input框失去焦点，隐藏下拉框* $(**".search-list"**).attr(**"data-status"**, **"hide"**);  
 });  
 *// 选择对应选项，并赋值给input框* $(**".search-list li a"**).click(**function**(){  
 **console**.log(**"执行"**);  
 $(**"input"**).val($(**this**).text());  
 });  
 })(**jQuery**);  
 </**script**>  
 </**body**>  
</**html**>

执行上述代码，会出现一个问题，并不能正确获取下拉框中某值。   
原因：由于JavaScript为单线程，同一时间只能执行处理一个事件。由上述示例3，可得知“blur优先于click执行”。而在本示例中，由于blur处理程序，会将对应的下拉框展示区隐藏，所以导致其后续click事件并不会执行。上述console的信息也不会被输出。

解决方案1：对blur事件进行延迟，让click先执行。

三、使用mousedown让其优先执行

示例4：将示例3中的click事件改为mousedown

<**input type="text" id="ipt"**>  
<**input type="button" id="btn" value="点我"**>  
<**script**>  
 **var *ipt*** = **document**.getElementById(**"ipt"**);  
 ***ipt***.addEventListener(**"blur"**, **function**(){  
 **console**.log(**"my input blur"**);  
 });  
  
 **var *btn*** = **document**.getElementById(**"btn"**);  
 ***btn***.addEventListener(**"mousedown"**, **function**(){  
 **console**.log(**"my button mousedown"**);  
 });  
</**script**>  
// 输出结果：  
my button mousedown  
my input blur

js动态修改整个页面样式（换肤）（面试题 掌握）

<!DOCTYPE **html**>  
<**html xmlns="http://www.w3.org/1999/xhtml"**>  
<**head**>  
 <**meta http-equiv="Content-Type" content="text/html; charset=utf-8"**/>  
 <**title**>动态修改页面样式</**title**>  
 <**link href="css/blue.css" rel="stylesheet" id="mylink"**/>  
 <**script type="text/javascript"**>  
 **function** *gel*(id) {  
 **return document**.getElementById(id);  
 }  
  
 **window**.onload = **function**() {  
 *//更换css文件* **var** lis = *gel*(**"uList"**).**childNodes**;  
 **for** (**var** i = 0; i < lis.**length**; i++) {  
 **if** (lis[i].**nodeType** == 1) {  
 lis[i].onclick = **function** () {  
 *gel*(**"mylink"**).**href** = **"css/"** + **this**.**className** + **".css"**;  
 };  
 }  
 }  
 };  
  
 </**script**>  
</**head**>  
  
<**body**>  
<**div**>  
 <**span**>修改整个页面的样式</**span**><**br**/>  
 <**input type="text" id="txt"**/>  
 <**input type="button" value="提交" class="btn"**/>  
</**div**>  
  
<**ul id="uList" style="text-decoration**: **none**;**margin-top**: 100**px**;**"**>  
 <**li style="display**: **block**;**width**: 30**px**;**height**: 20**px**;**background-color**: **red**;**" class="red"**></**li**>  
 <**li style="display**: **block**;**width**: 30**px**;**height**: 20**px**;**background-color**: **blue" class="blue"**></**li**>  
</**ul**>  
</**body**>  
</**html**>

Red.css样式

<**style**>  
 **\*** {  
 **margin**: 0**px**;**padding**: 0**px**;  
 }  
  
  
 **body** {  
 **background-color**: **#eeeeee**;  
 }  
 **span** {  
 **color**: **red**;  
 }  
 **#txt** {  
 **color**: **#f00**;**border**: 1**px solid #7d1515**;  
 }  
 .**btn** {  
 **background-color**: **#a52a2a**;**border**: **none**;**color**: **white**;**width**: 100**px**;**height**: 28**px**;  
 }  
</**style**>

Blue.css样式

<**style**>  
 **\*** {  
 **margin**: 0**px**;**padding**: 0**px**;  
 }  
  
  
 **body** {  
 **background-color**: **#eeeeee**;  
 }  
 **span** {  
 **color**: **blue**;  
 }  
 **#txt** {  
 **color**: **#0000cd**;**border**: 1**px solid #006400**;  
 }  
 .**btn** {  
 **background-color**: **#0000cd**;**border**: **none**;**color**: **white**;**width**: 100**px**;**height**: 28**px**;  
 }  
</**style**>

javascript之解决dom中存在的空白节点问题（面试题 掌握）

<**html**>  
<**head**>  
 <**title**>Introduction to the DOM</**title**>  
 <**script**>  
 *// We can't manipulate the DOM until the document  
 // is fully loaded* **window**.onload = **function**(){  
 *// Locate the element with an ID of 'everywhere'* **var** every = **document**.getElementById( **"everywhere"** );  
 *// and remove it from the document* **var** a=every.**parentNode**;  
 **console**.info(a.**childNodes**[0].**nodeType**);  
  
 *cleanWhitespace*(a)  
 **console**.info(a.**childNodes**[0].**nodeType**);  
 *//清除空白函数* **function** *cleanWhitespace*(oEelement){  
 **for**(**var** i=0;i<oEelement.**childNodes**.**length**;i++){  
 **var** node=oEelement.**childNodes**[i];  
 **if**(node.**nodeType**==3 && !/\S/.test(node.**nodeValue**)){  
 node.parentNode.removeChild(node)  
 }  
 }  
 }  
  
 };  
  
  
 </**script**>  
</**head**>  
<**body**>  
<**h1**>Introduction to the DOM</**h1**>  
<**p class="test"**>There are a number of reasons why the DOM is awesome, here are some:</**p**>  
<**ul**>  
 <**li id="everywhere"**>It can be found everywhere.</**li**>  
 <**li class="test"**>It's easy to use.</**li**>  
 <**li class="test"**>It can help you to find what you want, really quickly.</**li**>  
</**ul**>  
</**body**>  
</**html**>

# Javascript中事件捕获、事件冒泡以及事件委托机制

# 概念

事件冒泡：事件触发的最深层元素首先接收事件。然后是它的父元素，依次向上，直到document对象最终接收到事件。尽管相对于html元素来说，document没有独立的视觉表现，他仍然是html元素的父元素并且事件能冒泡到document元素。   
随便也说一下事件捕获。   
事件捕获：事件首先发生在DOM树的最高层对象(document)然后往最深层的元素传播。（注意IE6只有冒泡，没有捕获）   
事件委托：我认为事件委托是利用冒泡原理，把事件的监听转换到其父元素上，也就是把事件绑定到父元素上，然后在事件中获取子元素对象，对其进行相应的操作。优点：1.提高性能2.减少代码量

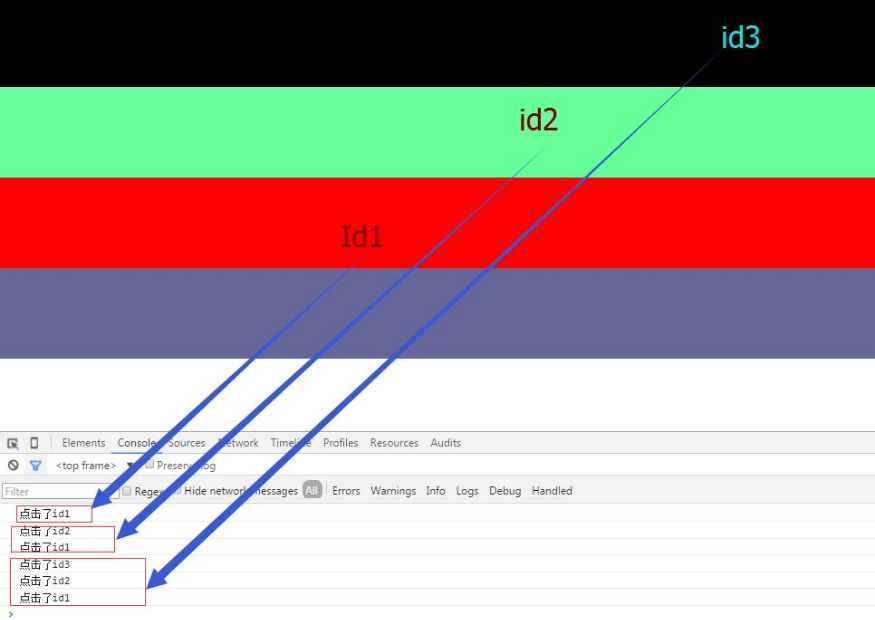
# 实例

## 事件冒泡实例1

事件默认是在冒泡阶段执行   
首先看下面代码：

<!DOCTYPE **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**style type="text/css"**>  
 **\***{**margin**: 0;**padding**: 0;}  
 </**style**>  
 </**head**>  
 <**body**>  
 <**div id="box" style="background-color**:**#669**;**widht**:600**px**; **height**:400**px**;**"**>  
 <**div id="id1" style="background-color**:**#F00**;**widht**:500**px**;**height**:300**px**;**"**>  
 <**div id="id2" style="background-color**:**#6F9**;**widht**:400**px**; **height**:200**px**;**"**>  
 <**div id="id3" style="background-color**:**#000**;**widht**:300**px**; **height**:100**px**;**"**></**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 <**script**>  
 **window**.onload=**function**(){  
 **var** oId1=**document**.getElementById(**'id1'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
 **var** oId3=**document**.getElementById(**'id3'**);  
   
 oId1.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id1"**);  
 });  
   
 oId2.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id2"**);  
   
 });  
   
 oId3.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id3"**);  
 });  
 }  
 </**script**>  
 </**body**>  
</**html**>

我依次点击id1、id2、id3，执行效果如下图：



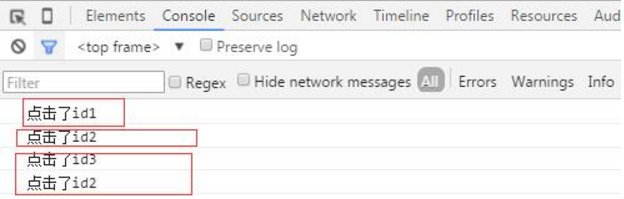
解析：因为点击id3时候，先从id3开始冒泡，执行id3上绑定的事件，在冒泡到id2，执行id2，上面的事件，最后执行id1上面的事件。

事件冒泡实例2

现在开始阻止id2的冒泡，修改JS如下

<**script**>  
 **window**.onload=**function**(){  
 **var** oId1=**document**.getElementById(**'id1'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
 **var** oId3=**document**.getElementById(**'id3'**);  
  
 oId1.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id1"**);  
 });  
  
 oId2.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id2"**);  
 e.stopPropagation();  
  
 });  
  
 oId3.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id3"**);  
 });  
 }  
</**script**>

此时，我依次点击id1、id2、id3，执行效果如下图：



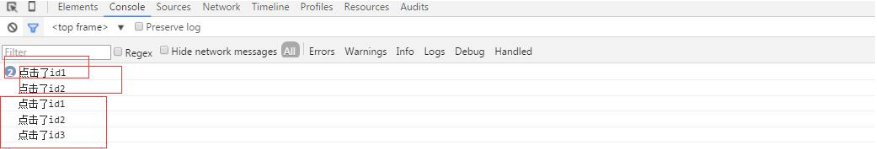
因为事件执行到id2，不在冒泡，故点击id2，id3的时候，不会执行id1绑定的事件。

事件捕获实例1

为了验证事件在捕获阶段执行，我将JS代码改为如下：

<**script**>  
 **window**.onload=**function**(){  
 **var** oId1=**document**.getElementById(**'id1'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
 **var** oId3=**document**.getElementById(**'id3'**);  
  
 oId1.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id1"**);  
 },**true**);  
  
 oId2.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id2"**);  
  
 },**true**);  
  
 oId3.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id3"**);  
 },**true**);  
 }  
</**script**>

此时，我依次点击id1、id2、id3，执行效果如下图：



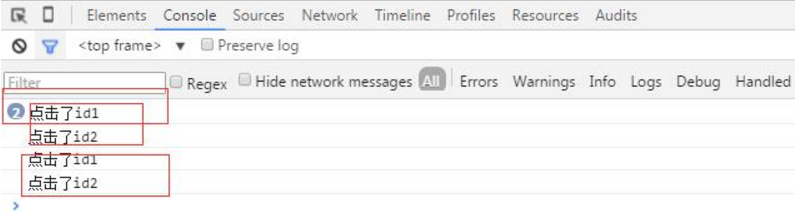
解析：每次点击，事件都会从根元素开始开始执行，即捕获到，如果有事件，就执行。

事件捕获实例2

此时，我将JS代码改为如下：

<**script**>  
 **window**.onload=**function**(){  
 **var** oId1=**document**.getElementById(**'id1'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
 **var** oId3=**document**.getElementById(**'id3'**);  
  
 oId1.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id1"**);  
 },**true**);  
  
 oId2.addEventListener(**'click'**,**function**(e){  
 e.stopPropagation();  
 **console**.log(**"点击了id2"**);  
  
 },**true**);  
  
 oId3.addEventListener(**'click'**,**function**(e){  
  
 **console**.log(**"点击了id3"**);  
 },**true**);  
 }  
  
</**script**>

此时，我依次点击id1、id2、id3，执行效果如下图：



从中发现了，一个现象，我点击id3的时候，发现执行了id1和id2上面绑定的事件，为什么不执行id3上面的事件呢？原来是因为取消冒泡e.stopPropagation();，也阻止了事件的捕获。

事件冒泡与捕获实例

现在我将JS修改为如下：

<**script**>  
 **window**.onload=**function**(){  
 **var** oId1=**document**.getElementById(**'id1'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
 **var** oId3=**document**.getElementById(**'id3'**);  
  
  
 oId1.onclick=**function**(){ *//该事件在冒泡阶段执行* **console**.log(**"点击了id1"**);  
 }  
  
 oId2.addEventListener(**'click'**,**function**(e){  
 **console**.log(**"点击了id2"**);  
  
 },**true**);  
  
 oId3.addEventListener(**'click'**,**function**(e){  
  
 **console**.log(**"点击了id3"**);  
 },**true**);  
 }  
</**script**>

此时，我依次点击id1、id2、id3，执行效果如下图：

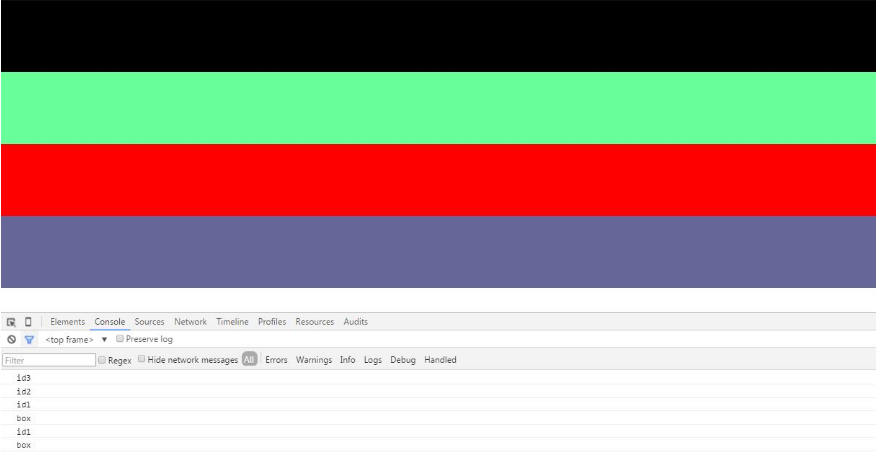


事件委托实例1

下面代码，当我在box绑定click事件的时候，通过e.srcElement，可以获取点击的是哪个元素。

<!DOCTYPE **html**>  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>Title</**title**>  
 <**style type="text/css"**>  
 **\***{**margin**: 0;**padding**: 0;}  
 </**style**>  
 </**head**>  
 <**body**>  
 <**div id="box" style="background-color**:**#669**;**widht**:600**px**; **height**:400**px**;**"**>  
 <**div id="id1" style="background-color**:**#F00**;**widht**:500**px**;**height**:300**px**;**"**>  
 <**div id="id2" style="background-color**:**#6F9**;**widht**:400**px**; **height**:200**px**;**"**>  
 <**div id="id3" style="background-color**:**#000**;**widht**:300**px**; **height**:100**px**;**"**></**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 <**script type="text/javascript"**>  
 **window**.onload=**function**(){  
 **var** oBox=**document**.getElementById(**'box'**);  
 oBox.onclick=**function**(e){  
 **var** curObj=e.**srcElement**;  
 **console**.log(curObj.**id**);  
 }  
 }  
  
 </**script**>  
 </**body**>  
</**html**>

最终结果如下:

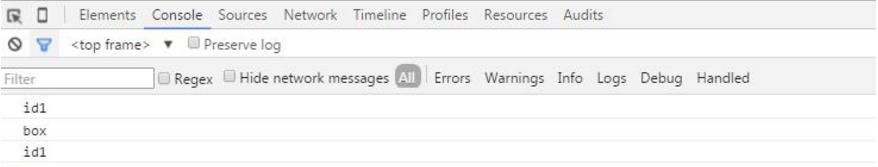


可以看出，我们在box绑定click事件里面能获取点击的元素。

事件委托实例2

验证事件委托中冒泡   
将上面的JS改为如下：

<**script type="text/javascript"**>  
 **window**.onload=**function**(){  
 **var** oBox=**document**.getElementById(**'box'**);  
 **var** oId2=**document**.getElementById(**'id2'**);  
  
 oId2.onclick=**function**(e){  
 e.stopPropagation();  
 }  
 oBox.onclick=**function**(e){  
 **var** curObj=e.**srcElement**;  
 **console**.log(curObj.**id**);  
 }  
 }  
</**script**>

此时点击，会发现，id2，id3的时候，没法通过e.srcElement获取到。因为我阻止了id2的冒泡。 

# 一张图轻松搞懂JavaScript event对象的clientX,offsetX,screenX,pageX区别

总是会被javascript的event对象的clientX,offsetX,screenX,pageX 弄得头晕，于是决定做个图来区分一下(画得我手那个酸呀。。。。)

先总结下区别：

event.clientX、event.clientY

鼠标相对于浏览器窗口可视区域的X，Y坐标（窗口坐标），可视区域不包括工具栏和滚动条。IE事件和标准事件都定义了这2个属性

event.pageX、event.pageY

类似于event.clientX、event.clientY，但它们使用的是文档坐标而非窗口坐标。这2个属性不是标准属性，但得到了广泛支持。IE事件中没有这2个属性。

event.offsetX、event.offsetY

鼠标相对于事件源元素（srcElement）的X,Y坐标，只有IE事件有这2个属性，标准事件没有对应的属性。

event.screenX、event.screenY

鼠标相对于用户显示器屏幕左上角的X,Y坐标。标准事件和IE事件都定义了这2个属性

上图！！！！

# 

**JavaScript原型继承机制**

JavaScript并非一个纯粹的面向的对象的语言，也没有一个实际意义上的继承机制。语言的设计者最初并未想实现继承机制，但是后来还是实现了一个类似的机制——原型继承机制。但实际上，这并非实际的继承，我们来一一揭开JavaScript的伪继承机制。  
一、typeof 和 instanceof

       在JavaScript中，typeof是测试对象的类型，只会测试出对象属于哪种内置类型，无法测试自定义类型。也就是所有自定义类型都会返回object。

       而obj instanceof sometype(obj 指对象，sometype指的是类型)，是测试obj的原型链中是否有sometype类型。所以通过错误的改变prototype的值会使得instanceof得出的结构不合理。

1. typeof

<**script**>  
 **var *num*** = 10;  
 **document**.write(**"typeof num:"**+typeof ***num***+**"<br/>"**);  
 **var *onum*** = Number(10);  
 **document**.write(**"typeof onum:"**+typeof ***onum***+**"<br/>"**);  
  
 **var *str*** = **"1010"**;  
 **document**.write(**"typeof str:"**+typeof ***str***+**"<br/>"**);  
 **var *ostr*** = String(**"1010"**);  
 **document**.write(**"typeof ostr:"**+typeof ***ostr***+**"<br/>"**);  
  
 **var *bol*** = **true**;  
 **document**.write(**"typeof bol:"**+typeof ***bol***+**"<br/>"**);  
 **var *obol*** = **true**;  
 **document**.write(**"typeof obol:"**+typeof ***obol***+**"<br/>"**);  
  
 **var *dat*** = **new** Date();  
 **document**.write(**"typeof dat:"**+typeof ***dat***+**"<br/>"**);  
  
 **var *func*** = **new Function**(**'x'**,**'y'**,**"return x+y;"**);  
 **document**.write(**"typeof func:"**+typeof ***func***+**"<br/>"**);  
  
 **var** *myconstructor* = **function** Person(){};  
 **document**.write(**"typeof myconstructor:"**+typeof *myconstructor*+**"<br/>"**);  
  
 **var *myobj*** = **new** *myconstructor*();  
 **document**.write(**"typeof myobj:"**+typeof ***myobj***+**"<br/>"**);  
</**script**>

结果：

typeof num:number  
typeof onum:number  
typeof str:string  
typeof ostr:string  
typeof bol:boolean  
typeof obol:boolean  
typeof dat:object  
typeof func:*function*typeof myconstructor:*function*typeof myobj:object

这说明元构造器的类型为：function  
内建的普通构造器类型为：function  
自定义普通构造器类型为：function  
所有内置对象类型为：内置类型  
所有自定义普通对象：object

2、instanceof

实际就是测试实例是否有继承基类的原型链，如obj  instanceof Type,若obj的原型链中Type则返回true，否则返回false。

<**script**>  
 *//案例一* **var** *Person* = **function**(){};  
 **var *obj*** = **new** *Person*();  
 **document**.write((***obj*** instanceof *Person*) +**"<br/>"**);  
 **document**.write((***obj*** instanceof Object) +**"<br/>"**);  
 **document**.write(***obj***.\_\_proto\_\_ +**"<br/>"**);  
 **document**.write(*Person*.**constructor**+**"<br/>"**);  
 **document**.write(***obj***.**constructor**+**"<br/>"**);  
 **document**.write((***obj***.**constructor** === *Person*)+**"<br/>"**);  
</**script**>

结果：

**true  
true** [object Object]  
**function** *Function*() { [native code]}  
*function* (){}  
**true**

<**script**>  
 *//案例二* **var** *other* = **function**(){};  
 Person.**prototype** = **new** *other*();  
 **document**.write((obj instanceof Person) +**"<br/>"**);  
 **document**.write((obj instanceof Object) +**"<br/>"**);  
 **document**.write((obj instanceof *other*) +**"<br/>"**);  
 **document**.write(obj.\_\_proto\_\_ +**"<br/>"**);  
 **document**.write(obj.**constructor**+**"<br/>"**);  
</**script**>

结果：

**false  
true  
false** [object Object]  
*function* (){}

*//案例三*obj.\_\_proto\_\_ = **new** other();  
**document**.write((obj instanceof Person) +**"<br/>"**);  
**document**.write((obj instanceof Object) +**"<br/>"**);  
**document**.write((obj instanceof other) +**"<br/>"**);  
**document**.write(obj.\_\_proto\_\_ +**"<br/>"**);  
结果：  
**false  
true  
true** [object Object]

下面我们就来具体分析这三个案例：

1. 案例一

var Person = function(){};  
这句具体发生了什么呢？我们用下面的伪代码来解释一下：生成了一个函数对象Person

{  
 **this**.**prototype** = {**constructor**:**this**}; *//{}表示一个Object构造的空对象* **var *prototype*** = **Function**.**prototype**; *//内部{Prototype}属性，不能改变* **this**.**\_\_proto\_\_** = ***prototype***; *//内部属性的外部访问方式，* **this**.**constructor** = **this**; *// 默认的，也是合理的值*}

 var obj = new Person();

这句发生了什么呢？我们也用伪代码来解释一下：生成普通对象obj

{  
 **var *prototype*** = Person.**prototype**;  
 **this**.**\_\_proto\_\_** = ***prototype***;  
 **this**.**constructor** = Person;  
}

按此来说，在案例一中：

*//function类型*Person.\_\_proto\_\_=== **Function**.**prototype**;  
*//object类型, 默认情况下每个构造器内部都有一个Object的实例对象，供所有子类共享,这就是所有对象都默认继承object的实例的原因*Person.**prototype** === obj.\_\_proto\_\_ ;  
*//验证每个自定义类型都默认继承自一个Object实例object*Person.**prototype**.\_\_proto\_\_ = Object.**prototype**;  
*//每个对象的构造器为构造它的函数对象*Person.**constructor** === **Function**;  
**Function**.**constructor** === **Function**; *//说明Function由自己构造，自举性*obj.**constructor** === Person;  
*//从中可以看出prototype告诉构造器，你构造的对象应该继承谁  
//\_\_proto\_\_可以告诉我们该对象继承自谁。*

其次我们可以看出：

*//Function 的自举性***Function**.**prototype** === **Function**.**prototype**;  
**Function**.\_\_proto\_\_ === **Function**.**prototype**;  
**Function**.**constructor** === **Function**;  
*//Object由Function构造*Object.**prototype** === Object.**prototype**;  
Object.\_\_proto\_\_ === **Function**.**prototype**;  
Object.**constructor** === **Function**;  
**Function**.**prototype**.\_\_proto\_\_ === Object.**prototype**; *//说明Object.prototype是万物之源*

而obj对象呢，也有着这些特性，不过它是普通对象，没有公开属性prototype:

*//普通对象特性*obj.\_\_proto\_\_ === Person.**prototype**;  
obj.**constructor** === Person;

我们再来看看Object.prototype:

*//元对象*Object.**prototype** === Object.**prototype**; *//对象的祖宗，根*Object.**prototype**.\_\_proto\_\_ === **null**; *//祖先没有祖先，单位了系统完备，其内部仍然有{Prototype}属性，置为null，合理。*Object.**prototype**.**constructor** === Object; *//祖先按理说是没有构造器的，但为了说明祖先仍然属于Object的实例，就这么做。*

接着我们再看看Function.prototype:

*//元构造器***Function**.**prototype** === **Function**.**prototype**; *//由Function构造出来的对象的原型***Function**.**prototype**.\_\_proto\_\_ === Object.**prototype**; *//对象Function.prototype的原型为Object.prototype***Function**.**prototype**.**constructor** === **Function**;  
*//对象Function.prototype是由构造器Function构造的，实际上并非如此，但是为了Function.prototype的类型是Function也就这么设计了。*

二、prototype和{Prototype}（即\_\_proto\_\_）

        在Js中有两个顶级上司，或者说顶级公民：Function.prototype 和Object.prototype。

1.一切都是对象

        在Js中一切都是对象，而几乎所有对象都由构造器产生。不过有一个例外，所有对象的基础（元对象）：Object.prototype，他不是由构造器产生，而是由Js引擎实现。它不同于java的Object——那是一个类，而Js中的Object.prototype仅仅是一个对象，它不能作为构造器。这来来讲就好比西方人类最初也只有两个人作为祖先——亚当和夏娃。但是在Js中只有一个没有构造功能的Object.prototype那么如何基于这个原型来构造子对象呢，这就需要下面讲到的元构造器——Function.prototype。

2.Object.prototype不是由构造器产生

3.Function.prototype由自己构造，但是却是继承自Object.prototype

4.继承只是继承对象，所谓对象A继承自B，说明对象A维护者对象B的引用

5.函数对象中的\_\_proto\_\_才是指明该函数对象作为对象时继承自谁，或者说维护了哪个对象的引用

6.函数对象中有个特别的特性prototype，这是普通对象所不具有的，这是函数对象作为构造器时的特性，它指明了该构造器作为”伪类“时的内部数据结构，他是一个引用类型。由该构造器构造出来的对象都维护着prototype所引用的对象的引用

           事实上C/C++可以实现类似的功能。