**g0001——添加多个事件，不覆盖。**

**IE8及以下：先绑定后执行**

**其他浏览器IE9 ：先绑定先执行**

g0001:function(){

console.log('g0001生效！');

function addEvent(dom,type,fn) {

if (dom.addEventListener) {

dom.addEventListener(type,fn,false);

}else if(dom.attachEvent){

dom.attachEvent('on'+type,fn);

}else{

dom['on'+type]=fn;

}

}

var fun1=function(){

console.log(1);

}

var fun2=function(){

console.log(2);

}

var doc=document;

addEvent(doc,'click',fun1);

addEvent(doc,'click',fun2);

}

**g0002——跨域问题**

**何为跨域？**

**一个域中的脚本仅仅有本域的权限：也就是本域脚本只能读写本域内的资源，而无法访问其它域的资源。**

同一域名不同端口号，

同一域名不同协议

域名和域名IP [www.baidu.com](http://www.baidu.com) 61.135.169.125

主域与子域 [www.bai.com](http://www.bai.com) www.a.com

子域与子域 [www.tieba.baidu.com](http://www.tieba.baidu.com) [www.fanyi.baidu.com](http://www.fanyi.baidu.com)

**同源策略的意义**

**安全性：web前端安全的基石，没有同源策略，浏览器就没有了安全性可言。**

1. **浏览器cookie权限如果没有同源策略，会被任意网站获取。用户信息（包括一些网站的登录信息，浏览习惯信息）会被盗取。**

**同源策略限制范围**

1. **无法共享cookie ,localStorage ,indexDB （如果共享，则会使得用于浏览信息或用户信息泄露）**
2. **无法通过flash发送http请求**
3. **无法发送ajax请求（服务器拒绝非同源的请求，返回403）**
4. **无法操作彼此dom元素（避免如iframe引入他人页面，通过父级页面操作iframe。获取用户信息或浏览信息，或者一个定时疯狂提交）。**
5. **其他**

**同源策略的弊端**

1. 牺牲了资源跨域请求，对于大型网站，视频，图片，js，html，css不可能全部布置在一台服务器上。

**规避同源策略的几种需求**

1. **跨域引用资源 script img style 。**

**只是引用这些资源，并不能读取这些资源的内容。**

1. **拥有相同的一级域名访问cookie。如a.baidu.com , b.baidu.com**

**2.1前端cookie设置**

可以在a.baidu.com下的一个页面通过 设置：

document.domain=’baidu.com’

document.cookie=”cross=yes”

或者

document.coolie=’cross=yes;path=/;domain=baidu.com’

这样这个cookie在所有baidu.com下的子域名都可以访问。

**2.2后端cookie设置**将response中对的cookie的domain设置为.baidu.com

**3 ajax请求**

**3.1 CORS（实际上服务器不会拦截跨域请求，数据任然会返回，但是没有设置response的返回头，浏览器将会拦截。也就是说实际上控制权在浏览器。）**

需要在服务端进行设置。

CORS在浏览器中如何运行

**对于简单请求**

**浏览器端**

浏览器如果发现ajax请求跨域，会在Header上**自动**加上一个origin(protocal + host + path + port)字段，标明这个请求来自于哪里。

CORS请求默认不携带cookie之类的信息。如果需要携带用户信息（如需要判断用户身份）可以将withCredentials设置为true

如

var xhr=new XMLHttpRequest()

xhr.withCredentials=true;

设置了这个值后，**需要设置**服务端的response中的Access-Control-Allow-Credentials也会设置为true.

**服务端**

服务端获得请求后，对origin进行判断，如果是允许范围的请求，会在response返回的Header中加上

Access-Control-Allow-Origin: origin 接受哪些域名的访问，为\*或者单一值（不支持多个值）

Access-Control-Allow-Credentials: true 是否拥有用户相关权限。如有为true,则必须origin不能设置为\*，必须设置为单一值。

Access-Control-Expose-Headers: something 其他可以被返回的头部数据

由于简单请求，response的头信息中，浏览器只能拿到以下几个基本字段（`Cache-Control`, `Content-Language`, `Content-Type`, `Expires`, `Last-Modified`, `Pragma`） 。如需额外信息，只能在Access-Control-Expose-Headers中设置

如：Access-Control-Expose-Headers: "Foo=foo"，浏览器就可以获取Foo这个字段所携带的信息了。

**对于非简单请求**

不同于简单请求，非简单请求实际上发送了两个请求。

**预请求：**

这个请求尽可能少携带信息，仅供服务器判断是否响应该请求。

Request Method会被设置为options

附带字段

origin:

Access-Control-Request-Method: 请求将要使用的方法

Access-Control-Request-Headers: 浏览器会额外发送哪些头信息

如果否定，直接返回不带这三个字段的response,浏览器将会这种返回判断为失败的返回。触发onerror

如果正确响应，再次请求同简单请求

**3.2 websocket**

**websocket在浏览器端没有做任何同源限制**

websocket在请求头中会带上origin这个字段，服务端可以通过这个字段判断是否因该响应。

**3.3 jsonp**

本质上是一种hack形式的请求，只是对静态文件的请求。因为只是支持对静态文件资源的请求，所以只支持GET请求。

**4 iframe 跨域**

**iframe中的同源策略**

如果连个页面不同源。则他们相互隔离

不同源你的两个网页不能：

1.操作彼此的DOM

2.获取/调用彼此window对象的属性/方法

不同源的两个网页可以：

改变父/子级的url

**iframe跨域的三种解决方案**

**4.1 使用片段标识符**

也就是hash（#后的） location.hash

片段标识符的几个特点：

1.改变url中hash，不触发刷新

2.父/子级可以改变，iframe的url。

3.window对象可以监听hashchange事件

由此可以实现基于hashchange操作页面。

**4.2 window.name**

**基本原理：**

window对象没有改变，在这个window中跳转的网页，都可以读取window.name;

也就是说在与父页面A非同源的子页面B中设置window.name，再改变B页面调到与父级页面同源的地址中，这时父页面A可以获取B的window.name了。

**4.3 跨文档通信API(cross-document messaging)**

上述两个实现都是hack方法。

HTML5中添加了一个window.postMessage的方法

// Page Foo

iframe.contentWindow.postMessage('Hello from foo', '/path/to/bar')

// Page Barwindow.parent.addEventListener('message', function (e) {

console.log(e.source) // 发送消息的窗口

console.log(e.origin) // 消息发向的网址

console.log(e.data) // 消息内容

})

1. 其他（canvas与flash）

详细网址：<https://segmentfault.com/a/1190000007366644>

5.1 canvas跨域

canvas.toDataURL

canvas.toBlob

canvas.getContent('2d').getImageData(x,y,w,h)

2.5 flash

**g0003——前端生成验证码**

**g0004——类型检测**

现有的原生类型检测的局限性

null undefined string number boolean function

对于null undefined NaN的检测比较方便。

Object.prototype.toString()

直接输出对象内部的[[Class]],绝对精确

当然，其也是有缺陷的，toString()只能针对原生数据类型，而对是否为window ,是否为纯净的JavaScript对象无能为力。