https://blog.csdn.net/yup1212/article/details/87633272

**1.同源策略如下：**

|  |  |  |
| --- | --- | --- |
| **URL** | **说明** | **是否允许通信** |
| http://www.a.com/a.js http://www.a.com/b.js | 同一域名下 | 允许 |
| http://www.a.com/lab/a.js http://www.a.com/script/b.js | 同一域名下不同文件夹 | 允许 |
| http://www.a.com:8000/a.js http://www.a.com/b.js | 同一域名，不同端口 | 不允许 |
| http://www.a.com/a.js https://www.a.com/b.js | 同一域名，不同协议 | 不允许 |
| http://www.a.com/a.js http://70.32.92.74/b.js | 域名和域名对应ip | 不允许 |
| http://www.a.com/a.js http://script.a.com/b.js | 主域相同，子域不同 | 不允许 |
| http://www.a.com/a.js http://a.com/b.js | 同一域名，不同二级域名（同上） | 不允许（cookie这种情况下也不允许访问） |
| http://www.cnblogs.com/a.js http://www.a.com/b.js | 不同域名 | 不允许 |

特别注意两点：

第一，如果是协议和端口造成的跨域问题“前台”是无能为力的，

第二：在跨域问题上，域仅仅是通过“URL的首部”来识别而不会去尝试判断相同的ip地址对应着两个域或两个域是否在同一个ip上。

*“URL的首部”指window.location.protocol +window.location.host，也可以理解为“Domains, protocols and ports must match”。*

**2. 前端解决跨域问题**

**1> document.domain + iframe      (只有在主域相同的时候才能使用该方法)**

1) 在www.a.com/a.html中：

1. document.domain = 'a.com';
2. var ifr = document.createElement('iframe');
3. ifr.src = 'http://www.script.a.com/b.html';
4. ifr.display = none;
5. document.body.appendChild(ifr);
6. ifr.onload = function(){
7. var doc = ifr.contentDocument || ifr.contentWindow.document;
8. //在这里操作doc，也就是b.html
9. ifr.onload = null;
10. };

2) 在www.script.a.com/b.html中：

document.domain = 'a.com';

**2> 动态创建script**

这个没什么好说的，因为script标签不受同源策略的限制。

1. function loadScript(url, func) {
2. var head = document.head || document.getElementByTagName('head')[0];
3. var script = document.createElement('script');
4. script.src = url;
6. script.onload = script.onreadystatechange = function(){
7. if(!this.readyState || this.readyState=='loaded' || this.readyState=='complete'){
8. func();
9. script.onload = script.onreadystatechange = null;
10. }
11. };
13. head.insertBefore(script, 0);
14. }
15. window.baidu = {
16. sug: function(data){
17. console.log(data);
18. }
19. }
20. loadScript('http://suggestion.baidu.com/su?wd=w',function(){console.log('loaded')});
21. //我们请求的内容在哪里？
22. //我们可以在chorme调试面板的source中看到script引入的内容

**3> location.hash + iframe**

原理是利用location.hash来进行传值。

假设域名a.com下的文件cs1.html要和cnblogs.com域名下的cs2.html传递信息。  
1) cs1.html首先创建自动创建一个隐藏的iframe，iframe的src指向cnblogs.com域名下的cs2.html页面  
2) cs2.html响应请求后再将通过修改cs1.html的hash值来传递数据  
3) 同时在cs1.html上加一个定时器，隔一段时间来判断location.hash的值有没有变化，一旦有变化则获取获取hash值  
注：由于两个页面不在同一个域下IE、Chrome不允许修改parent.location.hash的值，所以要借助于a.com域名下的一个代理iframe  
  
代码如下：  
先是a.com下的文件cs1.html文件：

1. function startRequest(){
2. var ifr = document.createElement('iframe');
3. ifr.style.display = 'none';
4. ifr.src = 'http://www.cnblogs.com/lab/cscript/cs2.html#paramdo';
5. document.body.appendChild(ifr);
6. }
8. function checkHash() {
9. try {
10. var data = location.hash ? location.hash.substring(1) : '';
11. if (console.log) {
12. console.log('Now the data is '+data);
13. }
14. } catch(e) {};
15. }
16. setInterval(checkHash, 2000);

cnblogs.com域名下的cs2.html:

1. //模拟一个简单的参数处理操作
2. switch(location.hash){
3. case '#paramdo':
4. callBack();
5. break;
6. case '#paramset':
7. //do something……
8. break;
9. }
11. function callBack(){
12. try {
13. parent.location.hash = 'somedata';
14. } catch (e) {
15. // ie、chrome的安全机制无法修改parent.location.hash，
16. // 所以要利用一个中间的cnblogs域下的代理iframe
17. var ifrproxy = document.createElement('iframe');
18. ifrproxy.style.display = 'none';
19. ifrproxy.src = 'http://a.com/test/cscript/cs3.html#somedata'; // 注意该文件在"a.com"域下
20. document.body.appendChild(ifrproxy);
21. }
22. }

a.com下的域名cs3.html

1. //因为parent.parent和自身属于同一个域，所以可以改变其location.hash的值
2. parent.parent.location.hash = self.location.hash.substring(1);

**4> window.name + iframe**

window.name 的美妙之处：**name 值在不同的页面（甚至不同域名）加载后依旧存在，并且可以支持非常长的 name 值（2MB）。**

1) 创建a.com/cs1.html

2) 创建a.com/proxy.html，并加入如下代码

1. <head>
2. <script>
3. function proxy(url, func){
4. var isFirst = true,
5. ifr = document.createElement('iframe'),
6. loadFunc = function(){
7. if(isFirst){
8. ifr.contentWindow.location = 'http://a.com/cs1.html';
9. isFirst = false;
10. }else{
11. func(ifr.contentWindow.name);
12. ifr.contentWindow.close();
13. document.body.removeChild(ifr);
14. ifr.src = '';
15. ifr = null;
16. }
17. };
19. ifr.src = url;
20. ifr.style.display = 'none';
21. if(ifr.attachEvent) ifr.attachEvent('onload', loadFunc);
22. else ifr.onload = loadFunc;
24. document.body.appendChild(iframe);
25. }
26. </script>
27. </head>
28. <body>
29. <script>
30. proxy('http://www.baidu.com/', function(data){
31. console.log(data);
32. });
33. </script>
34. </body>

3 在b.com/cs1.html中包含：

1. <script>
2. window.name = '要传送的内容';
3. </script>

**5> postMessage（HTML5中的XMLHttpRequest Level 2中的API）**

1) a.com/index.html中的代码：

1. <iframe id="ifr" src="b.com/index.html"></iframe>
2. <script type="text/javascript">
3. window.onload = function() {
4. var ifr = document.getElementById('ifr');
5. var targetOrigin = 'http://b.com'; *// 若写成'http://b.com/c/proxy.html'效果一样*
6. *// 若写成'http://c.com'就不会执行postMessage了*
7. ifr.contentWindow.postMessage('I was there!', targetOrigin);
8. };
9. </script>

2) b.com/index.html中的代码：

1. <script type="text/javascript">
2. window.addEventListener('message', function(event){
3. *// 通过origin属性判断消息来源地址*
4. if (event.origin == 'http://a.com') {
5. alert(event.data); *// 弹出"I was there!"*
6. alert(event.source); *// 对a.com、index.html中window对象的引用*
7. *// 但由于同源策略，这里event.source不可以访问window对象*
8. }
9. }, false);
10. </script>

**6> CORS**

CORS背后的思想，就是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功，还是应该失败。

IE中对CORS的实现是xdr

1. var xdr = new XDomainRequest();
2. xdr.onload = function(){
3. console.log(xdr.responseText);
4. }
5. xdr.open('get', 'http://www.baidu.com');
6. ......
7. xdr.send(null);

其它浏览器中的实现就在xhr中

1. var xhr = new XMLHttpRequest();
2. xhr.onreadystatechange = function () {
3. if(xhr.readyState == 4){
4. if(xhr.status >= 200 && xhr.status < 304 || xhr.status == 304){
5. console.log(xhr.responseText);
6. }
7. }
8. }
9. xhr.open('get', 'http://www.baidu.com');
10. ......
11. xhr.send(null);

实现跨浏览器的CORS

1. function createCORS(method, url){
2. var xhr = new XMLHttpRequest();
3. if('withCredentials' in xhr){
4. xhr.open(method, url, true);
5. }else if(typeof XDomainRequest != 'undefined'){
6. var xhr = new XDomainRequest();
7. xhr.open(method, url);
8. }else{
9. xhr = null;
10. }
11. return xhr;
12. }
13. var request = createCORS('get', 'http://www.baidu.com');
14. if(request){
15. request.onload = function(){
16. ......
17. };
18. request.send();
19. }

**7> JSONP**

JSONP包含两部分：回调函数和数据。

回调函数是当响应到来时要放在当前页面**被调用**的函数。

数据就是传入回调函数中的json数据，也就是回调函数的参数了。

1. function handleResponse(response){
2. console.log('The responsed data is: '+response.data);
3. }
4. var script = document.createElement('script');
5. script.src = 'http://www.baidu.com/json/?callback=handleResponse';
6. document.body.insertBefore(script, document.body.firstChild);
7. /\*handleResonse({"data": "zhe"})\*/
8. //原理如下：
9. //当我们通过script标签请求时
10. //后台就会根据相应的参数(json,handleResponse)
11. //来生成相应的json数据(handleResponse({"data": "zhe"}))
12. //最后这个返回的json数据(代码)就会被放在当前js文件中被执行
13. //至此跨域通信完成

 jsonp虽然很简单，但是有如下缺点：

1）安全问题(请求代码中可能存在安全隐患)

2）要确定jsonp请求是否失败并不容易

**8> web sockets**

web sockets是一种浏览器的API，它的目标是在一个单独的持久连接上提供全双工、双向通信。(同源策略对web sockets不适用)

web sockets原理：在JS创建了web socket之后，会有一个HTTP请求发送到浏览器以发起连接。取得服务器响应后，建立的连接会使用HTTP升级从HTTP协议交换为web sockt协议。

只有在支持web socket协议的服务器上才能正常工作。

1. var socket = new WebSockt('ws://www.baidu.com');//http->ws; https->wss
2. socket.send('hello WebSockt');
3. socket.onmessage = function(event){
4. var data = event.data;