## 源码：

<https://github.com/piLurk/origin-cross>

dongyuewu.me 网站源码 cross-orign 文件夹

a.dongyuew.me 网站源码 cross 文件夹

## 为什么存在同源策略

简单来说：

1. 非同源的页面不能产生交互（iframe）

主要是防止网站伪造，如果伪造者用一个irame将目标网站防止其网站内部，通过样式设置起来看起来没有任何不同，如果非同源页面可以交互，用户的输入都能被获取，包括密码。

1. 非同源的请求需要在授权的情况下才能读取数据（cors）,包括cookie传递

这是避免攻击者直接发送非同源请求（附带权限认证的cookie，如果他能拿到的h话），效果爆炸

但是针对cookie这个特殊的数据，非相同二级域下，无法共享（除非开启代理）。

## 跨域页面交互

1. document.domain + iframe跨域

[http://dongyuewu.me/pages/cross/iframe-cross/index.html](http://dongyuewu.me/iframe-cross/index.html)

**能实现同一二级域下的子域之间的iframe交互**

缺点：

必须在两个页面都设置同一个domain，且只能设置为自身或者上一级domain，而且都要设置

document.domain = ‘dongyuewu.me’

1. location.hash + iframe

[http://dongyuewu.me/pages/cross/hash-cross/a.html](http://dongyuewu.me/hash-cross/a.html)

**能实现不同二级域之间的iframe数据交互**

不同的二级域，父页面a可以操作子页面b的hash，但是子页面不能操作父页面的hash，因此需要在b页面引入与父页面在同一域的孙页面c。

a ------> b ----> c ----> a

1. a页面改变b页面的hash
2. b页面监控到hash改变，然后变c页面的hash
3. c页面监控到hash改变，然后将数据传入a页面

3.window.name + iframe跨域

[http://dongyuewu.me/pages/cross/name-cross/a.html](http://dongyuewu.me/name-cross/a.html)

**能实现不同二级域的单向数据获取**

原理：window.name有一个特殊的表现，在一个框架中，就算其location切换了，这个框架的window.name亦然会存在。但是在父级页面a不能获取非同源子页面b的框架的name.所以这里也需要一个与a页面同源的中间页c。

## 数据交互方案

4. postMessage跨域

[http://dongyuewu.me/pages/cross/postMessage-cross/a.html](http://dongyuewu.me/postMessage-cross/a.html)

**能实现不同二级域的数据交互**

**包括多窗口之间，iframe嵌套 之间的数据交互**

**获取到目标**

H5新方法

otherWindow..postMessage(data,origin)

window. addEventListener('message', function(e) {

alert('data from domain2 ---> ' + e.data);

}, false);

otherWindow：其他窗口的引用。

如 window.frames iframe.contentWindow 等等

origin:

‘\*’任意页面

‘/’同源页面

‘domain.com’具体页面

1. 通过jsonp跨域

[http://dongyuewu.me/pages/cross/jsonp/jsonp.html](http://dongyuewu.me/jsonp/jsonp.html)

**能实现不同二级域之间的通信**

根本原理在于，服务端监听到传过来的回调函数fn

返回一个全局脚本

fn（{...}）

缺点：

必须有一个fn全局函数。且只能是get请求

6. 跨域资源共享（CORS） -- 重点

一般而言，不需要前端请求有任何设置，浏览器会帮我们设置好一些需要的

请求头部，但是如果想要传递cookie时，需要设置withCredentials属性（有些浏览器会默认发送cookie）。

简单请求与复杂请求

简单请求可以认为就是可以通过普通form提交的请求，复杂请求则相反。

为什么简单请求没有CROS-preflight？因为就算不依赖xhr，表单也能发起跨域简单请求。

简单请求

浏览器直接发送请求

浏览器会在头部信息汇总加一个origin字段，表示发起资源请求的源是什么

服务器会更具origin，决定是否同意请求

同意的话：

返回的头部信息会有

Access-Control-Allow-Origin(请求该资源支持的域)

‘\*’表示所有域都可以请求这个资源

Access-Control-Allow-Credentials

返回是否携带cookie，默认不发送cookie 。如果是true，则Access-Control-Allow-Origin不能为‘\*’，Access-Control-Allow-Origin必须设置与请求域一致的域。

但是客户端的cookie也会遵循同源策略，不是上传给非同源的服务器

Access-Control-Expose-Headers（可以被获取的头部

复杂请求

最大不同的是有一个预请求，用于确认：当前域，当前请求方法，当前请求的额外头部信息是否符合服务器的要求。

origin

Access-Control-Request-Method

Access-Control-Request-Headers

7、 WebSocket协议跨域

H5新协议，实现了浏览器与服务双向通信，同时也允许跨域通讯。

使用socket.io方便一点

但是如果直接socket.io写跨域连接，在前端会产生一个跨域的ajax get请求，导致连接失败。所有和CORS一样，也要后端配置跨域。

## 完美方案

8、nginx代理跨域

。。。。。

9、 nodejs中间件代理跨域

[http://dongyuewu.me/pages/cross/node-proxy/proxy.html](http://dongyuewu.me/node-proxy/proxy.html)

大致原理与Nginx相似， 也是在前端域开启一个代理路径，转发到目标服务器上，

实现数据转发（代理时，修改相应的头部，让前后端都不跨域）

在vue的开发环境中可以通过webpack-dev-server开启中间代理

需要注意的是：

cookie和跨域没有关系

cookie是不能跨域名。

cookie不区分端口

cookie不区分协议

cookie并不严格按照path部分提供隔离