什么是跨域请求

概述

在 HTML 中，<a>, <form>, <img>, <script>, <iframe>, <link> 等标签以及 Ajax 都可以指向一个资源地址，而所谓的**跨域请求**就是指：当前发起请求的域与该请求指向的资源所在的域不一样。这里的域指的是这样的一个概念：我们认为若协议 + 域名 + 端口号均相同，那么就是同域。

举个例子：假如一个域名为 aaa.cn 的网站，它发起一个资源路径为 aaa.cn/books/getBookInfo 的 Ajax 请求，那么这个请求是同域的，因为资源路径的协议、域名以及端口号与当前域一致（例子中协议名默认为 http，端口号默认为 80）。但是，如果发起一个资源路径为 bbb.com/pay/purchase 的 Ajax 请求，那么这个请求就是跨域请求，因为域不一致，与此同时由于安全问题，这种请求会受到同源策略限制。

跨域请求的安全问题

通常，浏览器会对上面提到的跨域请求作出限制。浏览器之所以要对跨域请求作出限制，是出于安全方面的考虑，因为跨域请求有可能被不法分子利用来发动 CSRF 攻击。

CSRF 攻击:

CSRF（Cross-site request forgery），中文名称：跨站请求伪造，也被称为：one click attack/session riding，缩写为：CSRF/XSRF。CSRF 攻击者在用户已经登录目标网站之后，诱使用户访问一个攻击页面，利用目标网站对用户的信任，以用户身份在攻击页面对目标网站发起伪造用户操作的请求，达到攻击目的。

CSRF 攻击的原理大致描述如下：有两个网站，其中 A 网站是真实受信任的网站，而 B 网站是危险网站。在用户登陆了受信任的 A 网站是，本地会存储 A 网站相关的 Cookie，并且浏览器也维护这一个 Session 会话。这时，如果用户在没有登出 A 网站的情况下访问危险网站 B，那么危险网站 B 就可以模拟发出一个对 A 网站的请求（跨域请求）对 A 网站进行操作，而在 A 网站的角度来看是并不知道请求是由 B 网站发出来的（Session 和 Cookie 均为 A 网站的），这时便成功发动一次 CSRF 攻击。

因而 CSRF 攻击可以简单理解为：攻击者盗用了你的身份，以你的名义发送而已请求。CSRF 能够做的事情包括：以你名义发送邮件，发消息，盗取你的账

号，甚至于购买商品，虚拟货币转账.....造成的问题包括：个人隐私泄露以及财产安全。

因此，大多数浏览器都会跨域请求作出限制，这是从浏览器层面上的对 CSRF 攻击的一种防御，但是需要注意的是在复杂的网络环境中借助浏览器来防御 CSRF 攻击并不足够，还需要从服务端或者客户端方面入手防御。详细可以参考这篇文章浅谈 CSRF 攻击方式

同源策略(Same-origin Policy)

概述

- 同源策略是 Netscape 提出的一个著名的安全策略
- 同源策略是浏览器最核心最基础的安全策略
- 现在所有的可支持 Javascript 的浏览器都会使用这个策略
- web 构建在同源策略基础之上，浏览器对非同源脚本的限制措施是对同源策略的具体实现

同源策略的含义

- DOM 层面的同源策略：限制了来自不同源的"Document"对象或 JS 脚本，对当前"document"对象的读取或设置某些属性
- Cookie 和 XMLHttprequest 层面的同源策略：禁止 Ajax 直接发起跨域 HTTP 请求（其实可以发送请求，结果被浏览器拦截，不展示），同时 Ajax 请求不能携带与本网站不同源的 Cookie。
- 同源策略的非绝对性：<script><img><iframe><link><video><audio>等带有 src 属性的标签可以从不同的域加载和执行资源。
- 其他插件的同源策略：flash、java applet、silverlight、googlegears 等浏览器加载的第三方插件也有各自的同源策略，只是这些同源策略不属于浏览器原生的同源策略，如果有漏洞则可能被黑客利用，从而留下 XSS 攻击的后患

同源的具体含义

- 域名、协议、端口有一个不同就不是同源，三者均相同，这两个网站才是同源

跨域解决方法

虽然在安全层面上同源限制是必要的，但有时同源策略会对我们的合理用途造成影响，为了避免开发的应用受到限制，有多种方式可以绕开同源策略，下面介绍的是经常使用的 JSONP，CORS 方法。

JSONP

原理：

- JSONP 是一种非官方的跨域数据交互协议
- JSONP 本质上是利用 <script><img><iframe> 等标签不受同源策略限制，可以从不同域加载并执行资源的特性，来实现数据跨域传输。
- JSONP 由两部分组成：回调函数和数据。回调函数是当响应到来时应该在页面中调用的函数，而数据就是传入回调函数中的 JSON 数据。
- JSONP 的理念就是，与服务端约定好一个回调函数名，服务端接收到请求后，将返回一段 Javascript，在这段 Javascript 代码中调用了约定好的回调函数，并且将数据作为参数进行传递。当网页接收到这段 Javascript 代码后，就会执行这个回调函数，这时数据已经成功传输到客户端了。

示例：

首先当前页面中声明有这样的一个函数，它将作为 JSONP 的回调函数处理作为函数参数传入的数据

```
<script type="text/javascript">
    function dosomething(jsondata){
        //处理获得的 json 数据
    }
</script>
```

然后，我们就可以借助 <script><img><iframe> 等标签可以引入不同域资源的特性，将需要发送的请求的路径作为 src 参数，其中需要注意的是：需要告知服务端回调函数的函数名。

```
<script
src="http://example.com/data.php?callback=dosomething"></script>
```

这时服务端在返回数据的时候，就会返回一端 Javascript 代码，在 Javascript 代码中调用了回调函数，并且需要返回的数据作为回调函数的参数

```
dosomething(['a','b','c']);
```

最后页面成功加载了刚才指定路径的资源后，将会执行该 Javascript 代码，dosomething 函数将执行，这时一次跨域请求完成。

另外，如果页面引入了 jQuery，那么可以通过它封装的方法很方便的实现 JSONP 操作了

```
// Using YQL and JSONP
$.ajax({
    url: "http://query.yahooapis.com/v1/public/yql",

    // The name of the callback parameter, as specified by the YQL
service
    jsonp: "callback",

    // Tell jQuery we're expecting JSONP
    dataType: "jsonp",

    // Tell YQL what we want and that we want JSON
    data: {
        q: "select title,abstract,url from search.news where
query=\"cat\"",
        format: "json"
    },

    // Work with the response
    success: function( response ) {
        console.log( response ); // server response
    }
});
```

优缺点：

JSONP 的优点是：它不像 XMLHttpRequest 对象实现的 Ajax 请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行。

JSONP 的缺点是：它只支持 GET 请求，而不支持 POST 请求等其他类型的 HTTP 请求

CORS

介绍

跨源资源共享 Cross-Origin Resource Sharing(CORS) 是一个新的 W3C 标准，它新增的一组 HTTP 首部字段，允许服务端其声明哪些源站有权限访问哪些资源。换言之，它允许浏览器向声明了 CORS 的跨域服务器，发出 XMLHttpReuest 请求，从而克服 Ajax 只能同源使用的限制。

另外，规范也要求对于非简单请求，浏览器必须首先使用 OPTION 方法发起一个预检请求(preflight request)，从而获知服务端是否允许该跨域请求，在服务器确定允许后，才发起实际的 HTTP 请求。对于简单请求、非简单请求以及预检请求的详细资料可以阅读 HTTP 访问控制（CORS） 。

HTTP 协议 Header 简析

下面对 CORS 中新增的 HTTP 首部字段进行简析：

- Access-Control-Allow-Origin

  响应首部中可以携带这个头部表示服务器允许哪些域可以访问该资源，其语法如下：

  Access-Control-Allow-Origin: <origin> | *

  其中，origin 参数的值指定了允许访问该资源的外域 URI。对于不需要携带身份凭证的请求，服务器可以指定该字段的值为通配符，表示允许来自所有域的请求。

- Access-Control-Allow-Methods

  该首部字段用于预检请求的响应，指明实际请求所允许使用的 HTTP 方法。其语法如下：

  Access-Control-Allow-Methods: <method>[, <method>]*

- Access-Control-Allow-Headers

  该首部字段用于预检请求的响应。指明了实际请求中允许携带的首部字段。其语法如下：

  Access-Control-Allow-Headers: <field-name>[, <field-name>]*

- Access-Control-Max-Age

该首部字段用于预检请求的响应，指定了预检请求能够被缓存多久，其语法如下：

Access-Control-Max-Age: <delta-seconds>

- Access-Control-Allow-Credentials

  该字段可选。它的值是一个布尔值，表示是否允许发送 Cookie。默认情况下，Cookie 不包括在 CORS 请求之中。设为 true，即表示服务器明确许可，Cookie 可以包含在请求中，一起发给服务器。其语法如下：

  Access-Control-Allow-Credentials: true

  另外，如果要把 Cookie 发送到服务器，除了服务端要带上 Access-Control-Allow-Credentials 首部字段外，另一方面请求中也要带上 withCredentials 属性。

  但是需要注意的是：如果需要在 Ajax 中设置和获取 Cookie，那么 Access-Control-Allow-Origin 首部字段不能设置为* ，必须设置为具体的 origin 源站。详细可阅读文章 CORS 跨域 Cookie 的设置与获取

- Origin

  该首部字段表明预检请求或实际请求的源站。不管是否为跨域请求，Origin 字段总是被发送。其语法如下：

  Origin: <origin>

- Access-Control-Request-Method

  该首部字段用于预检请求。其作用是，将实际请求所使用的 HTTP 方法告诉服务器。其语法如下：

  Access-Control-Request-Method: <method>

- Access-Control-Request-Headers

  该首部字段用于预检请求。其作用是，将实际请求所携带的首部字段告诉服务器。其语法如下：

  Access-Control-Request-Headers: <field-name>[, <field-name>]*

示例

假设我们在 bbb.cn 域名下，发送一个 Ajax 请求到 aaa.cn 域名，其路径如下：http://aaa.cn/localserver/api/corsTest 。由于同源策略，这样的 Ajax 请求将会被浏览器所拦截，得到下面的信息：

若想能够发送跨域请求，我们只需要在服务器的响应中配置适当的 CORS HTTP 首部字段就可以了，例如可以加入以下的首部字段：

Access-Control-Allow-Methods：*

此时，Ajax 请求就可以顺利的发送和接收了，对应的请求和响应头部如下：

对于在 Java Web 项目中，如何在 Servlet 或这 Spring MVC 中配置 CORS 可以阅读文章 Spring MVC 实现 CORS 跨域 。

与 JSONP 的比较

- JSONP 只能实现 GET 请求，而 CORS 支持所有类型的 HTTP 请求
- 使用 CORS ，开发者可以是使用普通的 XMLHttpRequest 发起请求和获取数据，比起 JSONP 有更好的错误处理
- 虽然绝大多数现代的浏览器都已经支持 CORS，但是 CORS 的兼容性比不上 JSONP，一些比较老的浏览器只支持 JSONP