京东一面：post为什么会发送两次请求?

京东一面：post为什么会发送两次请求？🤪🤪🤪

在前段时间的一次面试中，被问到了一个如标题这样的问题。要想好好地去回答这个问题，这里牵扯到的知识点也是比较多的。

那么接下来这篇文章我们就一点一点开始引出这个问题。

同源策略

在浏览器中，内容是很开放的，任何资源都可以接入其中，如 JavaScript 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。

但也不是说浏览器就是完全自由的，如果不加以控制，就会出现一些不可控的局面，例如会出现一些安全问题，如：

- 跨站脚本攻击（XSS）
- SQL 注入攻击
- OS 命令注入攻击
- HTTP 首部注入攻击
- 跨站点请求伪造（CSRF）
- 等等.....

如果这些都没有限制的话，对于我们用户而言，是相对危险的，因此需要一些安全策略来保障我们的隐私和数据安全。

这就引出了最基础、最核心的安全策略：同源策略。

什么是同源策略

同源策略是一个重要的安全策略，它用于限制一个源的文档或者它加载的脚本如何能与另一个源的资源进行交互。

如果两个 URL 的协议、主机和端口都相同，我们就称这两个 URL 同源。

- 协议：协议是定义了数据如何在计算机内和之间进行交换的规则的系统，例如 HTTP、HTTPS。
- 主机：是已连接到一个计算机网络的一台电子计算机或其他设备。网络主机可以向网络上的用户或其他节点提供信息资源、服务和应用。使用 TCP/IP 协议族参与网络的计算机也可称为 IP 主机。
- 端口：主机是计算机到计算机之间的通信，那么端口就是进程到进程之间的通信。

如下表给出了与 URL `http://store.company.com:80/dir/page.html` 的源进行对比的示例：

URL	结果	原因
http://store.company.com:80/dir2/page.html	同源	只有路径不同
http://store.company.com:80/dir/inner/another.html	同源	只有路径不同
https://store.company.com:443/secure.html	不同源	协议不同，HTTP 和 HTTPS
http://store.company.com:81/dir/etc.html	不同源	端口不同
http://news.company.com:80/dir/other.html	不同源	主机不同

同源策略主要表现在以下三个方面：DOM、Web 数据和网络。

- DOM 访问限制：同源策略限制了网页脚本（如 JavaScript）访问其他源的 DOM。这意味着通过脚本无法直接访问跨源页面的 DOM 元素、属性或方法。这是为了防止恶意网站从其他网站窃取敏感信息。
- Web 数据限制：同源策略也限制了从其他源加载的 Web 数据（例如 XMLHttpRequest 或 Fetch API）。在同源策略下，XMLHttpRequest 或 Fetch 请求只能发送到与当前网页具有相同源的目标。这有助于防止跨站点请求伪造（CSRF）等攻击。
- 网络通信限制：同源策略还限制了跨源的网络通信。浏览器会阻止从一个源发出的请求获取来自其他源的响应。这样做是为了确保只有受信任的源能够与服务器进行通信，以避免恶意行为。

出于安全原因，浏览器限制从脚本内发起的跨源 HTTP 请求，XMLHttpRequest 和 Fetch API，只能从加载应用程序的同一个域请求 HTTP 资源，除非使用 CORS 头文件

CORS

对于浏览器限制这个词，要着重解释一下：不一定是浏览器限制了发起跨站请求，也可能是跨站请求可以正常发起，但是返回结果被浏览器拦截了。

浏览器将不同域的内容隔离在不同的进程中，网络进程负责下载资源并将其送到渲染进程中，但由于跨域限制，某些资源可能被阻止加载到渲染进程。如果浏览器发现一个跨域响应包含了敏感数据，它可能会阻止脚本访问这些数据，即使网络进程已经获得了这些数据。CORB 的目标是在渲染之前尽早阻止恶意代码获取跨域数据。

> CORB 是一种安全机制，用于防止跨域请求恶意访问跨域响应的数据。渲染进程会在 CORB 机制的约束下，选择性地将哪些资源送入渲染进程供页面使用。

例如，一个网页可能通过 AJAX 请求从另一个域的服务器获取数据。虽然某些情况下这样的请求可能会成功，但如果浏览器检测到请求返回的数据可能包含恶意代码或与同源策略冲突，浏览器可能会阻止网页访问返回的数据，以确保用户的安全。

跨源资源共享（Cross-Origin Resource Sharing，CORS）是一种机制，允许在受控的条件下，不同源的网页能够请求和共享资源。由于浏览器的同源策略限制了跨域请求，CORS 提供了一种方式来解决在 Web 应用中进行跨域数据交换的问题。

CORS 的基本思想是，服务器在响应中提供一个标头（HTTP 头），指示哪些源被允许访问资源。浏览器在发起跨域请求时会先发送一个预检请求（OPTIONS 请求）到服务器，服务器通过设置适当的 CORS 标头来指定是否允许跨域请求，并指定允许的请求源、方法、标头等信息。

简单请求

不会触发 CORS 预检请求。这样的请求为 `简单请求` ，。若请求满足所有下述条件，则该请求可视为 `简单请求` ：

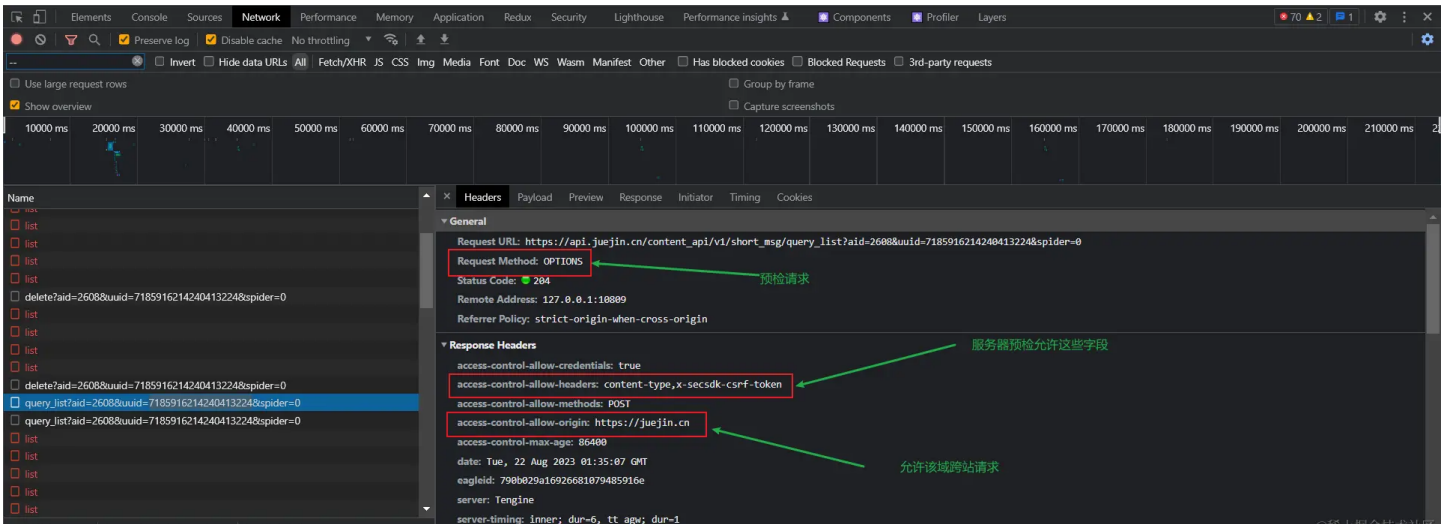
1. HTTP 方法限制：只能使用 GET、HEAD、POST 这三种 HTTP 方法之一。如果请求使用了其他 HTTP 方法，就不再被视为简单请求。
2. 自定义标头限制：请求的 HTTP 标头只能是以下几种常见的标头： `Accept` 、 `Accept-Language` 、 `Content-Language` 、 `Last-Event-ID` 、 `Content-Type` （仅限于 `application/x-www-form-urlencoded` 、 `multipart/form-data` 、 `text/plain` ）。HTML 头部 header field 字段：DPR、Download、Save-Data、Viewport-Width、WIdth。如果请求使用了其他标头，同样不再被视为简单请求。
3. 请求中没有使用 ReadableStream 对象。
4. 不使用自定义请求标头：请求不能包含用户自定义的标头。
5. 请求中的任意 XMLHttpRequestUpload 对象均没有注册任何事件监听器；XMLHttpRequestUpload 对象可以使用 XMLHttpRequest.upload 属性访问

预检请求

非简单请求的 CORS 请求，会在正式通信之前，增加一次 HTTP 查询请求，称为 `预检请求` 。

需预检的请求要求必须首先使用 OPTIONS 方法发起一个预检请求到服务器，以获知服务器是否允许该实际请求。 `预检请求` 的使用，可以避免跨域请求对服务器的用户数据产生未预期的影响。

例如我们在掘金上删除一条沸点:



它首先会发起一个预检请求,预检请求的头信息包括两个特殊字段：

- Access-Control-Request-Method：该字段是必须的，用来列出浏览器的 CORS 请求会用到哪些 HTTP 方法，上例是 POST。
- Access-Control-Request-Headers：该字段是一个逗号分隔的字符串，指定浏览器 CORS 请求会额外发送的头信息字段，上例是 `content-type`，`x-secsdk-csrf-token`。
- access-control-allow-origin：在上述例子中，表示 `https://juejin.cn` 可以请求数据，也可以设置为 `*` 符号，表示统一任意跨源请求。
- access-control-max-age：该字段可选，用来指定本次预检请求的有效期，单位为秒。上面结果中，有效期是 1 天（86408 秒），即允许缓存该条回应 1 天（86408 秒），在此期间，不用发出另一条预检请求。

一旦服务器通过了 `预检请求`，以后每次浏览器正常的 CORS 请求，就都跟简单请求一样，会有一个 Origin 头信息字段。服务器的回应，也都会有一个 Access-Control-Allow-Origin 头信息字段。

上面头信息中，Access-Control-Allow-Origin 字段是每次回应都必定包含的。

附带身份凭证的请求与通配符

在响应附带身份凭证的请求时：

- 为了避免恶意网站滥用 Access-Control-Allow-Origin 头部字段来获取用户敏感信息，服务器在设置时不能将其值设为通配符 `*`。相反，应该将其设置为特定的域，例如：Access-Control-Allow-Origin: `https://juejin.cn`。通过将 Access-Control-Allow-Origin 设置为特定的域，服务器只允许来自指定域的请求进行跨域访问。这样可以限制跨域请求的范围，避免不可信的域获取到用户敏感信息。
- 为了避免潜在的安全风险，服务器不能将 Access-Control-Allow-Headers 的值设为通配符 `*`。这是因为不受限制的请求头可能被滥用。相反，应该将其设置为一个包含标头名称的列表，例如：Access-Control-Allow-Headers: X-PINGOTHER, Content-Type。通过将 Access-Control-Allow-Headers 设置为明确的标头名称列表，服务器可以限制哪些自定义请求头是允许的。只有在允许的标头列表中的头部字段才能在跨域请求中被接受。

- 为了避免潜在的安全风险，服务器不能将 Access-Control-Allow-Methods 的值设为通配符 `*`。这样做将允许来自任意域的请求使用任意的 HTTP 方法，可能导致滥用行为的发生。相反，应该将其设置为一个特定的请求方法名称列表，例如：Access-Control-Allow-Methods: POST, GET。通过将 Access-Control-Allow-Methods 设置为明确的请求方法列表，服务器可以限制哪些方法是允许的。只有在允许的方法列表中的方法才能在跨域请求中被接受和处理。
- 对于附带身份凭证的请求（通常是 Cookie），

这是因为请求的标头中携带了 Cookie 信息，如果 Access-Control-Allow-Origin 的值为 `*`，请求将会失败。而将 Access-Control-Allow-Origin 的值设置为 `https://juejin。cn`，则请求将成功执行。

另外，响应标头中也携带了 Set-Cookie 字段，尝试对 Cookie 进行修改。如果操作失败，将会抛出异常。

为什么本地使用 webpack 进行 dev 开发时，不需要服务器端配置 cors 的情况下访问到线上接口?

当你在本地通过 Ajax 或其他方式请求线上接口时，由于浏览器的同源策略，会出现跨域的问题。但是在服务器端并不会出现这个问题。

它是通过 Webpack Dev Server 来实现这个功能。当你在浏览器中发送请求时，请求会先被 Webpack Dev Server 捕获，然后根据你的代理规则将请求转发到目标服务器，目标服务器返回的数据再经由 Webpack Dev Server 转发回浏览器。这样就绕过了浏览器的同源策略限制，使你能够在本地开发环境中访问线上接口。

参考文章

- CORS 简单请求+预检请求（彻底理解跨域）
- 跨域资源共享 CORS 详解
- 跨源资源共享（CORS）

总结

预检请求是在进行跨域资源共享 `CORS` 时，由浏览器自动发起的一种 OPTIONS 请求。它的存在是为了保障安全，并允许服务器决定是否允许跨域请求。

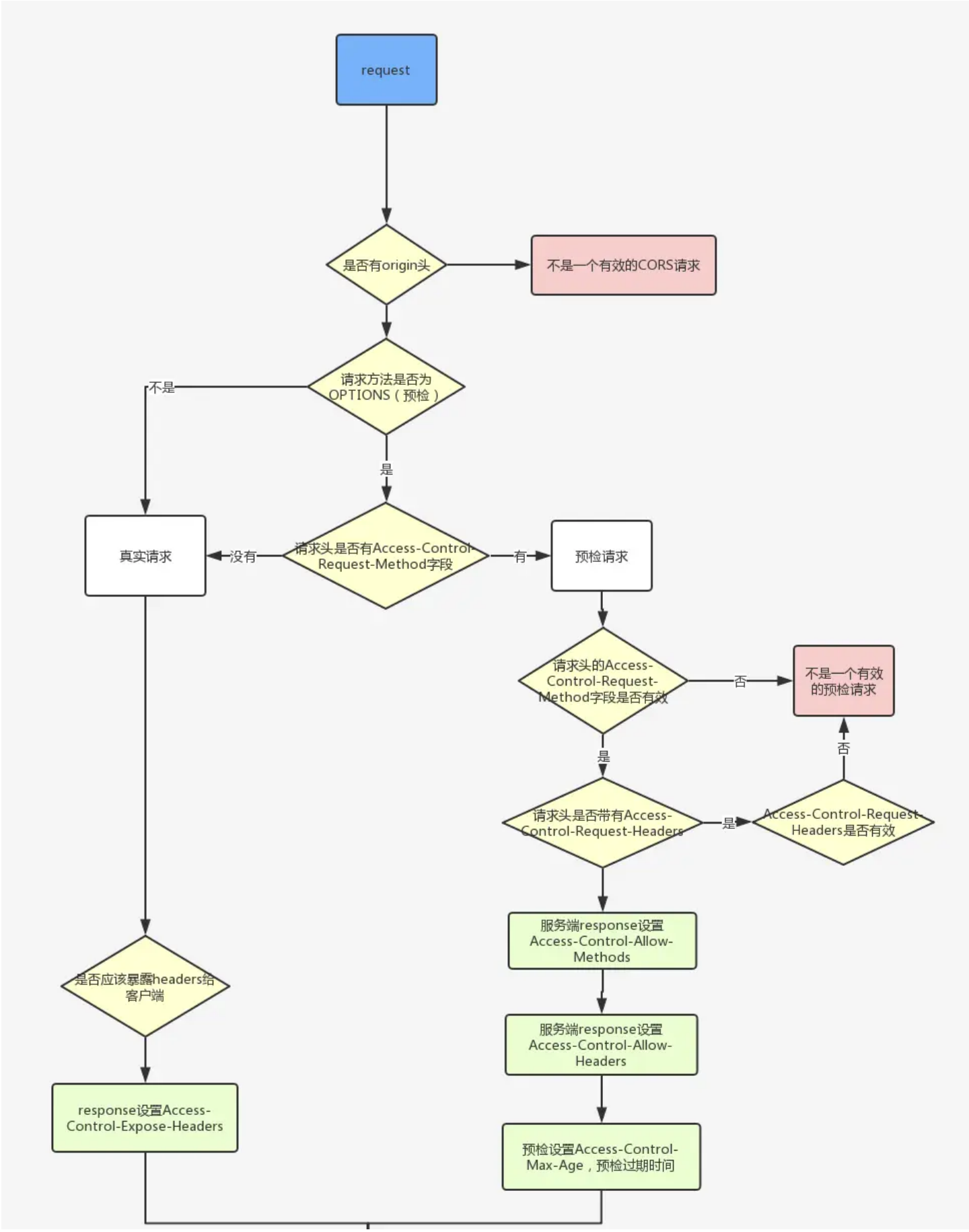
跨域请求是指在浏览器中向不同域名、不同端口或不同协议的资源发送请求。出于安全原因，浏览器默认禁止跨域请求，只允许同源策略。而当网页需要进行跨域请求时，浏览器会自动发送一个预检请求，以确定是否服务器允许实际的跨域请求。

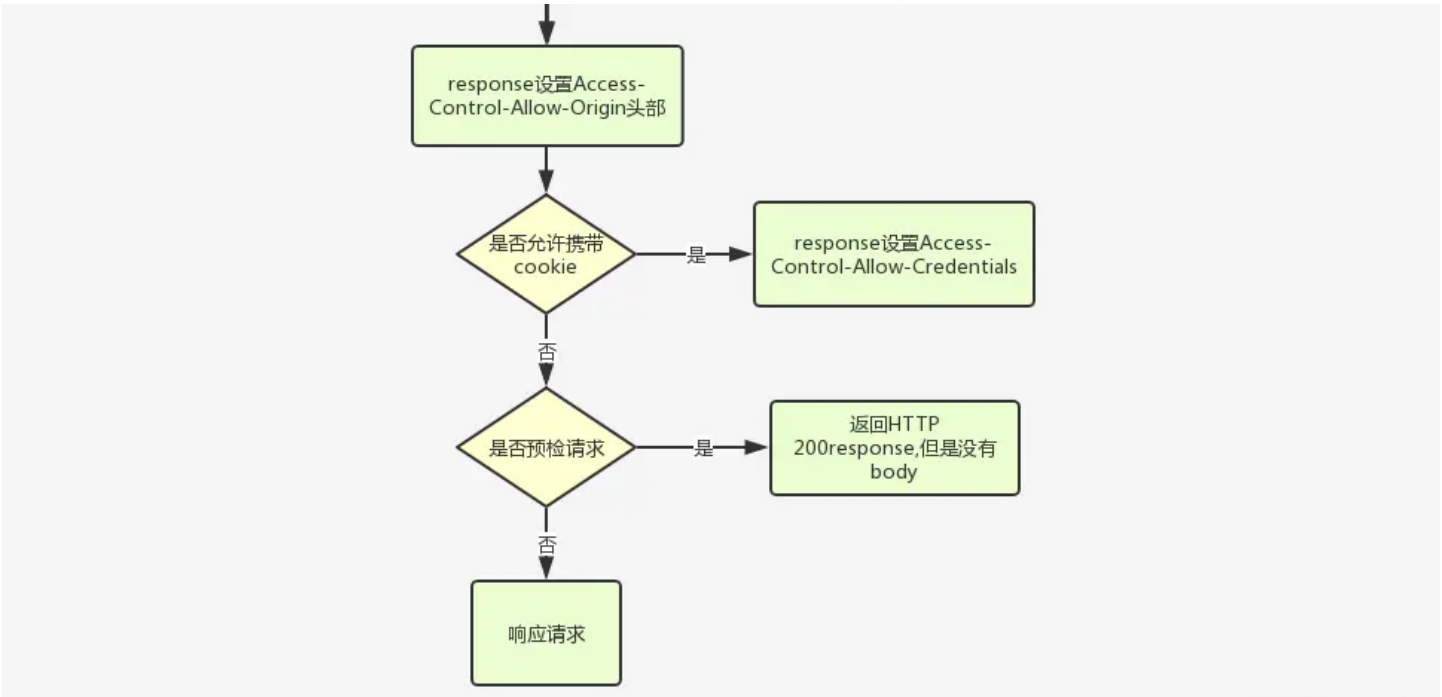
预检请求中包含了一些额外的头部信息，如 Origin 和 Access-Control-Request-Method 等，用于告知服务器实际请求的方法和来源。服务器收到预检请求后，可以根据这些头部信息，进行验证和授权判

断。如果服务器认可该跨域请求，将返回一个包含 Access-Control-Allow-Origin 等头部信息的响应，浏览器才会继续发送实际的跨域请求。

使用预检请求机制可以有效地防范跨域请求带来的安全风险，保护用户数据和隐私。

整个完整的请求流程有如下图所示：





最后分享两个我的两个开源项目,它们分别是: