**摘要：** CORS，一个看似有点“冷门”的领域，但在日常开发中实际上经常碰到这方面的问题。纵观各大技术论坛，却没有几个帖子能讲透CORS的那点事，本期云享团邀请嘉宾深扒了CORS的中外家史，为你打通跨域请求的“任督二脉”。

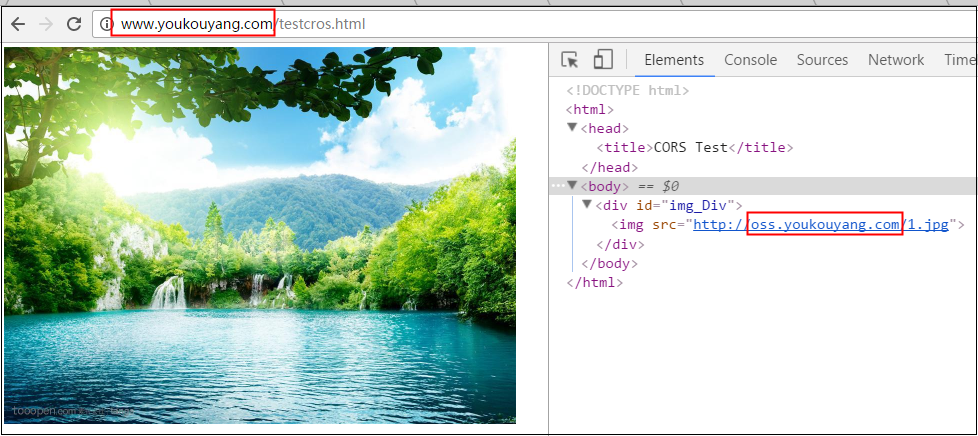
【本期嘉宾介绍】睿得，具有多年研发、运维、安全等IT相关从业经历。目前从事CDN、存储、视频直播点播的技术支持。喜爱钻研，喜爱编码，喜爱分享。



在日常的项目开发时会不可避免的需要进行跨域操作，而在实际进行跨域请求时，经常会遇到类似 No 'Access-Control-Allow-Origin' header is present on the requested resource.这样的报错。  
  
这样的错误，一般是由于CORS跨域验证机制设置不正确导致的，本文将详细讲解CORS跨域验证机制的原理，让您轻松掌握CORS跨域设置的使用方法，安全、方便的进行前端开发。

# 什么是CORS

CORS（Cross-Origin Resource Sharing 跨源资源共享），当一个请求url的协议、域名、端口三者之间任意一与当前页面地址不同即为跨域。

例如最常见的，在一个域名下的网页中，调用另一个域名中的资源。  


相对于上面这种静态的调用方式，还可以通过Ajax技术来**动态发起跨域请求**。例如如下的方式，利用XMLHttpRequest对象发送一个GET请求，获取另一个域名下的图片内容。

<!DOCTYPE html> <html>

<head>CORS Test</head>

<body>

<div id="img\_Div"></div>

<script type="text/javascript">

//XmlHttpRequest对象

function createXmlHttpRequest(){

if(window.ActiveXObject){ //如果是IE浏览器

return new ActiveXObject("Microsoft.XMLHTTP");

}else if(window.XMLHttpRequest){ //非IE浏览器

return new XMLHttpRequest();

}

}

function getFile() {

var img\_Container = document.getElementById("img\_Div");

var xhr = createXmlHttpRequest();

xhr.open('GET', 'http://oss.youkouyang.com/1.jpg', true);

xhr.setRequestHeader('Content-Type', 'image/jpeg');

xhr.responseType = "blob";

xhr.onload = function() {

if (this.status == 200) {

var blob = this.response;

var img = document.createElement("img");

img.onload = function(e) {

window.URL.revokeObjectURL(img.src);

};

img.src = window.URL.createObjectURL(blob);

img\_Container.appendChild(img);

}

}

xhr.send(null);

}

</script>

<div class="row">

<input type="button" onclick="getFile()" value="Get" />

</div>

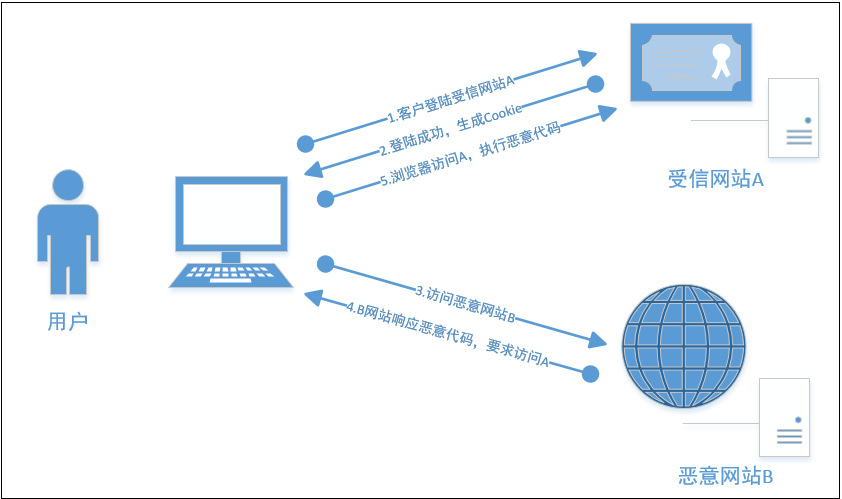
</body></html>

# CORS的作用

为了改善网络应用程序，开发人员要求浏览器供应商允许跨域请求。跨域请求主要用于：

* 调用XMLHttpRequest或fetchAPI通过跨站点方式访问资源
* 网络字体，例如Bootstrap（通过CSS使用@font-face 跨域调用字体）
* 通过canvas标签，绘制图表和视频。

# CORS的安全隐患

跨域请求和Ajax技术都会极大地提高页面的体验，但同时也会带来安全的隐患，其中最主要的隐患来自于CSRF（Cross-site request forgery）跨站请求伪造。  


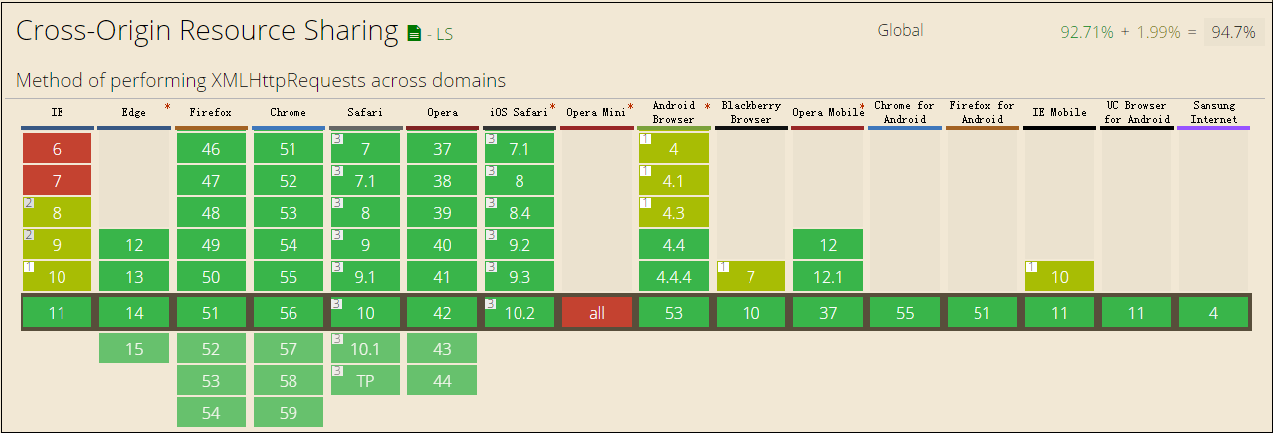
CSRF攻击的大致原理是：

1. 用户通过浏览器，访问正常网站A（例如某银行），通过用户的身份认证（比如用户名/密码）成功A网站。
2. 网站A产生Cookie信息并返回给用户的浏览器；
3. 用户保持A网站页面登录状态，在同一浏览器中，打开一个新的TAB页访问恶意网站B；
4. 网站B接收到用户请求后，返回一些攻击性代码，请求A网站的资源（例如转账请求）；
5. 浏览器执行恶意代码，在用户不知情的情况下携带Cookie信息，向网站A发出请求。
6. 网站A根据用户的Cookie信息核实用户身份（此时用户在A网站是已登录状态），A网站会处理该请求，导致来自网站B的恶意请求被执行。

# CORS验证机制

出于安全原因，浏览器限制从脚本中发起的跨域HTTP请求。默认的安全限制为同源策略， 即JavaScript或Cookie只能访问同域下的内容。  
W3C推荐了一种跨域的访问验证的机制，即CORS（Cross-Origin Resource Sharing 跨源资源共享）。  
这种机制让Web应用服务器能支持跨站访问控制，使跨站数据传输更加安全，减轻跨域HTTP请求的风险。  
CORS验证机制需要客户端和服务端协同处理。

## CORS浏览器支持情况

目前主流浏览器都已基本提供对跨域资源共享的支持，移动端浏览器也几乎全部支持。  


## 客户端处理机制

基于上述的CSRF的风险，各主流的浏览器都会对动态的跨域请求进行特殊的验证处理。验证处理分为简单请求验证处理和预先请求验证处理。

### 简单请求

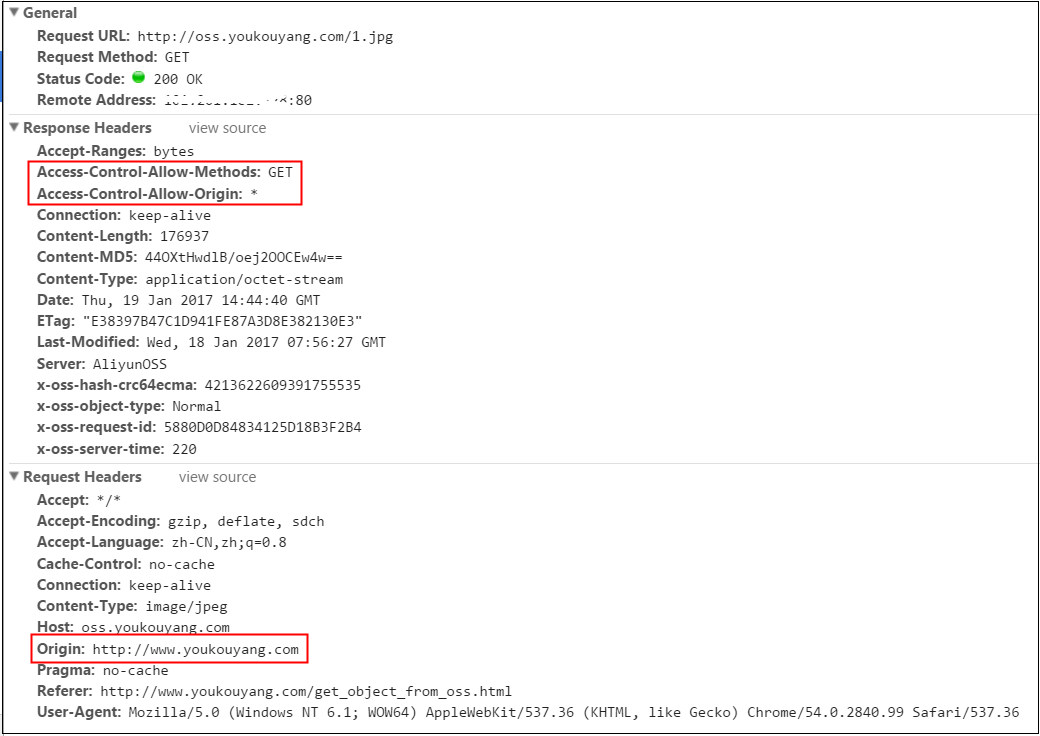
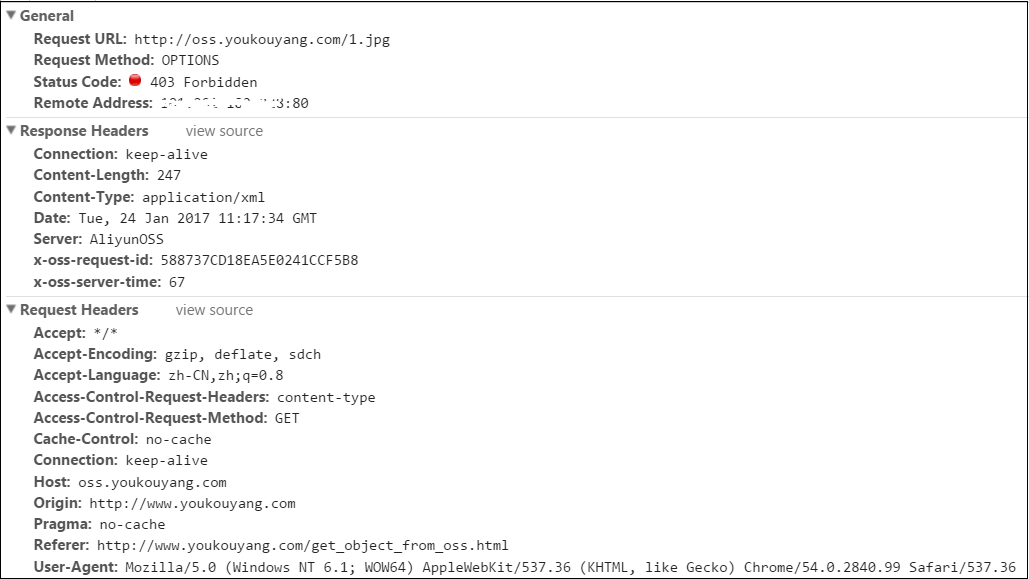
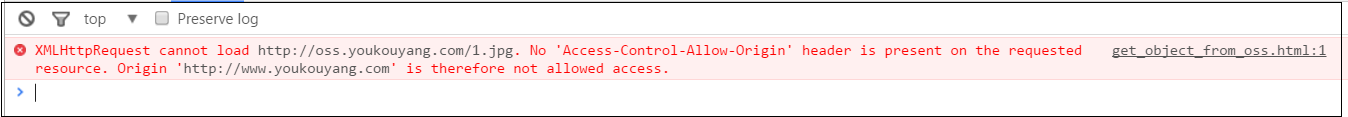
当请求**同时满足**下面两个条件时，浏览器会直接发送GET请求，在同一个请求中做跨域权限的验证。

请求方法是下列之一：

* GET
* HEAD
* POST

请求头中的Content-Type请求头的值是下列之一：

* application/x-www-form-urlencoded
* multipart/form-data
* text/plain

简单请求时，浏览器会直接发送跨域请求，并在请求头中携带Origin 的header，表明这是一个跨域的请求。  
服务器端接到请求后，会根据自己的跨域规则，通过Access-Control-Allow-Origin和Access-Control-Allow-Methods响应头，来返回验证结果。  
如果验证成功，则会直接返回访问的资源内容。  
  
  
如果验证失败，则返回403的状态码，不会返回跨域请求的资源内容。  
  
  
可以通过浏览器的Console查看具体的验证失败原因  


### 预先请求

当请求满足下面**任意一个**条件时，浏览器会先发送一个OPTION请求，用来与目标域名服务器协商决定是否可以发送实际的跨域请求。

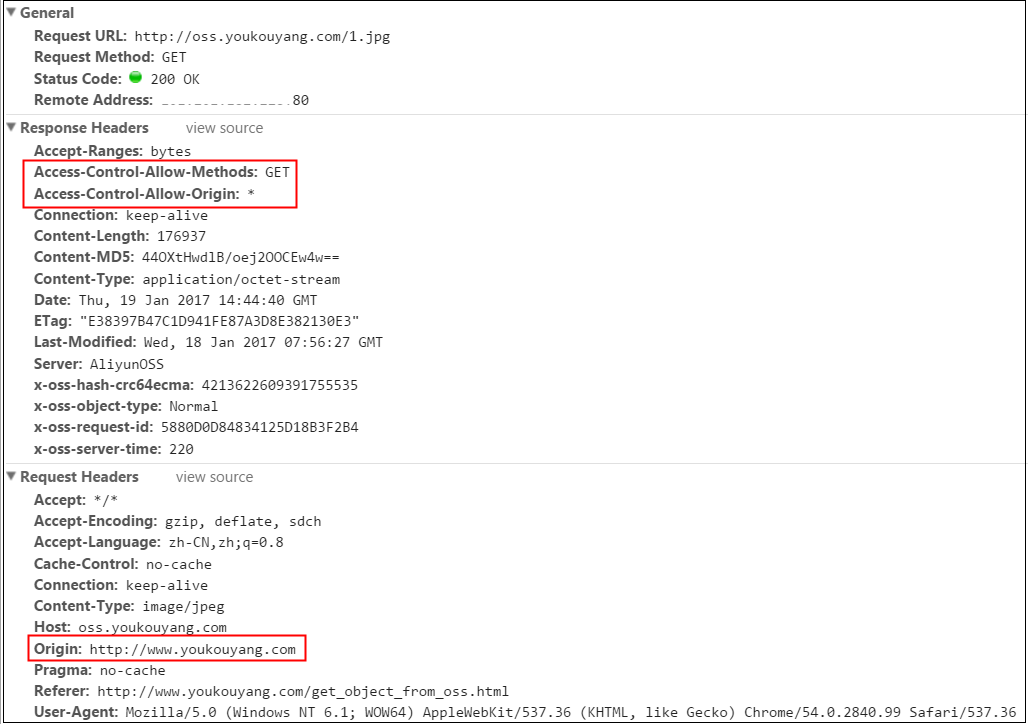
请求方法**不是**下列之一：

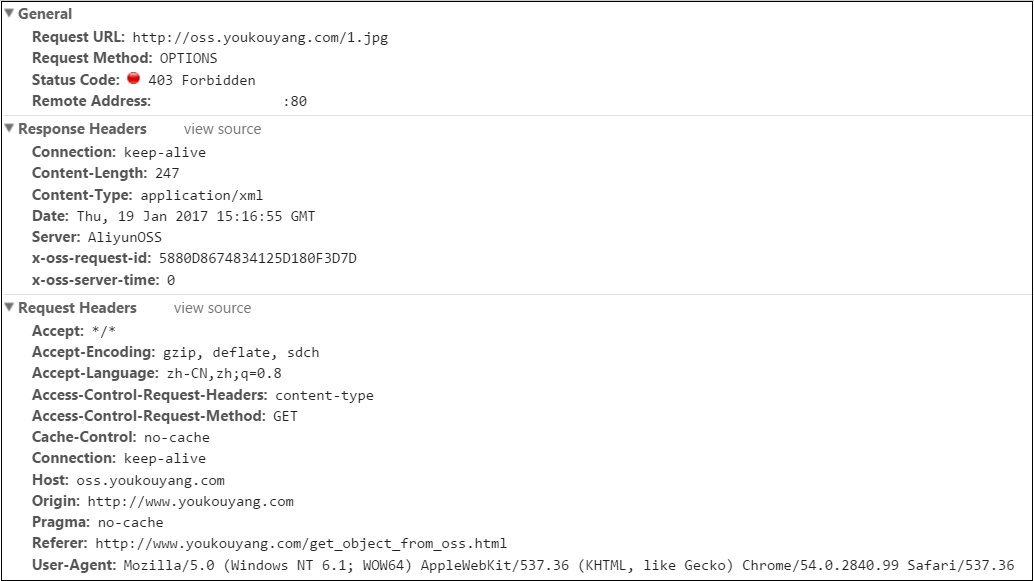
* GET
* HEAD
* POST

请求头中的Content-Type请求头的值**不是**下列之一：

* application/x-www-form-urlencoded
* multipart/form-data
* text/plain

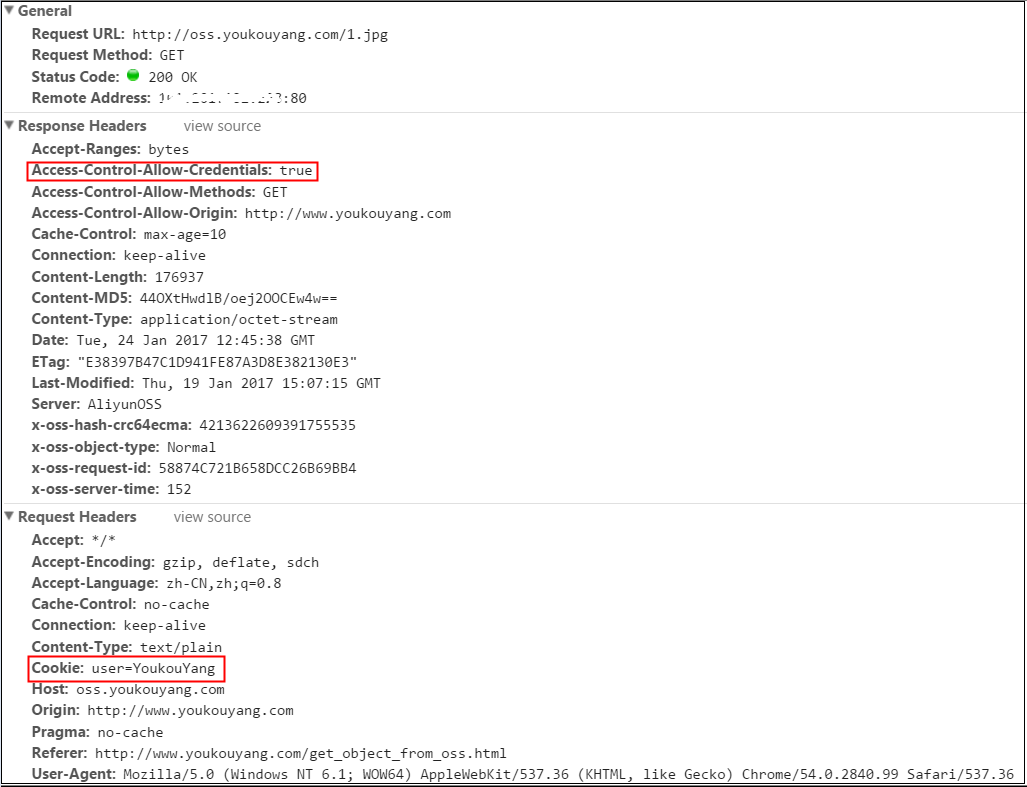
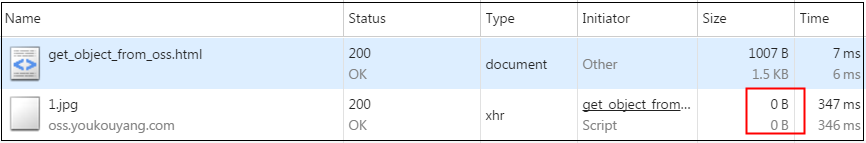
浏览器在发现页面中有上述条件的动态跨域请求的时候，并不会立即执行对应的请求代码，而是会先发送Preflighted requests（预先验证请求），Preflighted requests是一个OPTION请求，用于询问要被跨域访问的服务器，是否允许当前域名下的页面发送跨域的请求。 

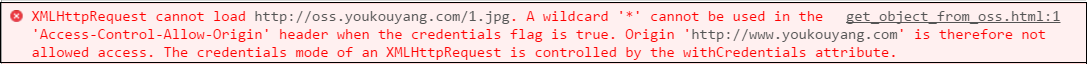
OPTIONS请求头部中会包含以下头部：Origin、Access-Control-Request-Method、Access-Control-Request-Headers。  
服务器收到OPTIONS请求后，设置Access-Control-Allow-Origin、Access-Control-Allow-Method、Access-Control-Allow-Headers头部与浏览器沟通来判断是否允许这个请求。  
如果Preflighted requests验证通过，浏览器才会发送真正的跨域请求。  


如果Preflighted requests验证失败，则会返回403状态，浏览器不会发送真正的跨域请求。  
 

 可以通过浏览器的Console查看具体的验证失败原因  


### 带认证的请求

默认情况下，跨源请求不提供凭据(cookie、HTTP认证及客户端SSL证明等)。通过将withCredentials属性设置为true，可以指定某个请求应该发送凭据。  
xhr.withCredentials = true;  
如果服务器接收带凭据的请求，会用下面的HTTP头部来响应。  
Access-Control-Allow-Credentials: true  
服务器还可以在Preflight响应中发送这个HTTP头部，表示允许源发送带凭据的请求。  
  
  
如果发送的是带凭据的请求，但服务器的响应中没有包含这个头，那么浏览器就不会把响应交给JavaScript(responseText中将是空字符串，size为0)。  


注意，当withCredentials属性设置为true，需要response header中的'Access-Control-Allow-Origin'为一个确定的域名，而不能使用'\*'这样的通配符。  


## 服务端处理机制

服务器端对于跨域请求的处理流程如下：

1. 首先查看http头部有无origin字段；
2. 如果没有，或者不允许，直接当成普通请求处理，结束；
3. 如果有并且是允许的，那么再看是否是preflight(method=OPTIONS)；
4. 如果不是preflight（简单请求），就返回Allow-Origin、Allow-Credentials等，并返回正常内容。
5. 如果是preflight（预先请求），就返回Allow-Headers、Allow-Methods等，内容为空；

# HTTP Header

## Request header

### Origin

Origin头在跨域请求或预先请求中，标明发起跨域请求的源域名。

### Access-Control-Request-Method

Access-Control-Request-Method头用于表明跨域请求使用的实际HTTP方法

### Access-Control-Request-Headers

Access-Control-Request-Headers用于在预先请求时，告知服务器要发起的跨域请求中会携带的请求头信息

## Response header

### Access-Control-Allow-Origin

Access-Control-Allow-Origin头中携带了服务器端验证后的允许的跨域请求域名，可以是一个具体的域名或是一个\*（表示任意域名）。简单请求时，浏览器会根据此响应头的内容决定是否给脚本返回相应内容，预先验证请求时，浏览器会根据此响应头决定是否发送实际的跨域请求。

### Access-Control-Expose-Headers

Access-Control-Expose-Headers头用于允许返回给跨域请求的响应头列表，在列表中的响应头的内容，才可以被浏览器访问。

### Access-Control-Max-Age

Access-Control-Max-Age用于告知浏览器可以将预先检查请求返回结果缓存的时间，在缓存有效期内，浏览器会使用缓存的预先检查结果判断是否发送跨域请求。

### Access-Control-Allow-Credentials

Access-Control-Allow-Credentials用于告知浏览器当withCredentials属性设置为true时，是否可以显示跨域请求返回的内容。简单请求时，浏览器会根据此响应头决定是否显示响应的内容。预先验证请求时，浏览器会根据此响应头决定在发送实际跨域请求时，是否携带认证信息。

### Access-Control-Allow-Methods

Access-Control-Allow-Methods用于告知浏览器可以在实际发送跨域请求时，可以支持的请求方法，可以是一个具体的方法列表或是一个\*（表示任意方法）。简单请求时，浏览器会根据此响应头的内容决定是否给脚本返回相应内容，预先验证请求时，浏览器会根据此响应头决定是否发送实际的跨域请求。

### Access-Control-Allow-Headers

Access-Control-Allow-Headers用于告知浏览器可以在实际发送跨域请求时，可以支持的请求头，可以是一个具体的请求头列表或是一个\*（表示任意请求头）。简单请求时，浏览器会根据此响应头的内容决定是否给脚本返回相应内容，预先验证请求时，浏览器会根据此响应头决定是否发送实际的跨域请求。

# 配置CORS规则

## nginx上的CORS配置 IMG_271

## OSS上的CORS配置 IMG_272

## CDN上的CORS配置 IMG_273

注意：由于CDN的缓存特性，CDN配合OSS时，需要在CDN中设置CORS配置。