CORS（Cross-Origin Resource Sharing，跨源资源共享）

出于安全性，浏览器限制脚本内发起的跨源HTTP请求，这也称之为同源安全策略，基于该策略，浏览器默认阻止“跨域”获取资源。

CORS是一种基于HTTP头的机制，
该机制通过允许服务器标示除了它自己以外的其它 origin，使得浏览器允许这些 origin 访问加载自己的资源。

服务器在标示时， 主要涉及：

- 跨源请求允许的origin （相关头部有 `origin`, `Access-Control-Allow-Origin`）
- 跨源请求的方法 (相关头部有 `Access-Control-Request-Method`, `Access-Control-Allow-Methods`)
- 跨源请求允许带的headers (相关头部有 `Access-Control-Request-Headers`, `Access-Control-Allow-Headers`)
- 跨源请求是否允许带cookie (相关头部有 `cookie`, `Access-Control-Allow-Credentials`)
- 脚本处理跨源请求时能获取到的首部 (相关头部有 `Access-Control-Expose-Headers`)

预检请求（preflight request）

规范要求，对那些可能对服务器数据产生副作用的 HTTP 请求方法（特别是 GET 以外的 HTTP 请求），
浏览器必须首先使用 OPTIONS 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源请求。
服务器确认允许之后，才发起实际的 HTTP 请求。

对于预检请求，需要正确处理相关头（对于相应的行为标示允许或者不允许）并返回204。

nginx中对OPTIONS请求返回204：

```
if ( $request_method = OPTIONS ) {
 return 204;
}
```

简单请求

某些请求不会触发预检，称这样的请求为简单请求。

关于cookie

简单请求不带cookie， 带了cookie就不再是简单请求。

请求后端接口时，一般都需要cookie传递鉴权信息和其它会话信息，所以cookie是必须的。

请求前端完全无状态的静态资源时，cookie则是冗余的。

（假设前端的资源都是静态的、无状态的，无副作用的。）

对于axios，跨源时默认不带cookie，若要带上cookie，需要在配置里显示添加上 `withCredentials: true`：

```
axios.get('/xxx', { withCredentials: true });
```

对于Fetch API，跨源时默认不带cookie，若要带上cookie，需要在配置里显示添加上 `credentials: 'include'`：

```
fetch('/xxx', { credentials: 'include' });
```

带了cookie就不再是简单请求。

若请求中带有cookie，服务器标示允许访问的origin时不能用通配符 `*` 表示允许所有origin跨源访问。

为了简单和最小权限原则，对于后端接口的跨源访问应允许使用cookie，同时对前端无状态的静态资源的跨源访问应不使用cookie。

标示cookie

对前端资源的请求是简单请求，不需要配置`Access-Control-Allow-Credentials`。

对后端接口的请求，因带有cookie，会先有一个OPTION预检请求。

服务器需要在后端的接口的响应中设置`Access-Control-Allow-Credentials`为`true`，以允许请求这些接口时带上cookie。

在配置nginx时，可对后端接口这样配置：

```
add_header Access-Control-Allow-Credentials true always;
```

关于always：

nginx默认会对成功的请求使用add_header指令，但有时鉴权失败时后端服务器会返回401，这时是不会使用add_header指令的，
为了这时也能添加头部成功，需要在后面添加一个`always`。

标示origin

浏览器跨源访问一个资源时，响应需要包含`Access-Control-Allow-Origin`这个头部(该头部的值是用英文逗号分隔的origin列表), 并在值中包含当前的origin。

比如，若在https://www.baidu.com 下向 https://www.google.com 发送一个http请求，则该http请求能跨源成功需要google在它的响应的头中设置：
`Access-Control-Allow-Origin: https://www.baidu.com` 。

在响应附带身份凭证（即有cookie，Authorization）时的请求时，服务器不能将 Access-Control-Allow-Origin 的值设为通配符“*”，而应将其设置为特定的源。

对于后端接口来说，身份凭证总是必须的。

然后对我们的情况来说，当开发环境允许跨源时，总是想允许所有的源的，且没办法将允许的源指定为特定的源（因为不知道大家开发环境的ip和port）。

在http请求中，有一个叫origin的header，在配置nginx时， 可以用变量 `$http_origin` 获取到该值，可以用该值达到类似于通配符 `*` 的效果。

```
add_header Access-Control-Allow-Origin $http_origin always;
```

对于前端无状态的静态资源请求是简单请求，对于允许访问的origin的标示，可简单配置作：

```
add_header Access-Control-Allow-Origin *;
```

标示method

只有预检请求时才会向服务器发送`Access-Control-Request-Method`询问是否允许使用该方法发起跨源请求。

对前端静态资源的跨源请求是简单请求。简单请求没有额外的预检请求，所以对于前端资源不需要标示允许访问的method。

对后端接口的跨源请求都不是简单请求，都会有一个额外的预检请求，预检请求中，需要返回一个`Access-Control-Allow-Methods`，
里面包含`Access-Control-Request-Method`对应的方法即可。

对于后端接口标示允许的method:

```
add_header Access-Control-Allow-Methods "GET, POST, PUT, DELETE, OPTIONS" always;
```

后端接口是带有鉴权信息的，这时`Access-Control-Allow-Methods`的值不能简单地用通配符`*`表示所有。

标示header

部分简单响应首部是无需标示的。

对于header的标示有两部分：跨源请求时允许**额外**带的header，以及请求成功后，可以暴露给脚本的header。

对前端静态资源的跨源请求是无副作用的，不需要带额外的header，也没有需要暴露给脚本的header，服务器对此无需要标示。

对后端接口的跨源请求因都带有cookie，需要先进行预检请求。

请求头 `Access-Control-Request-Headers` 出现于 preflight request（预检请求）中，用于通知服务器在真正的请求中会采用哪些请求头。

服务器若同意接下来的请求，则在预检请求的响应中用 `Access-Control-Allow-Headers` 列出这些允许的首部。

在跨源访问时，默认情况下只有七个简单响应首部
Cache-Control、Content-Language、Content-Length、Content-Type、Expires、Last-Modified、Pragma暴露给外部。

如果想要让客户端可以访问到其他的首部信息，则需要服务器将它们在 Access-Control-Expose-Headers 里面列出来。

`Access-Control-Expose-Headers` 头让服务器把允许浏览器访问的首部放入白名单。

后端接口是带有鉴权信息的，这时`Access-Control-Allow-Headers`和`Access-Control-Expose-Headers`的值不能简单地用通配符`*`表示所有。

对于允许传递的首部，服务器可以"浏览器请求传啥就同意传啥"，以达到类似于通配符`*`的效果。

在nginx中，可以用`$http_access_control_request_headers`获取到预检请求中通知的会传递的首部。

同时，定义了一个`Access-Control-Eager-Headers`的首部，用于向资源方请示跨源访问时希望暴露给外部的其它响应首部。
在nginx中，可以用`$http_access_control_eager_headers`来获取`Access-Control-Eager-Headers`首部的值。

【注意】`Access-Control-Eager-Headers`是一个自定义的扩展首部。

nginx配置如下：

```
add_header Access-Control-Allow-Headers     $http_access_control_request_headers
always;
add_header Access-Control-Expose-Headers    $http_access_control_eager_headers always;
```

暴露额外首部的示例

以某个文件下载接口为例，它需要访问响应中的`content-disposition`这个首部，那么需要在请求时，显示地请求服务器允许暴露这个header：

```
axios.post('/xxx', /*payload*/{}, {
  withCredentials: true,
  responseType: 'blob'
  headers: {
    // 大小写均可
    'Access-Control-Eager-Headers': 'content-disposition'
  }
});
```

浏览器发送时，会先发送一个预检请求：

```
OPTION /xxx HTTP/1.1
origin xx.xx.xx
access-control-allow-headers  access-control-eager-headers,content-type
access-control-request-method  POST
```

然后服务器返回响应：

```
HTTP/1.1 204 No Content
access-control-allow-credentials true
access-control-allow-headers access-control-eager-headers,content-type
access-control-allow-methods GET, POST, PUT, DELETE, OPTIONS
access-control-allow-origin xx.xx.xx
```

预栓请求成功，浏览器开始正式发送请求：

```
POST /xxx HTTP/1.1
origin xx.xx.xx
content-type application/json
access-control-eager-headers content-disposition
cookie xxxx
```

然后在服务器把请求头中的`Access-Control-Eager-Headers`的值，用响应头`Access-Control-Expose-Headers`返回回来：

```
HTTP/1.1 200 Success
access-control-allow-credentials true
access-control-allow-methods GET, POST, PUT, DELETE, OPTIONS
access-control-allow-origin xx.xx.xx
access-control-expose-headers content-disposition
content-disposition xxxxx
```

这样就成功地访问了响应中的content-disposition首部。

nginx的完整CORS配置

（这里遵循跨源请求前端不带cookie，跨源请求后端均带cookie的策略。）

```
# 后端接口
location ~  ^/api/ {
  # always保证在非成功请求，如401时也能正常跨源
  # 这里用 $http_origin 实现了类似 * 的效果，  但这里 $http_origin 不能替换成 *
  add_header Access-Control-Allow-Origin      $http_origin always;
  # 允许跨源请求的方法列表
```

```
  add_header Access-Control-Allow-Methods     "GET, POST, PUT, DELETE, OPTIONS" always;
  # 允许跨源请求时额外带上的头
  add_header Access-Control-Allow-Headers     $http_access_control_request_headers
always;
  # 跨源请求时允许客户端访问到的更多首部
  add_header Access-Control-Expose-Headers    $http_access_control_eager_headers
always;
  # 对预检请求返回204（，并设置上述响应头）
  if ( $request_method = OPTIONS ) {
    return 204;
  }

  # 其它部分
  # ...
}

# 前端静态资源
location ~ ^/[^?]+\. {
  # 仅配置了源， 这里 $http_origin 也可以替换作 *
  add_header Access-Control-Allow-Origin $http_origin;
  # 对前端静态资源的请求是简单请求，无OPTION预检请求

  # 其它部分
  # ...
}
```