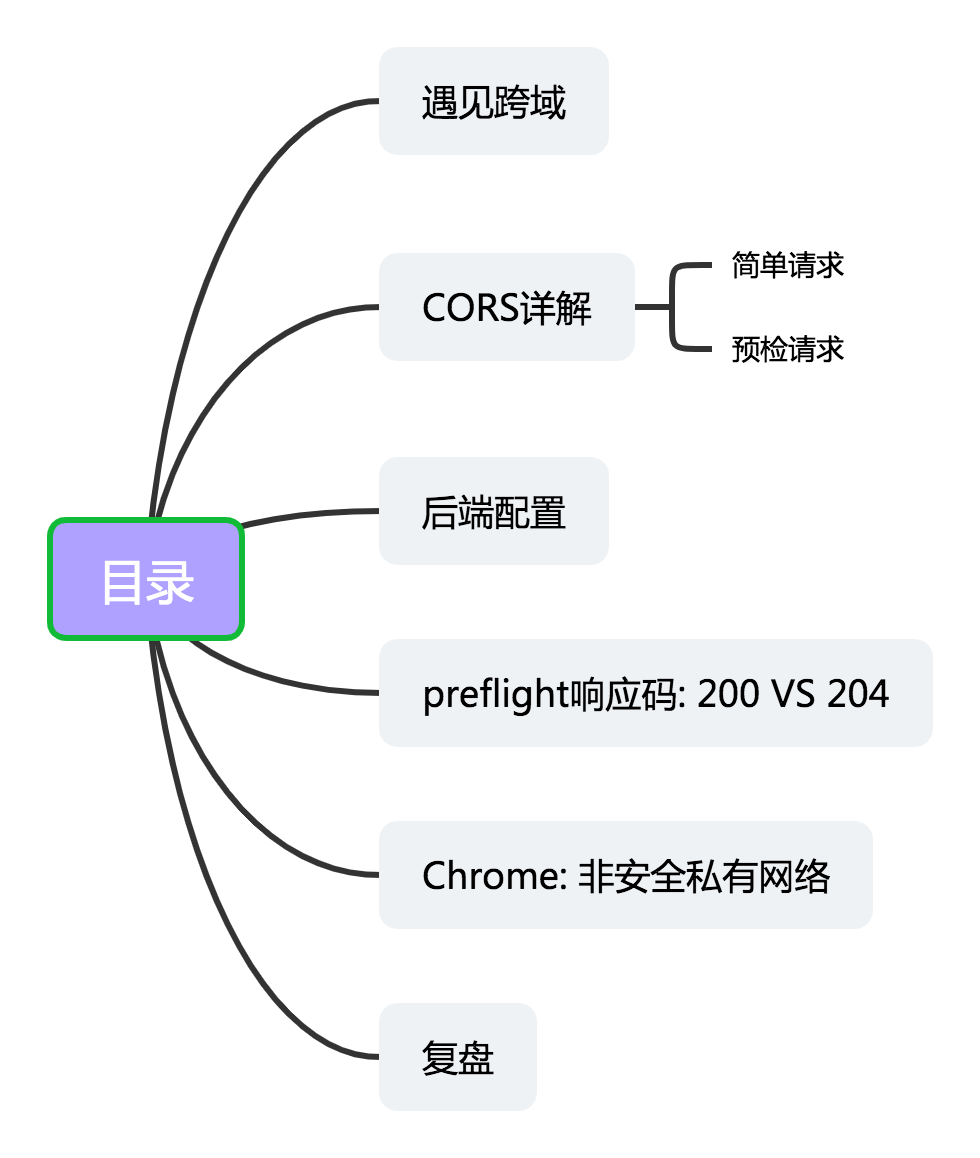
**后端工程师的「跨域」之旅**

<https://mp.weixin.qq.com/s/ckOlQDWcX9HvTWOy1-FcFw>

**跨域**，对后端工程师来说，可谓既熟悉又陌生。

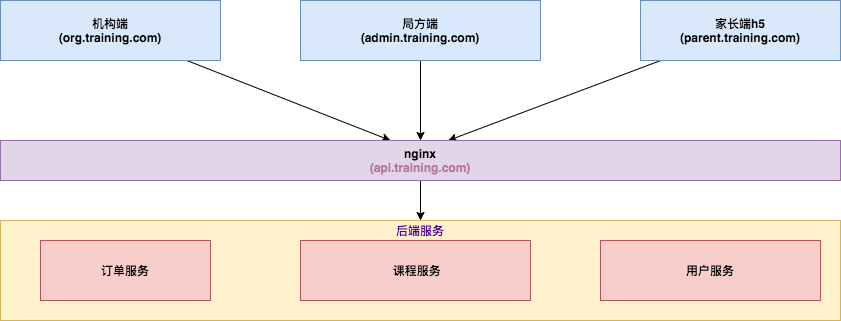
这两个月我以架构师的角色参与一款教育产品的孵化，有了一段难忘的**跨域之旅**。

写这篇文章，我想分享我在跨域这个知识点上的经历和思考，希望对大家有所启发。



**1 遇见跨域**

产品有多端：机构端，局方端 ，家长端等 。每端都有独立的域名，有的是在PC上访问，有的是通过微信公众号来访问，有的是扫码后H5展现。



接入层调用的接口域名统一使用 api.training.com这个独立的域名，通过Nginx来配置请求转发。

通常，我们提到的跨域指：**CORS**。

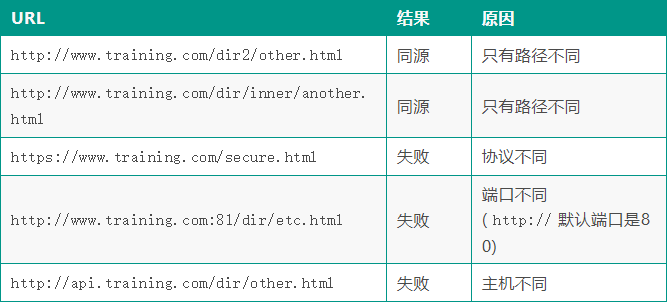
**CORS**是一个W3C标准，全称是"跨域资源共享"（Cross-origin  resource  sharing）, 它需要浏览器和服务器同时支持他，允许浏览器向跨源服务器发送XMLHttpRequest请求，从而克服 AJAX 只能**同源**使用的限制。

那么如何定义同源呢？我们先看下一个典型的网站的地址：



**同源**是指：**协议、域名、端口号完全相同**。

下表给出了与 URL http://www.training.com/dir/page.html 的源进行对比的示例:



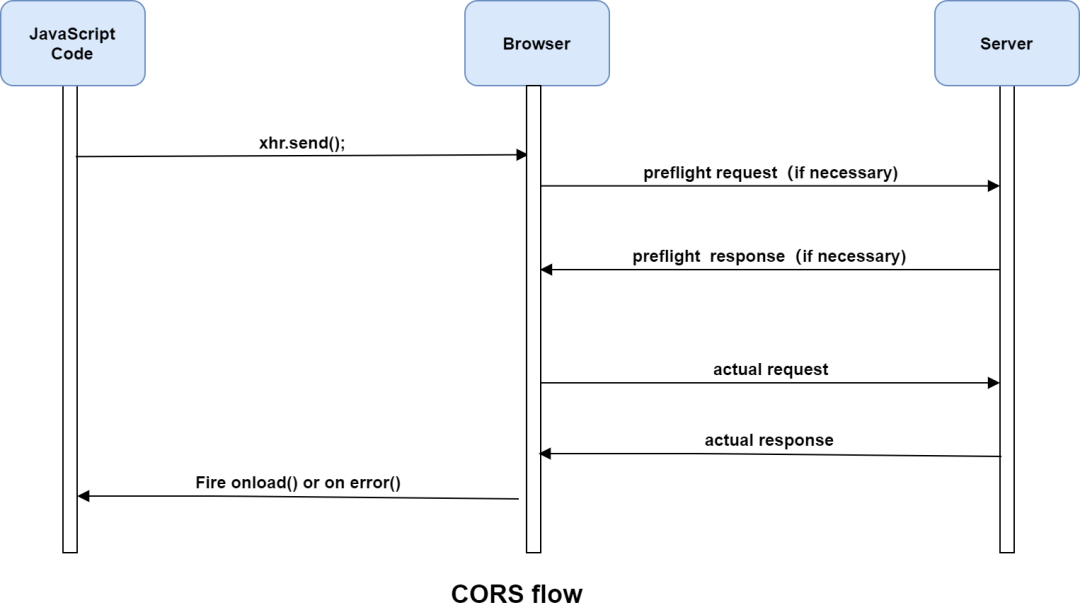
当用户通过浏览器访问应用（http://admin.training.com）时，调用接口的域名非同源域名(http://api.training.com)，这是显而易见的跨域场景。

**2  CORS详解**

**跨域资源共享**标准新增了一组 HTTP 首部字段，允许服务器声明哪些源站通过浏览器有权限访问哪些资源。

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，或者搭配某些 MIME 类型的 POST 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域请求。

服务器确认允许之后，才发起实际的 HTTP 请求。在预检请求的返回中，服务器端也可以通知客户端，是否需要携带身份凭证（包括 Cookies 和 HTTP 认证相关数据）。

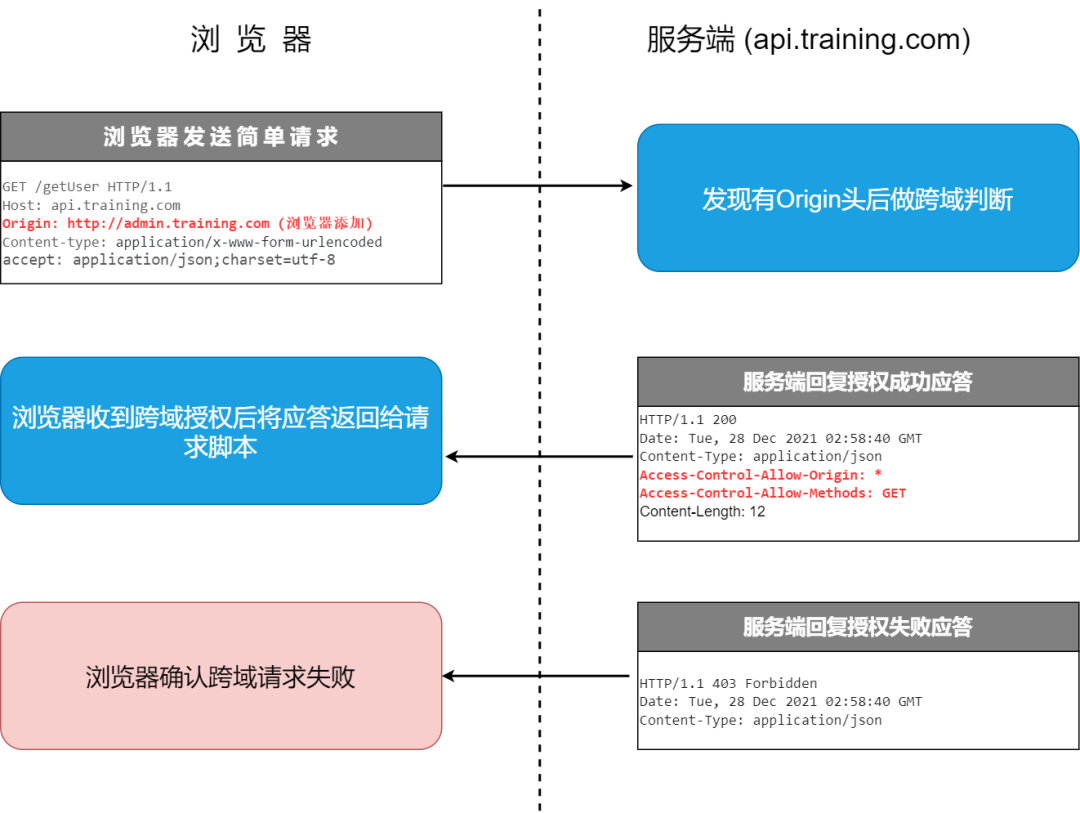


**2.1 简单请求**

当请求**同时满足如下条件时**，CORS验证机制会使用简单请求， 否则CORS验证机制会使用预检请求。

1. 使用GET、POST、HEAD其中一种方法；
2. 只使用了如下的安全首部字段，不得人为设置其他首部字段；
   * Accept
   * Accept-Language
   * Content-Language
   * Content-Type 仅限三种之一：text/plain，multipart/form-data，application/x-www-form-urlencoded：
   * HTML头部 header field字段：DPR、Download、Save-Data、Viewport-Width、WIdth
3. 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；
4. 请求中没有使用 ReadableStream 对象。

简单请求模式，浏览器直接发送跨域请求，并在请求头中携带Origin的头，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。



应答中携带了跨域头 Access-Control-Allow-Origin。使用 Origin 和 Access-Control-Allow-Origin 就能完成最简单的访问控制。本例中，服务端返回的 Access-Control-Allow-Origin: \* 表明，该资源可以被任意外域访问。如果服务端仅允许来自 http://admin.training.com 的访问，该首部字段的内容如下：

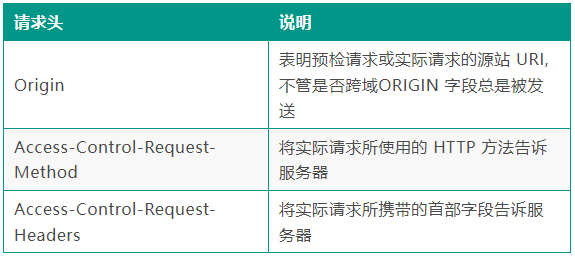
Access-Control-Allow-Origin: http://admin.training.com

现在，除了 http://admin.training.com，其它外域均不能访问该资源。

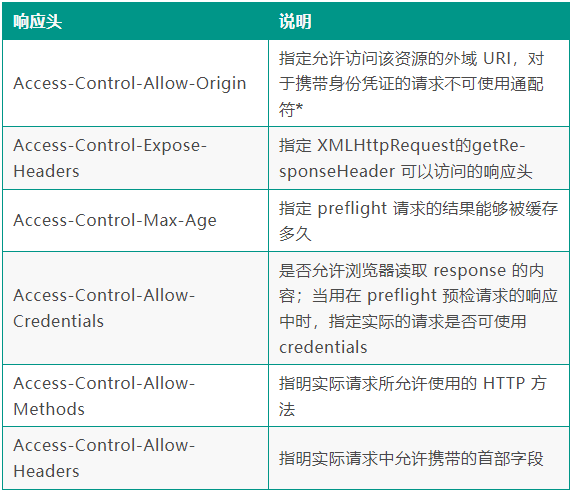
## 2.2 预检请求

浏览器在发现页面发出的请求非简单请求，并不会立即执行对应的请求代码，而是会触发预先请求模式。预先请求模式会先发送preflight request（预先验证请求），preflight request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。在得到服务器的跨域授权后才能发送真正的HTTP请求。

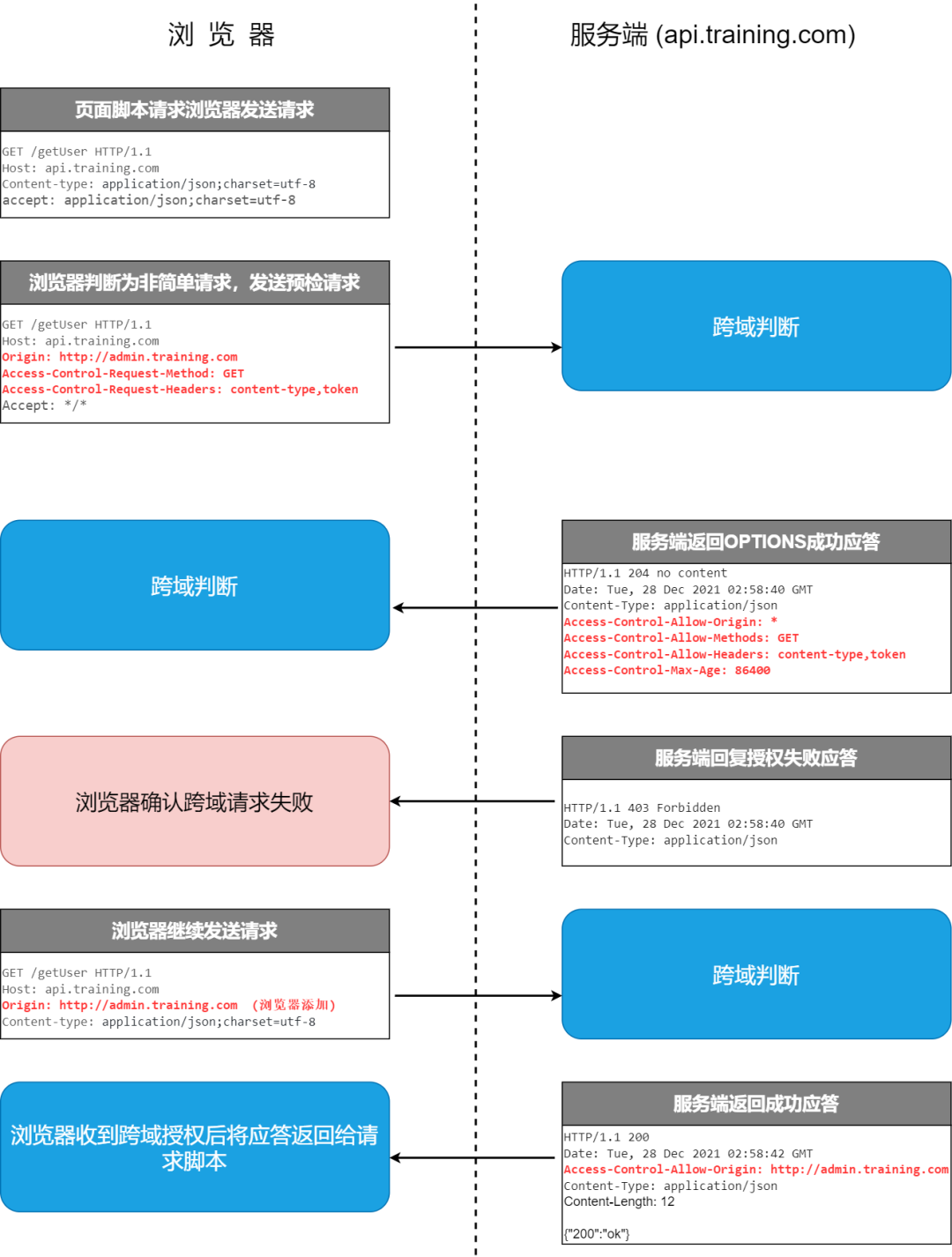
OPTIONS请求头部中会包含以下头部：



服务器收到OPTIONS请求后，设置头部与浏览器沟通来判断是否允许这个请求。



如果preflight request验证通过，浏览器才会发送真正的跨域请求。



# 3  后端配置

后端配置我尝试过两种方式，经过两个月的测试，都能非常稳定的运行。

* MND推荐的Nginx配置；
* SpringBoot自带CorsFilter配置。

## ▍MND推荐的Nginx配置

Nginx配置相当于在请求转发层配置。

location / {  
     if ($request\_method = 'OPTIONS') {  
        add\_header 'Access-Control-Allow-Origin' '\*';  
        add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS';  
        #  
        # Custom headers and headers various browsers \*should\* be OK with but aren't  
        #  
        add\_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range';  
        #  
        # Tell client that this pre-flight info is valid for 20 days  
        #  
        add\_header 'Access-Control-Max-Age' 1728000;  
        add\_header 'Content-Type' 'text/plain; charset=utf-8';  
        add\_header 'Content-Length' 0;  
        return 204;  
     }  
     if ($request\_method = 'POST') {  
        add\_header 'Access-Control-Allow-Origin' '\*' always;  
        add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;  
        add\_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range' always;  
        add\_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range' always;  
     }  
     if ($request\_method = 'GET') {  
        add\_header 'Access-Control-Allow-Origin' '\*' always;  
        add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always;  
        add\_header 'Access-Control-Allow-Headers' 'DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range' always;  
        add\_header 'Access-Control-Expose-Headers' 'Content-Length,Content-Range' always;  
     }  
}

在配置Access-Control-Allow-Headers属性的时候，因为自定义的header包含签名和token，数量较多。为了简洁方便，我把Access-Control-Allow-Headers配置成 \* 。

在Chrome和firefox下没有任何异常，但在IE11下报了如下的错：

Access-Control-Allow-Headers 列表中不存在请求标头 content-type。

原来IE11要求预检请求返回的Access-Control-Allow-Headers的值必须以逗号分隔。

## ▍SpringBoot自带CorsFilter

首先基础框架里默认有如下跨域配置。

public void addCorsMappings(CorsRegistry registry) {  
    registry.addMapping("/\*\*")  
      .allowedOrigins("\*")  
      .allowedMethods("POST", "GET", "PUT", "OPTIONS", "DELETE")  
      .allowCredentials(true)  
      .allowedHeaders("\*")  
      .maxAge(3600);  
}

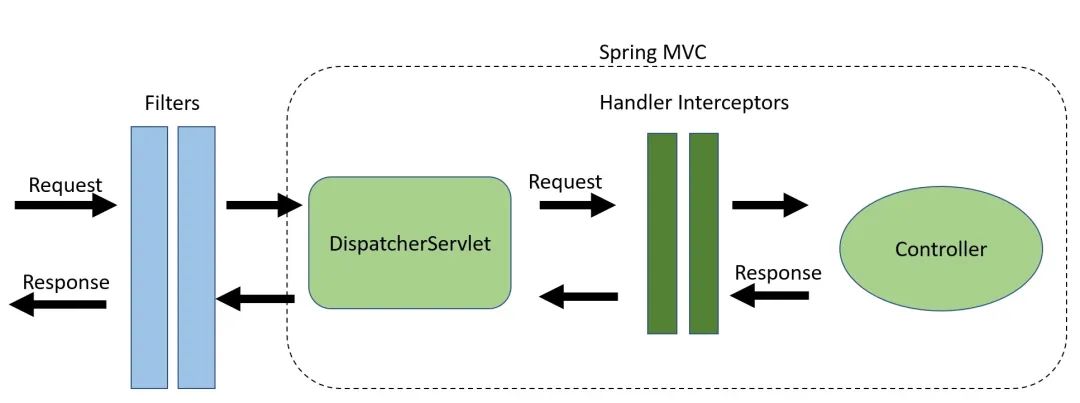
可是部署完成，进入还是报CORS异常：



从nginx和tomcat日志来看，仅仅收到一个OPTION请求，springboot应用里有一个拦截器**ActionInterceptor**，从header中获取token，调用用户服务查询用户信息，放入request中。当没有获取token数据时，会返回给前端JSON格式数据。

但从现象来看CorsMapping并没有生效。

为什么呢？实际上还是执行顺序的概念。下图展示了 过滤器，拦截器，控制器的执行顺序。



DispatchServlet.doDispatch()方法是SpringMVC的核心入口方法。



代码里有预检判断，通过 PreFlightHandler.handleRequest() 中处理，但是处于正常的业务拦截器之后。

最终选择CorsFilter 主要基于两点原因：

* 过滤器的执行顺序优先级最高；
* 通过调试CorsFilter的源码，发现源码有很多细节的处理。



下面的代码里，allowHeader是通配符 \* 的时候，CorsFilter在设置 Access-Control-Allow-Headers 的时候，会将 **Access-Control-Request-Headers** 以逗号拼接起来，这样就可以避免IE11响应头的问题。