CORS 跨域资源共享原理与实践

李睿远

Jun 21, 2026

在现代 Web 应用中，前后端分离已经成为主流架构。当前端页面位于 `https://app.example.com`，而 REST API 部署在 `https://api.example.com` 时，浏览器会严格执行同源策略，阻止跨域读取响应。CORS 机制正是为了在保留安全边界的前提下，提供受控的跨域访问能力。本文将系统梳理同源策略的判定规则、CORS 的请求流程与响应头语义，并结合 Node.js 与 Spring Boot 给出生产级配置示例。

1　同源策略的判定规则

同源策略（Same-Origin Policy）由协议、域名和端口三部分共同决定资源是否同源。只有当三者完全一致时，浏览器才允许脚本读取 DOM、Cookie 或通过 `XMLHttpRequest`/`Fetch` 获取响应。协议差异如 `http` 与 `https`，端口差异如 `:80` 与 `:8080`，都会导致跨域。早期浏览器缺乏统一的跨域方案，开发者常借助 JSONP 或反向代理绕过限制，但这些方法或牺牲安全性，或增加运维复杂度。CORS 作为 WHATWG Fetch 标准的一部分，将跨域决策权交给服务器，通过显式响应头实现细粒度授权。

2　简单请求与预检请求的区分

浏览器在发起跨域请求前，会依据方法、请求头和 `Content-Type` 判断是否为简单请求。简单请求必须满足方法属于 `GET`、`HEAD`、`POST` 之一，且 `Content-Type` 仅为 `text/plain`、`multipart/form-data` 或 `application/x-www-form-urlencoded`。当请求包含自定义头如 `X-Auth-Token` 或使用 `PUT` 方法时，浏览器会先发送一个 OPTIONS 预检请求。服务器需在预检响应中返回 `Access-Control-Allow-Methods` 和 `Access-Control-Allow-Headers`，浏览器确认后再发送实际请求。这一两阶段流程确保服务器有机会拒绝不安全的跨域意图。

3　关键响应头字段语义

服务器通过 `Access-Control-Allow-Origin` 声明允许的源。值可以是具体域名 `https://app.example.com`，也可使用通配符 `*`，但当 `Access-Control-Allow-Credentials` 为 `true` 时，通配符被禁止，且必须回显精确的 `Origin` 值。`Access-Control-Expose-Headers` 控制前端脚本可读取的自定义响应头，`Access-Control-Max-Age` 则告知浏览器预检结果可缓存的秒数，减少重复 OPTIONS 请求。需要特别注意的是，携带凭证时 `Access-Control-Allow-Origin` 不能为 `*`，否则浏览器会忽略整个响应。

4 凭证与 Cookie 的交互

当前端设置 `credentials: 'include'` 时，浏览器会在请求中携带同站 Cookie。服务器必须同时设置 `Access-Control-Allow-Credentials: true` 和精确的 `Access-Control-Allow-Origin`，否则凭证信息不会被发送。现代浏览器还引入 SameSite Cookie 属性，进一步限制跨站请求携带凭证。开发者需确保前端与后端对凭证策略的一致性，避免因配置不匹配导致静默失败。

5 Node.js Express 中间件实现

在 Express 中，可使用官方 `cors` 中间件实现动态白名单。以下代码展示如何从环境变量读取逗号分隔的域名列表，并根据请求 `Origin` 动态决定响应头。

```
const cors = require('cors');
const allowedOrigins = (process.env.ALLOWED_ORIGINS || '').split(',');

const corsOptions = {
  origin: function (origin, callback) {
    if (!origin || allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  },
  credentials: true,
  maxAge: 86400
};

app.use(cors(corsOptions));
```

这段代码首先解析环境变量，将允许的源存入数组。`origin` 函数在每次请求时接收浏览器发送的 `Origin` 头，若匹配白名单则调用 `callback(null, true)` 允许跨域，否则抛出错误。`credentials: true` 确保 `Set-Cookie` 响应头能被浏览器接受，`maxAge` 将预检结果缓存一天，降低 OPTIONS 请求频率。

6 Spring Boot 全局配置示例

Spring Boot 提供 `CorsFilter` 和 `WebMvcConfigurer` 两种方式。使用 `WebMvcConfigurer` 可在 Java 配置类中声明跨域规则。

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    @Override
```

```
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**")
                .allowedOrigins("https://app.example.com")
                .allowedMethods("GET", "POST", "PUT", "DELETE")
                .allowedHeaders("*")
                .allowCredentials(true)
                .maxAge(3600);
    }
}
```

上述配置将 `/api/**` 路径下的所有请求注册 CORS 规则。`allowedOrigins` 指定白名单，`allowedMethods` 限制允许的 HTTP 方法，`allowCredentials(true)` 开启凭证支持。`maxAge` 设置预检缓存时间为 3600 秒，减少重复验证开销。

7 前端 Fetch 配置要点

前端使用 `fetch` 时，需显式声明凭证模式以匹配后端设置。

```
fetch('https://api.example.com/users', {
  method: 'POST',
  credentials: 'include',
  headers: {
    'Content-Type': 'application/json',
    'X-Request-Id': 'abc123'
  },
  body: JSON.stringify({ name: 'Alice' })
});
```

此配置中 `credentials: 'include'` 指示浏览器发送 Cookie，`Content-Type: application/json` 会触发预检请求。自定义头 `X-Request-Id` 同样属于非简单请求头，浏览器会在实际请求前发送 OPTIONS 进行确认。

8 开发环境代理与生产差异

Vite 与 Webpack DevServer 均支持开发时代理，将跨域请求转发到后端，避免浏览器同源策略限制。配置示例中，`/api` 前缀的请求被代理到 `http://localhost:8080`，且修改 `changeOrigin` 使后端看到正确的 `Host` 头。生产环境部署时，必须移除代理，直接使用 CORS 响应头，否则会导致生产环境跨域失败。

9 多环境白名单隔离策略

生产系统通常将白名单存储在配置中心或数据库中，按环境动态加载。开发环境允许 `http://localhost:3000`，预发布环境允许 `https://staging.example.com`，生产环境仅允许正式域名。通过环境变量注入白名单，避

免在代码仓库中硬编码敏感域名，降低配置泄露风险。

10 网关层统一处理

在 Nginx 或 APISIX 等网关层实现 CORS，可避免后端服务重复配置。以 APISIX 为例，可编写 Lua 插件读取请求 `Origin`，匹配白名单后动态设置响应头。WASM 插件则提供更高性能的边缘计算能力，适合高并发场景。

11 常见错误排查

当控制台出现「No 'Access-Control-Allow-Origin'」时，首先检查响应头中是否包含该字段，以及值是否与请求 `Origin` 精确匹配。若预检返回 405，需确认服务器是否正确处理 OPTIONS 方法并返回 200。若携带 Cookie 时跨域失败，检查 `Access-Control-Allow-Credentials` 是否为 `true`，且 `Access-Control-Allow-Origin` 不是通配符。

12 演进中的新特性

Private Network Access 提案要求从公共网站访问私有 IP 地址时，必须显式授权。CHIPS（Cookies Having Independent Partitioned State）则通过 `Partitioned` 属性实现跨站 Cookie 的存储分区，减少第三方跟踪的同时，也改变了 CORS 携带凭证的行为。开发者需持续关注 Fetch 标准更新，及时调整跨域策略。

生产级 CORS 配置应遵循白名单而非通配符、开启凭证时回显精确 Origin、合理设置 Max-Age 降低预检开销、在网关层集中管理跨域策略、记录被拒绝的 Origin 以便监控。结合 CSP、COOP 等安全头，可构建纵深防御体系。建议定期审计跨域配置，避免因环境漂移导致的安全漏洞。