微服务实战篇

极客时间

扫码试看/订阅

《.NET Core 开发实战》视频课程

2.11 HttpClientFactory：管理向外请求的最佳实践

组件包

- Microsoft.Extensions.Http

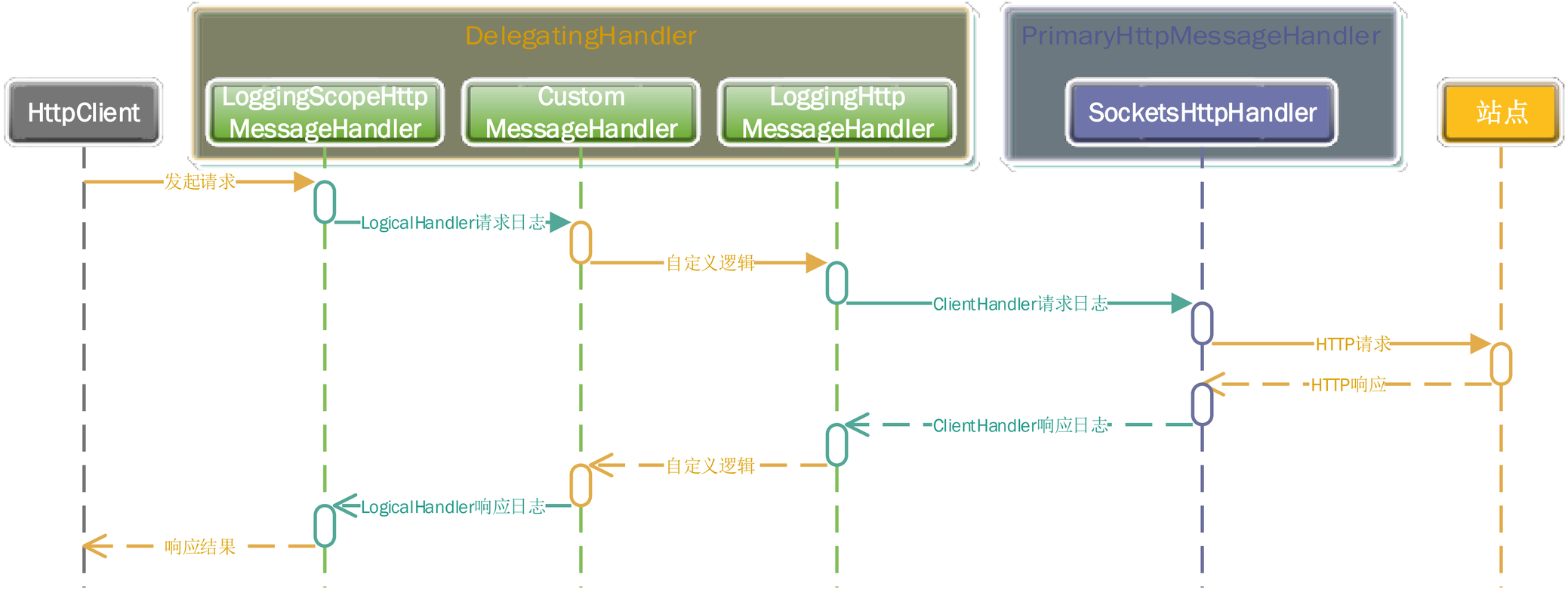
核心能力

- 管理内部 HttpMessageHandler 的生命周期，灵活应对资源问题和 DNS 刷新问题
- 支持命名化、类型化配置，集中管理配置，避免冲突
- 灵活的出站请求管道配置，轻松管理请求生命周期
- 内置管道最外层和最内层日志记录器，有 Information 和 Trace 输出

核心对象

- HttpClient
- HttpMessageHandler
- SocketsHttpHandler
- DelegatingHandler
- IHttpClientFactory
- IHttpClientBuilder

管道模型


DelegatingHandler
PrimaryHttpMessageHandler
HttpClient
LoggingScopeHttp MessageHandler
Custom MessageHandler
LoggingHttp MessageHandler
SocketsHttpHandler
站点
发起请求
LogicalHandler请求日志
自定义逻辑
ClientHandler请求日志
HTTP请求
HTTP响应
ClientHandler响应日志
自定义逻辑
LogicalHandler响应日志
响应结果

创建模式

- 工厂模式
- 命名客户端模式
- 类型化客户端模式

2.12 gRPC：内部服务间通讯利器

什么是 gRPC

- 一个远程过程调用框架
- 由 Google 公司发起并开源

gRPC 的特点

- 提供几乎所有主流语言的实现，打破语言隔阂
- 基于 HTTP/2 ，开放协议，受到广泛的支持，易于实现和集成
- 默认使用 Protocol Buffers 序列化，性能相较于 RESTful Json 好很多
- 工具链成熟，代码生成便捷，开箱即用
- 支持双向流式的请求和响应，对批量处理、低延时场景友好

.NET 生态对 gRPC 的支持情况

- 提供基于 HttpClient 的原生框架实现
- 提供原生的 ASP.NET Core 集成库
- 提供完整的代码生成工具
- Visual Studio 和 Visual Stuido Code 提供 proto 文件的智能提示

服务端核心包

- Grpc.AspNetCore

客户端核心包

- Google.Protobuf
- Grpc.Net.Client
- Grpc.Net.ClientFactory
- Grpc.Tools

.proto 文件

- 定义包、库名
- 定义服务 “service”
- 定义输入输出模型 “message”

gRPC 异常处理

- 使用 Grpc.Core.RpcException
- 使用 Grpc.Core.Interceptors.Interceptor

gRPC 与 HTTPS 证书

- 使用自制证书
- 使用非加密的 HTTP2

2.13 gRPC：用代码生成工具提高生产效率

工具核心包

- Grpc.Tools
- dotnet-grpc

命令

- dotnet grpc add-file
- dotnet grpc add-url
- dotnet grpc remove
- dotnet grpc refresh

最佳实践

- 使用单独的 Git 仓库管理 proto 文件
- 使用 submodule 将 proto 文件集成到工程目录中
- 使用 dotnet-grpc 命令行添加 proto 文件及相关依赖包引用

备注：

由 proto 生成的代码文件会存放在 obj 目录中，不会被签入到 Git 仓库

2.14 Polly：用失败重试机制提升服务可用性

Polly 组件包

- Polly
- Polly.Extensions.Http
- Microsoft.Extensions.Http.Polly

Polly 的能力

- 失败重试
- 服务熔断
- 超时处理
- 舱壁隔离
- 缓存策略
- 失败降级
- 组合策略

Polly 使用步骤

- 定义要处理的异常类型或返回值
- 定义要处理动作（重试、熔断、降级响应等）
- 使用定义的策略来执行代码

适合失败重试的场景

- 服务“失败”是短暂的，可自愈的
- 服务是幂等的，重复调用不会有副作用

场景举例

- 网络闪断
- 部分服务节点异常

最佳实践

- 设置失败重试次数
- 设置带有步长策略的失败等待间隔
- 设置降级响应
- 设置断路器

2.15 Polly：熔断慢请求避免雪崩效应

策略的类型

- 被动策略（异常处理、结果处理）
- 主动策略（超时处理、断路器、舱壁隔离、缓存）

组合策略

- 降级响应
- 失败重试
- 断路器
- 舱壁隔离

策略与状态共享

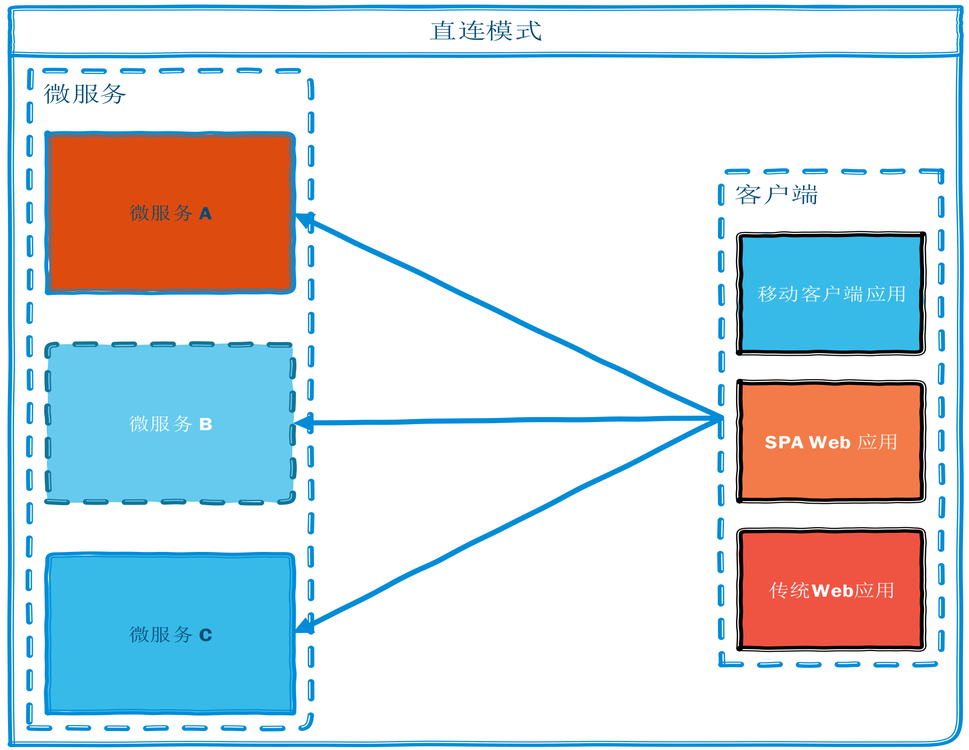
Policy 类型	状态	说明
CircuitBreaker	有状态	共享成功失败率，以决定是否熔断
Bulkhead	有状态	共享容量使用情况，以决定是否执行动作
Cache	有状态	共享缓存的对象，以决定是否命中
其它策略	无状态	-

2.16 网关与 BFF：区分场景与职责

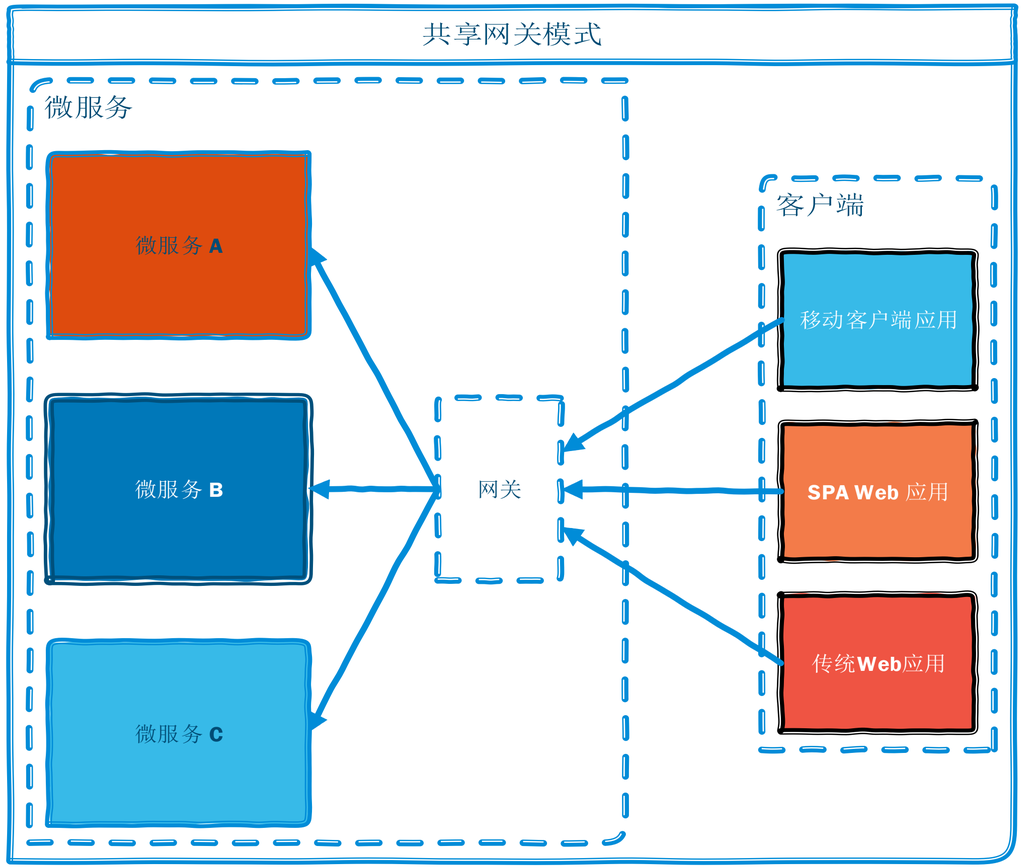
什么是 BFF

- 全称为 Backend For Frontend
- 负责认证/授权
- 负责服务聚合
- 目标是为前端的诉求服务
- 网关职责的一种进化

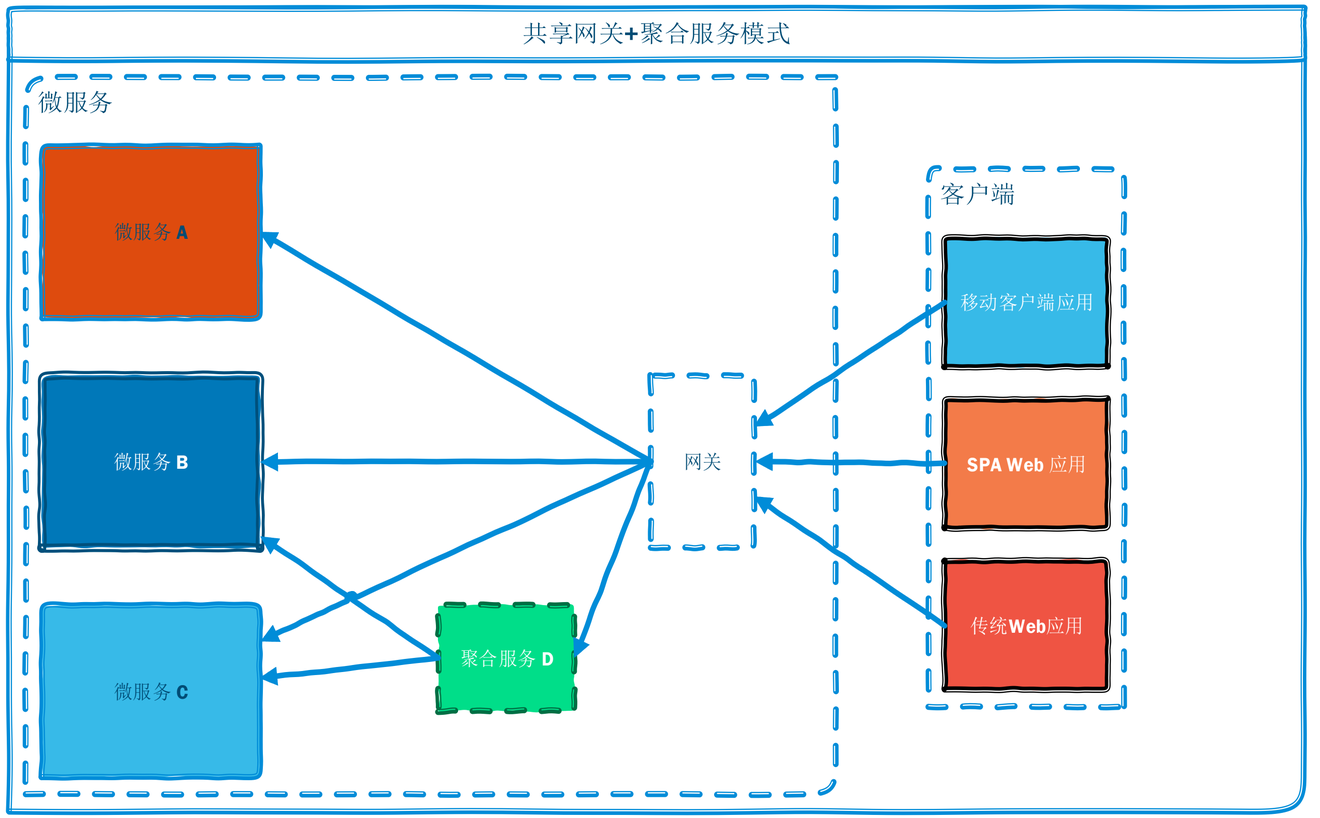
直连模式



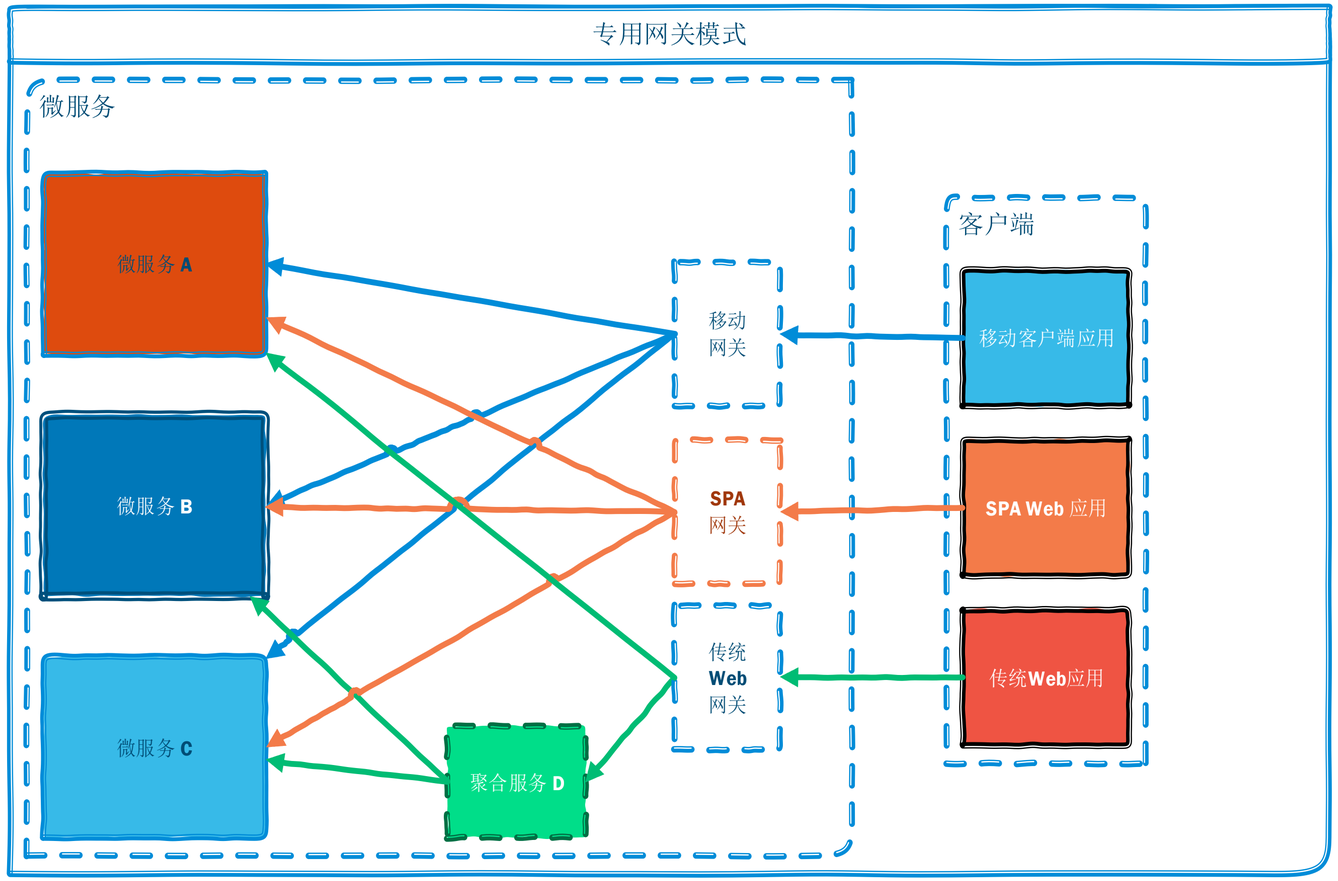
共享网关模式



共享网关+聚合服务模式


共享网关+聚合服务模式
微服务
微服务 A
微服务 B
微服务 C
聚合服务 D
网关
客户端
移动客户端应用
SPA Web 应用
传统Web应用

专用网关模式


专用网关模式
微服务
微服务 A
微服务 B
微服务 C
聚合服务 D
移动
网关
SPA
网关
传统
Web
网关
客户端
移动客户端应用
SPA Web 应用
传统Web应用

打造网关

- 添加包 Ocelot 14.0.3
- 添加配置文件 ocelot.json
- 添加配置读取代码
- 注册 Ocelot 服务
- 注册 Ocelot 中间件

2.17 网关与 BFF：使用 JWT 来实现身份认证与授权

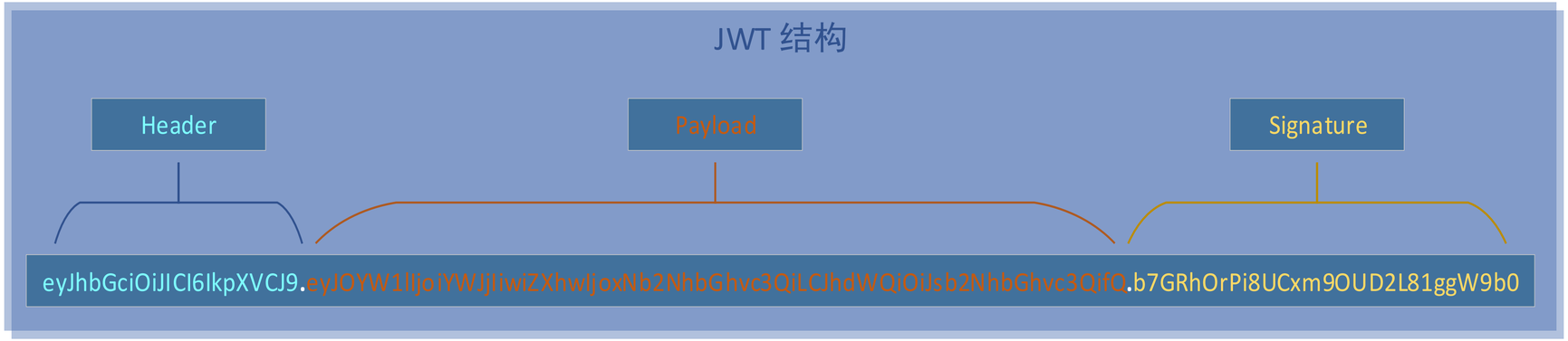
身份认证方案介绍

- Cookie
- JWT Bearer

JWT 是什么

- 全称 JSON Web Tokens
- 支持签名的数据结构

JWT 数据结构



- Header，令牌类型、加密类型等信息
- Payload，表示令牌内容，预定义了部分字段信息，支持自定义
- Signature，根据 Header、Payload 和 私有密钥计算出来的签名

启用 JwtBearer 身份认证

- Microsoft.AspNetCore.Authentication.JwtBearer

配置身份认证

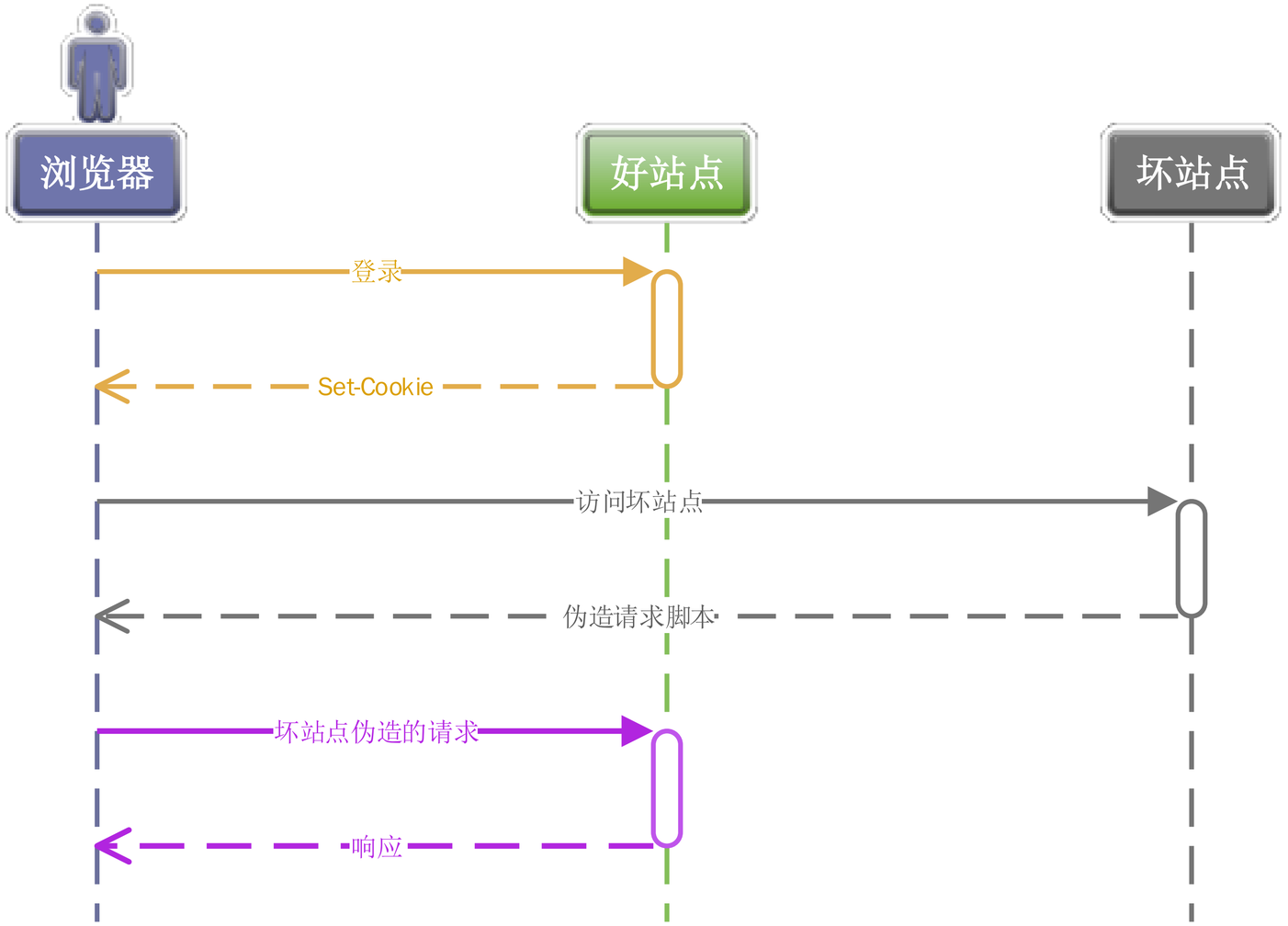
- Ocelot 网关配置身份认证
- 微服务配置认证与授权

JWT 注意事项

- Payload 信息不宜过大
- Payload 不宜存储敏感信息

2.18 安全：反跨站请求伪造

攻击过程



攻击核心

- 用户已登录“好站点”
- “好站点”通过 Cookie 存储和传递身份信息
- 用户访问了“坏站点”

如何防御

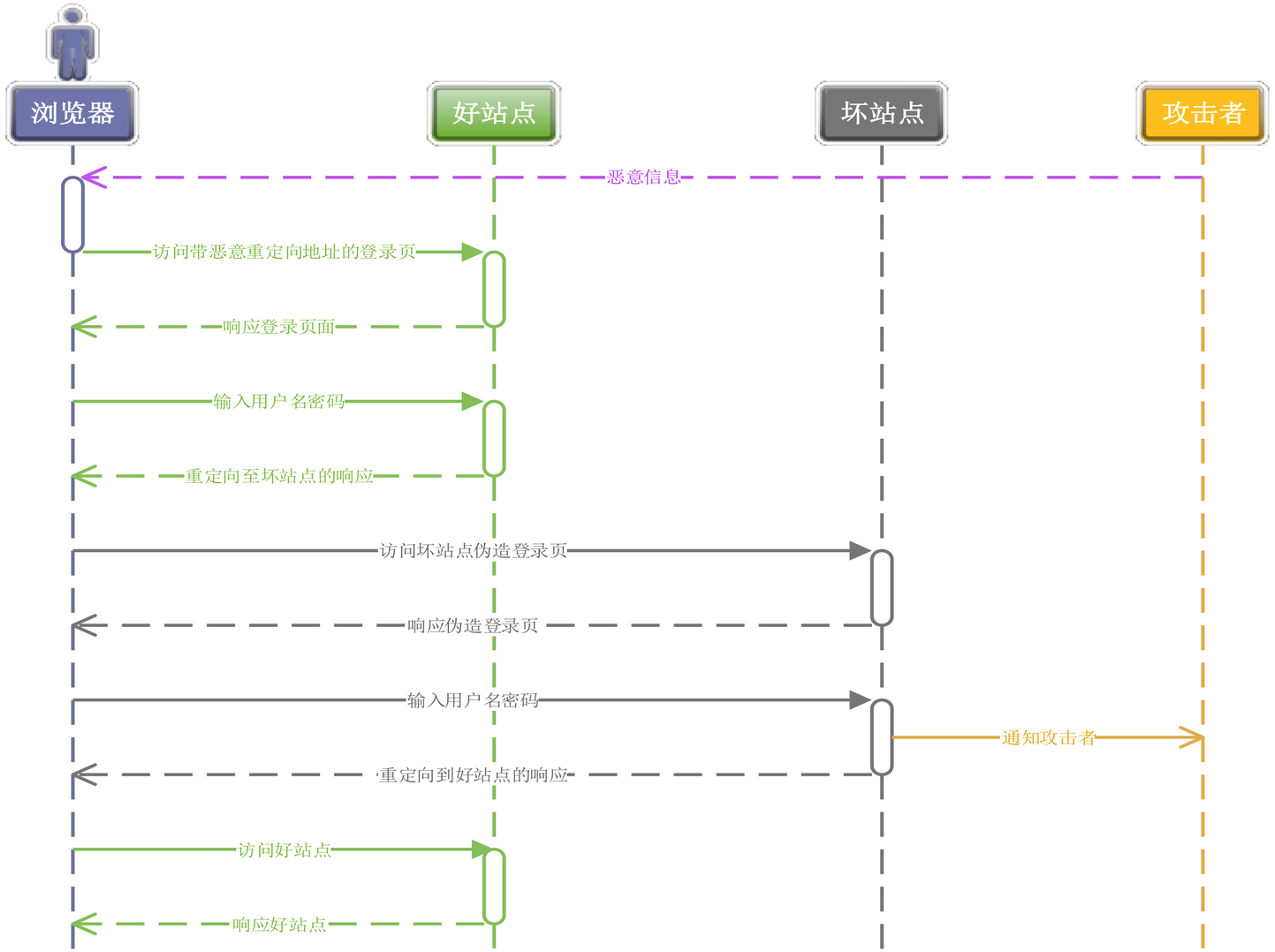
- 不使用 Cookie 来存储和传输身份信息
- 使用 AntiforgeryToken 机制来防御
- 避免使用 GET 作为业务操作的请求方法

两种选择

- ValidateAntiForgeryToken
- AutoValidateAntiforgeryToken

2.19 安全：防开放重定向攻击

攻击过程


浏览器
好站点
坏站点
攻击者
恶意信息
访问带恶意重定向地址的登录页
响应登录页面
输入用户名密码
重定向至坏站点的响应
访问坏站点伪造登录页
响应伪造登录页
输入用户名密码
通知攻击者
重定向到好站点的响应
访问好站点
响应好站点

攻击核心

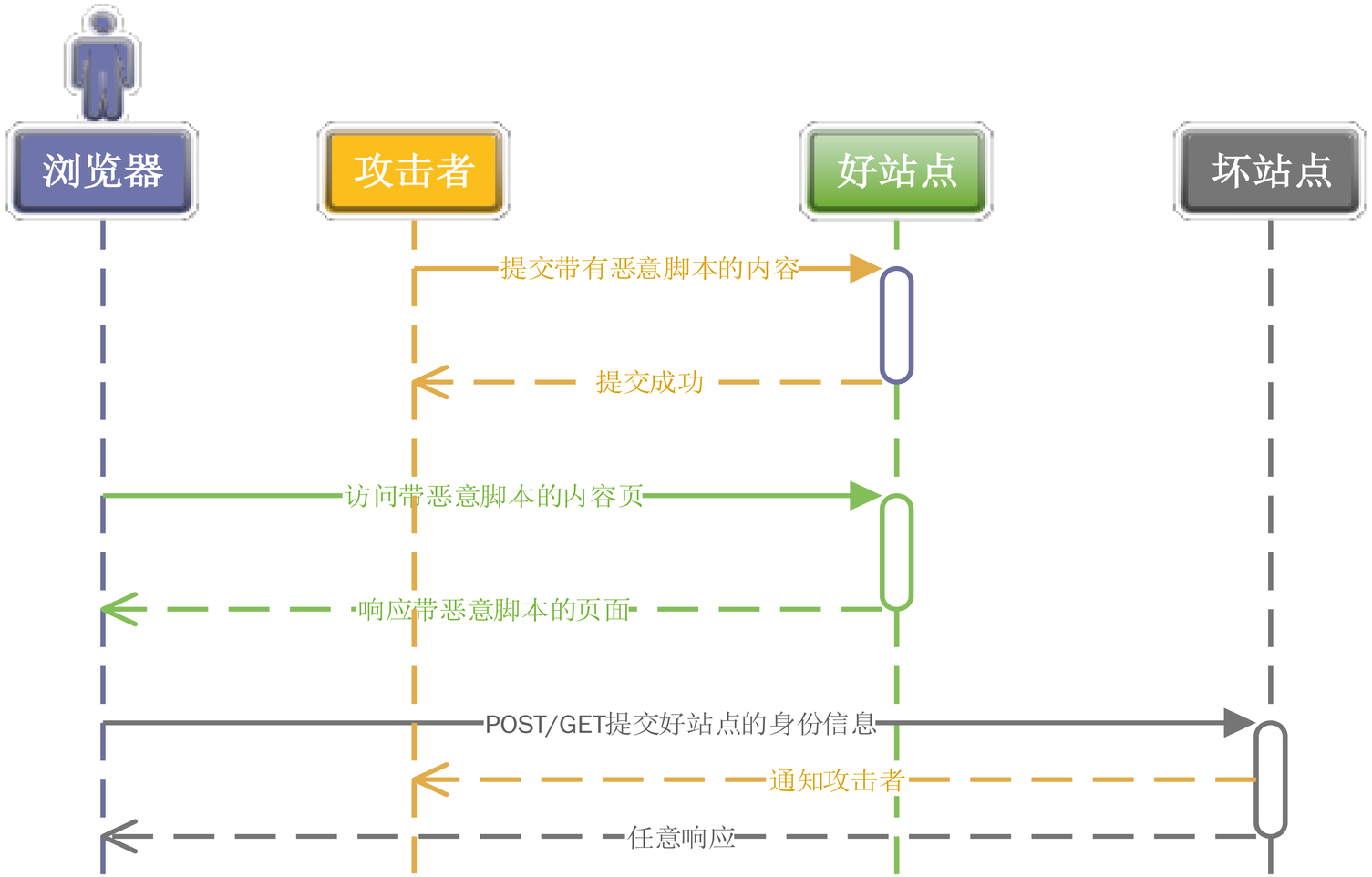
- “好站点”的重定向未验证目标 URL
- 用户访问了“坏站点”

防范措施

- 使用 LocalRedirect 来处理重定向
- 验证重定向的目标域名是否合法

2.20 安全：防跨站脚本

攻击过程


浏览器
攻击者
好站点
坏站点
提交带有恶意脚本的内容
提交成功
访问带恶意脚本的内容页
响应带恶意脚本的页面
POST/GET提交好站点的身份信息
通知攻击者
任意响应

防范措施

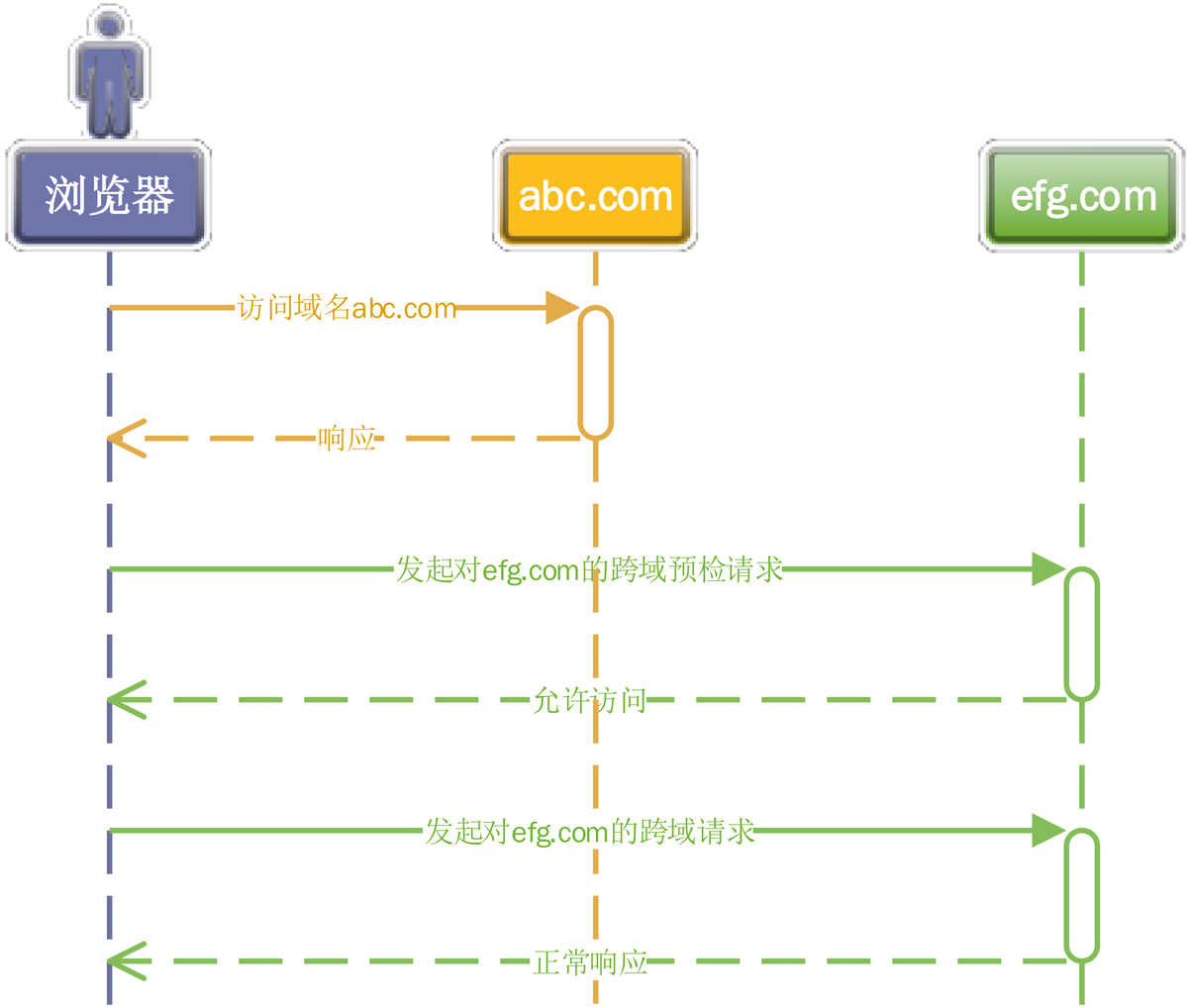
- 对用户提交内容进行验证，拒绝恶意脚本
- 对用户提交的内容进行编码 UrlEncoder、JavaScriptEncoder、UrlEncoder
- 慎用 HtmlString 和 HtmlHelper.Raw
- 身份信息 Cookie 设置为 HttpOnly
- 避免使用 Path 传递带有不受信的字符，使用 Query 进行传递

2.21 安全：跨域请求

同源与跨域

- 方案相同 (HTTP/HTTPS)
- 主机（域名）相同
- 端口相同

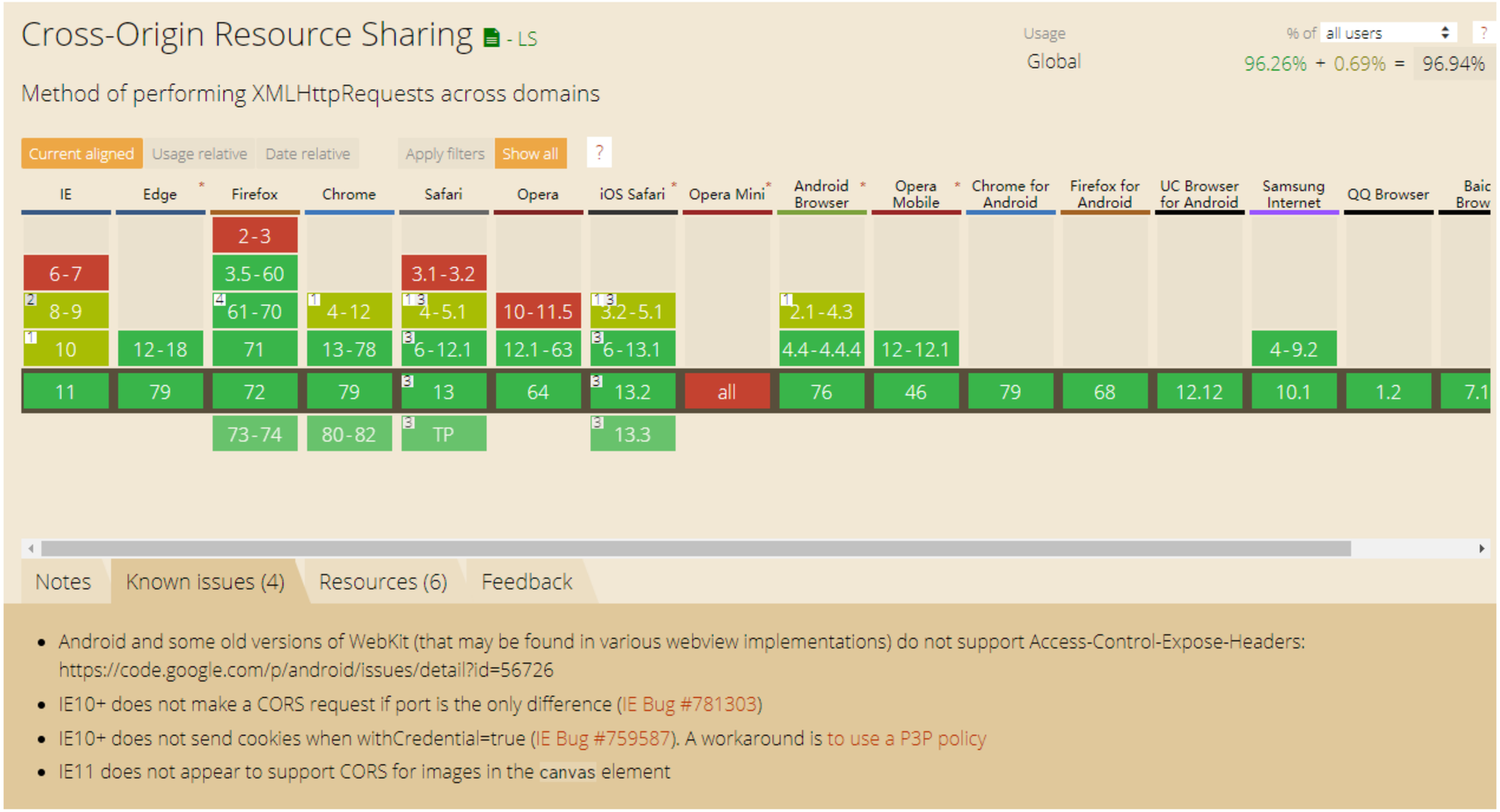
CORS 过程


浏览器
abc.com
efg.com
访问域名abc.com
响应
发起对efg.com的跨域预检请求
允许访问
发起对efg.com的跨域请求
正常响应

CORS 是什么

- CORS 是浏览器允许跨域发起请求“君子协定”
- 它是浏览器行为协议
- 它并不会让服务器拒绝其它途径发起的 HTTP 请求
- 开启时需要考虑是否存在被恶意网站攻击的情形

浏览器支持情况


Cross-Origin Resource Sharing - LS
Method of performing XMLHttpRequests across domains
Usage
Global
% of all users
96.26% + 0.69% = 96.94%
Current aligned
Usage relative
Date relative
Apply filters
Show all
IE
Edge
Firefox
Chrome
Safari
Opera
iOS Safari
Opera Mini
Android Browser
Opera Mobile
Chrome for Android
Firefox for Android
UC Browser for Android
Samsung Internet
QQ Browser
Baic Brow
2-3
6-7
3.5-60
3.1-3.2
8-9
61-70
4-12
4-5.1
10-11.5
3.2-5.1
2.1-4.3
10
12-18
71
13-78
6-12.1
12.1-63
6-13.1
4.4-4.4.4
12-12.1
4-9.2
11
79
72
79
13
64
13.2
all
76
46
79
68
12.12
10.1
1.2
7.1
73-74
80-82
TP
13.3
Notes
Known issues (4)
Resources (6)
Feedback
Android and some old versions of WebKit (that may be found in various webview implementations) do not support Access-Control-Expose-Headers: https://code.google.com/p/android/issues/detail?id=56726
IE10+ does not make a CORS request if port is the only difference (IE Bug #781303)
IE10+ does not send cookies when withCredential=true (IE Bug #759587). A workaround is to use a P3P policy
IE11 does not appear to support CORS for images in the canvas element

CORS 请求头

- Origin 请求源
- Access-Control-Request-Method
- Access-Control-Request-Headers

CORS 响应头

- Access-Control-Allow-Origin
- Access-Control-Allow-Credentials
- Access-Control-Expose-Headers
- Access-Control-Max-Age
- Access-Control-Allow-Methods
- Access-Control-Allow-Headers

默认支持的 Expose Headers

- Cache-Control
- Content-Language
- Content-Type
- Expires
- Last-Modified
- Pragma

2.22 缓存：为不同的场景设计合适的缓存策略

缓存是什么

- 缓存是计算结果的“临时”存储和重复使用
- 缓存本质是用“空间”换取“时间”

现实生活中的“缓存”

- 车站的等待大厅
- 物流仓库
- 家里的冰箱

缓存的场景

- 计算结果，如：反射对象缓存
- 请求结果，如：DNS 缓存
- 临时共享数据，如：会话存储
- 热点访问内容页，如：商品详情
- 热点变更逻辑数据，如：秒杀的库存数

缓存的策略

- 越接近最终的输出结果（靠前），效果越好
- 缓存命中率越高越好，命中率低就意味着“空间”的浪费

缓存位置

- 浏览器中
- 反向代理服务器中（负载均衡）
- 应用进程内存中
- 分布式存储系统中

缓存实现的要点

- 缓存 Key 生成策略，表示缓存数据的范围、业务含义
- 缓存失效策略，如：过期时间机制、主动刷新机制
- 缓存更新策略，表示更新缓存数据的时机

几个问题

- 缓存失效，导致数据不一致
- 缓存穿透，查询无数据时，导致缓存不生效，查询都落在数据库
- 缓存击穿，缓存失效瞬间，大量请求访问到数据库
- 缓存雪崩，大量缓存同一时间失效，导致数据库压力

用到的组件

- ResponseCache
- Microsoft.Extensions.Caching.Memory.IMemoryCache
- Microsoft.Extensions.Caching.Distributed.IDistributedCache
- EasyCaching

内存缓存和分布式缓存的区别

- 内存缓存可以存储任意的对象
- 分布式缓存的对象需要支持序列化
- 分布式缓存远程请求可能失败，内存缓存不会

极客时间

扫码试看/订阅

《.NET Core 开发实战》视频课程