HTTP 协议探究，基于 Node.js 实战编写 HTTP Server

吴阳

DAY 3

目录

访问控制

Status Code：401

HTTP/1.1 401 Unauthorized

当前请求需要用户验证。

当需要用户认证信息或者认证信息不正确时
可以使用 401 来提示客户端对用户身份进行认证

Header：WWW-Authenticate

定义使用何种验证方式去获取对资源的连接。

见于 Response 之中

用法：

WWW-Authenticate: <type> realm=<realm>

配合 401 响应码可以唤醒浏览器用户密码输入窗口

本例只实现 type=basic 模式

WWW-Authenticate: Basic realm= “login"

Header：Authorization

用于传递服务器用于验证用户代理身份的凭证。

见于 Request 之中

用法：

Authorization: <type> <credentials>

本例只实现 type=basic

Authorization: Basic <base64(user:pass)>

Header：Set-Cookie

服务器端向客户端发送 cookie 。

见于 Response 之中

用法：

Set-Cookie: <cookie-name>=<cookie-value>

Set-Cookie: <cookie-name>=<cookie-value>; <option>=<value>

本例只实现【用法 1】

Header：Cookie

用于向服务器传递之前由 Set-Cookie 设置的 Cookie

见于 Request 之中

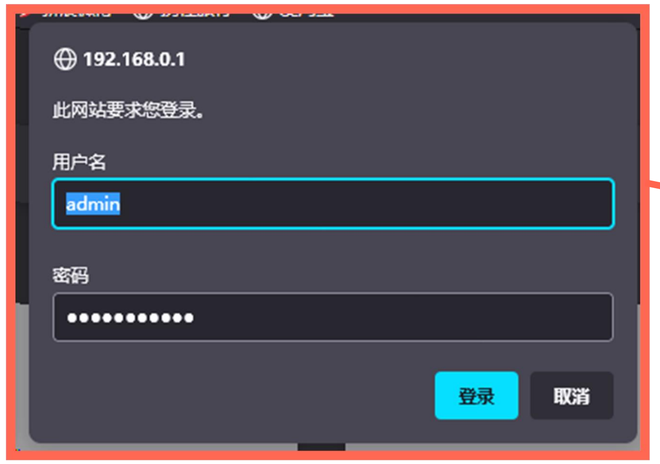
用法：

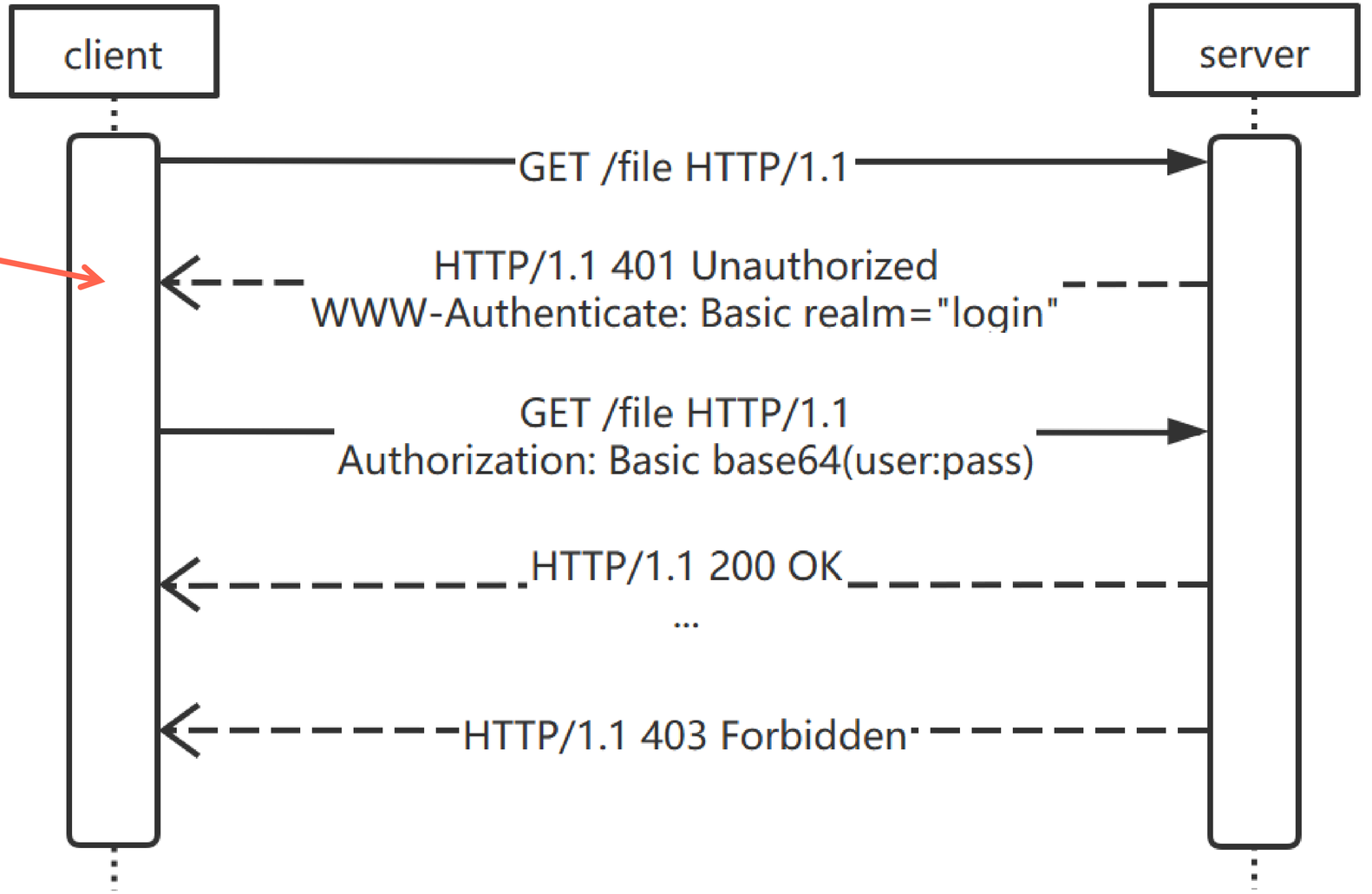
Cookie: <cookie-list>

Cookie: name=value

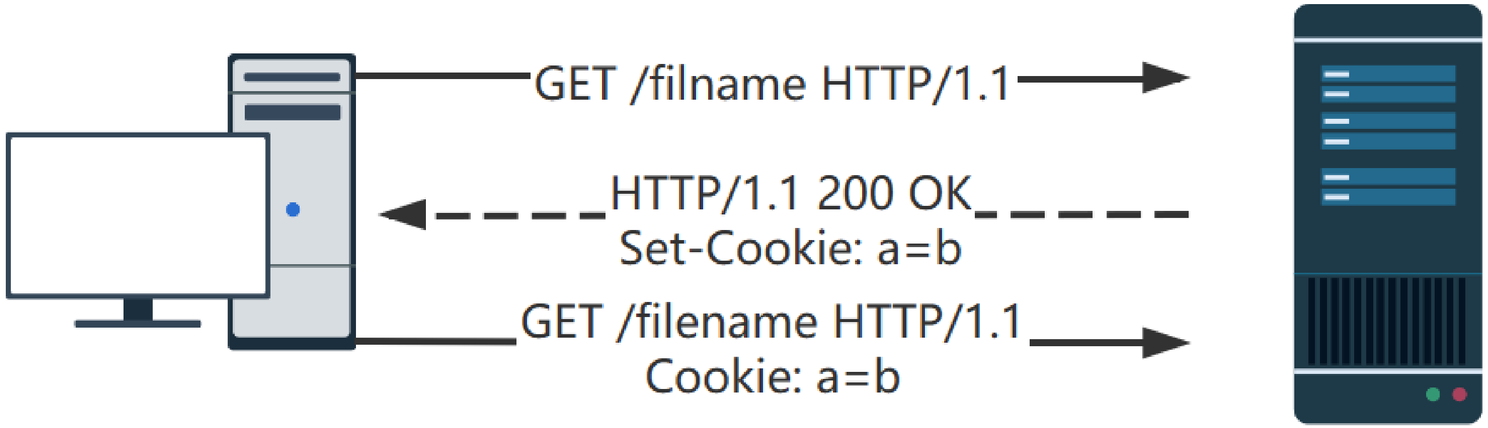
Cookie: name=value; name2=value2; name3=value3

访问控制 (HTTP Basic 认证)



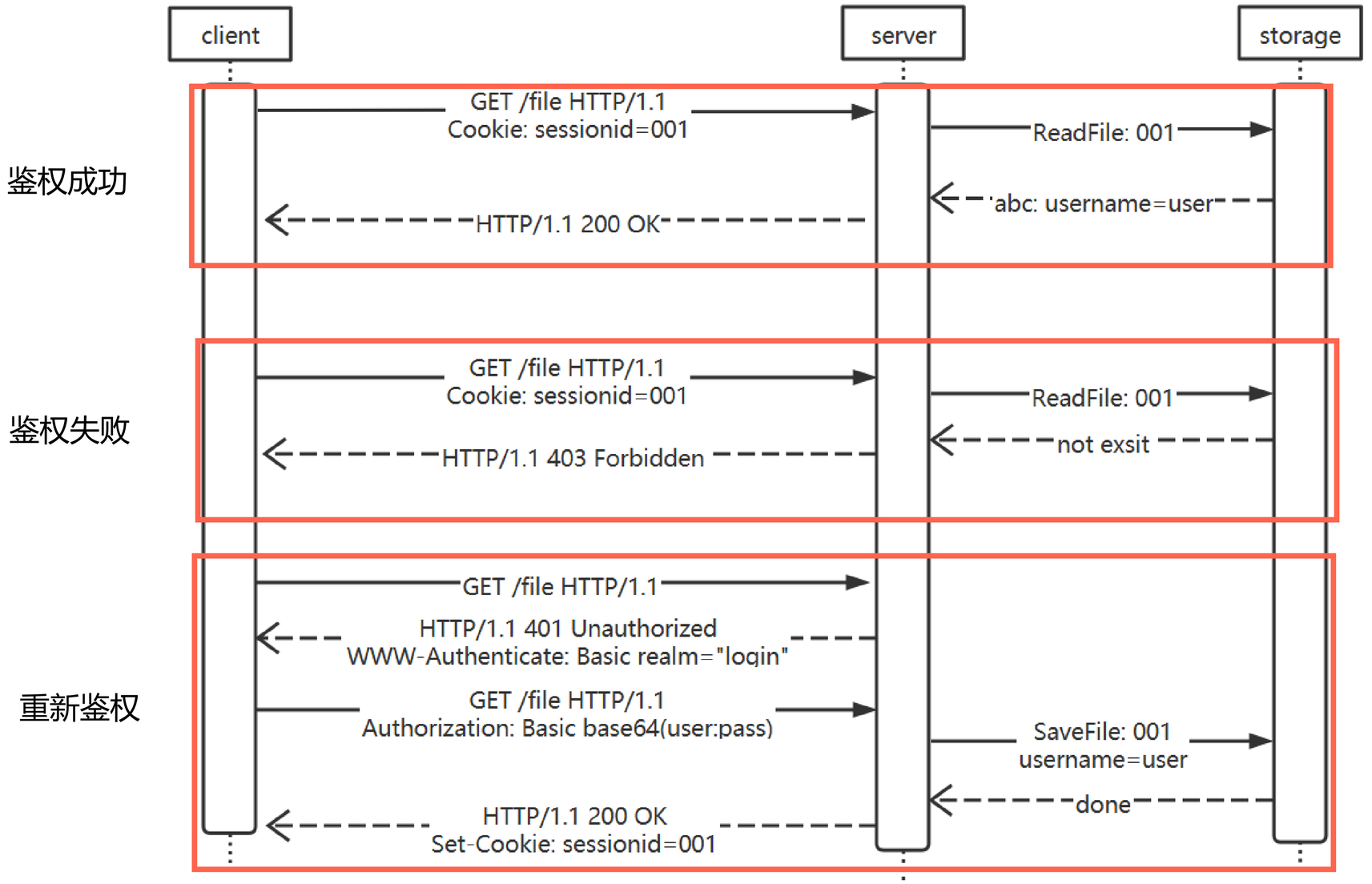


访问控制（cookie）



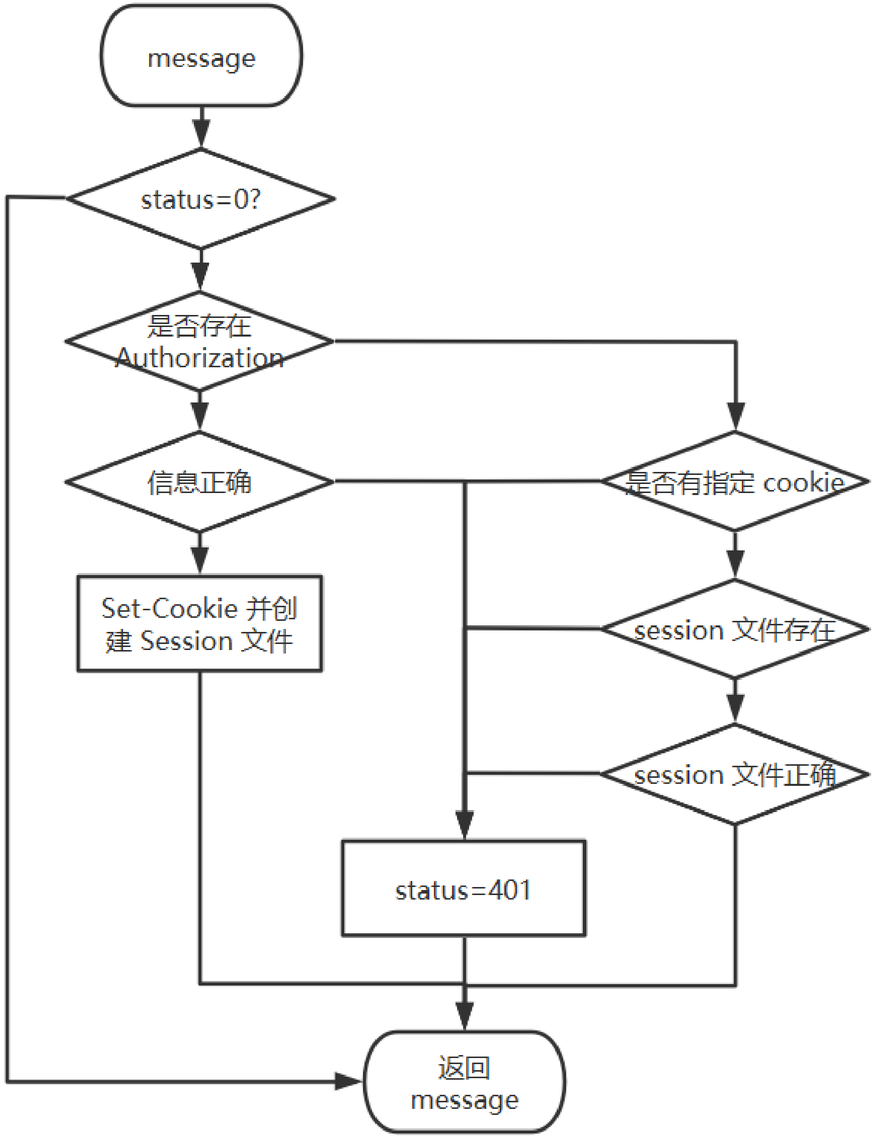
Cookie 的内容还包含域名和有效期，这里只展示了 key=value

访问控制 (session)



- Session 是因为数据安全原因将 Cookie 的内容保存在服务端的一种方案
- Session 文件可以存储在服务端的任意持久化或非持久化介质上

访问控制



新知识点

- Cookie
- Set-Cookie
- Authorization
- WWW-Authenticate
- Status Code：401

跨域访问

同源策略与跨域

同源策略是一个重要的安全策略

它用于限制一个 origin 的文档或者它加载的脚本如何能与另一个源的资源进行交互。

它能帮助阻隔恶意文档，减少可能被攻击的媒介。

如果两个 URL 的 protocol、port 和 host 都相同的话，则这两个 URL 是同源。

对不满足同源策略的 URL 请求，被称为跨域请求

http://a.com:81/b/c.html

[protocol]://[host]:[port][uri]

跨源资源共享（CORS）

- CORS 标准新增了一组 HTTP Header，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。

- 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 (Preflight) 请求，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。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。

HTTP Method：OPTIONS

用于获取目的资源所支持的通信选项 (HTTP/1.1)

在 CORS 中，可以使用 OPTIONS 方法发起一个预检 (Preflight) 请求以检测实际请求是否可以被服务器所接受。

Header：Access-Control-Allow-Origin

指定了该响应的资源是否被允许与给定的 Origin 共享

见于 Response 之中

用法：

Access-Control-Allow-Origin: *

Access-Control-Allow-Origin: <origin>

Header：Access-Control-Allow-Methods

对预检 (Preflight) 请求的应答中明确了客户端所要访问的资源允许使用的方法或方法列表。

见于 Response 之中

用法：

Access-Control-Allow-Methods: <method>, <method>, ...

Header：Access-Control-Allow-Credentials

是否接受请求中的 Credentials

Credentials 可以是 cookies, authorization headers 或 TLS client certificates。

见于 Response 之中

用法：

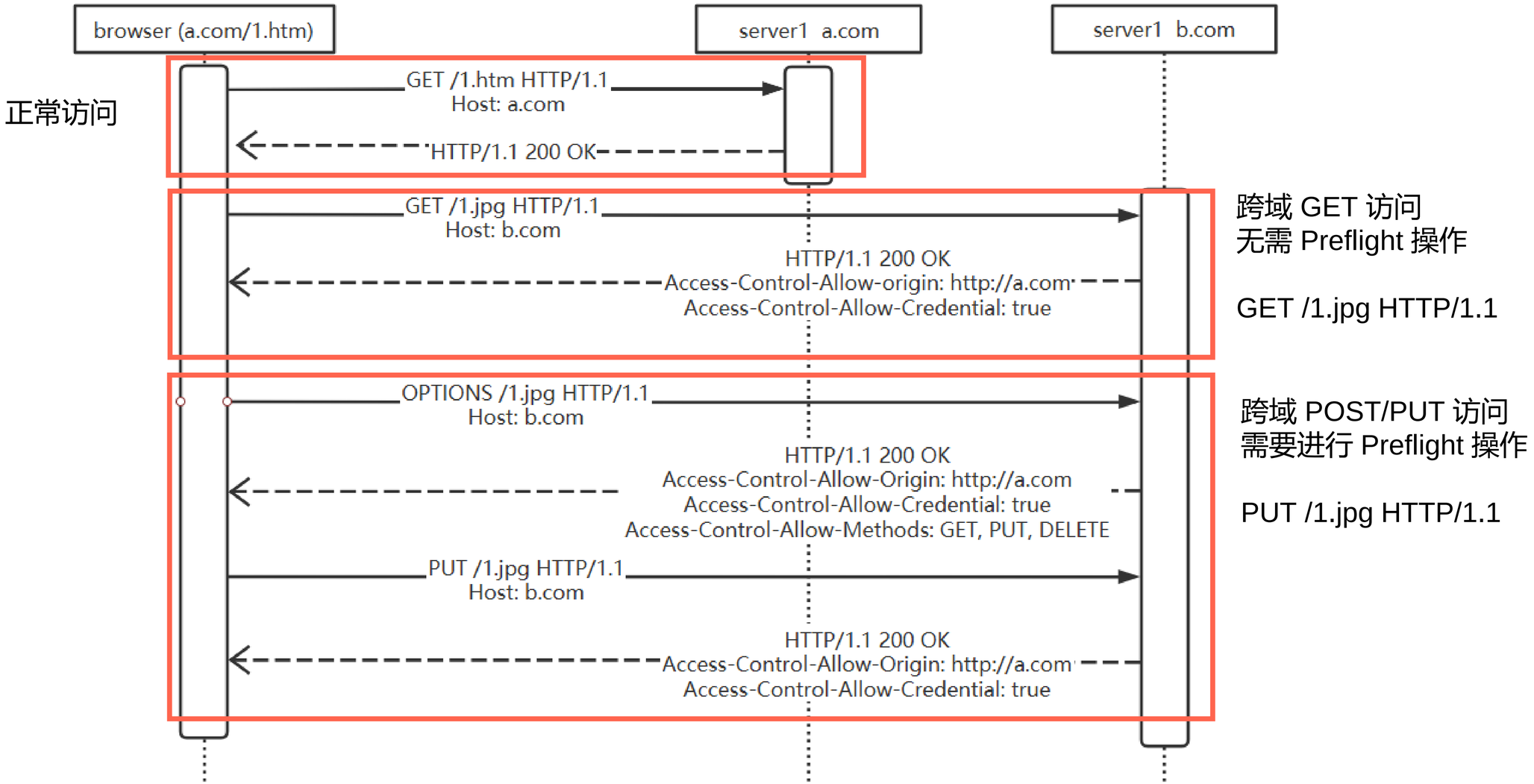
Access-Control-Allow-Credentials: true

跨域请求时使用以下方法发送 Credentials

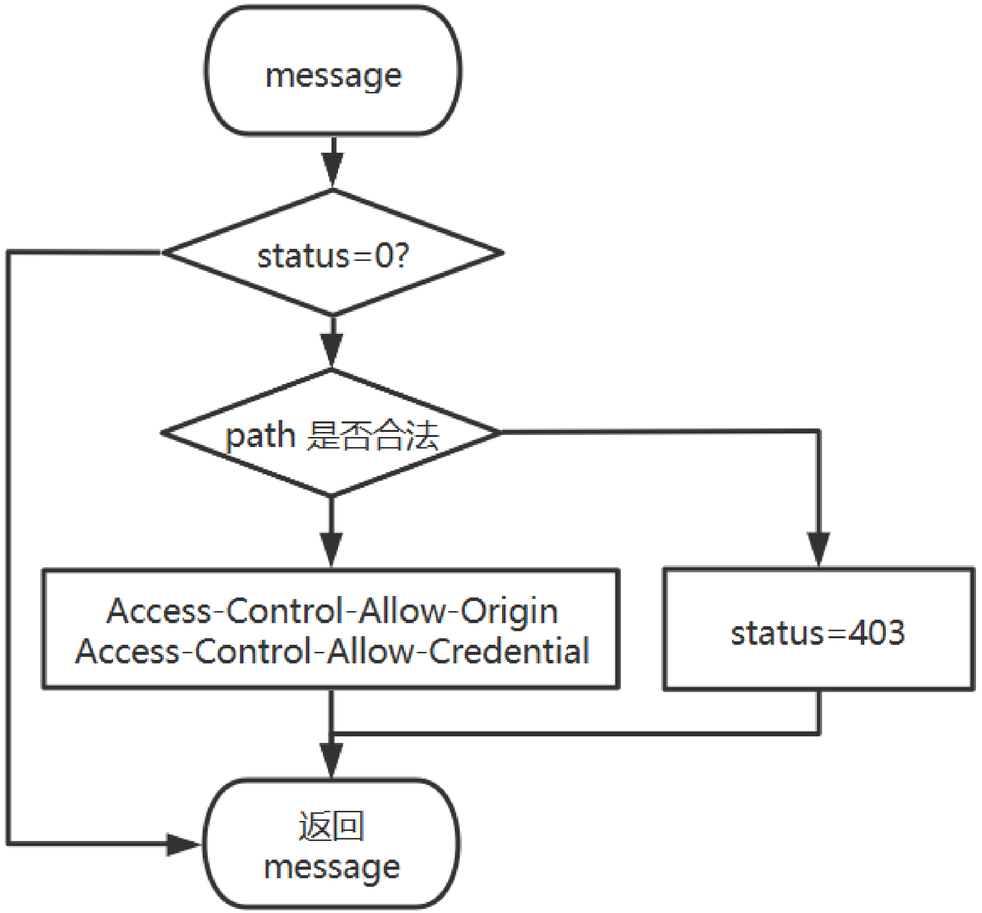
```
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;

fetch(url, { credentials: 'include' })
```

跨域访问 (原理)


browser (a.com/1.htm)
server1 a.com
server1 b.com
正常访问
GET /1.htm HTTP/1.1
Host: a.com
HTTP/1.1 200 OK
GET /1.jpg HTTP/1.1
Host: b.com
HTTP/1.1 200 OK
Access-Control-Allow-origin: http://a.com
Access-Control-Allow-Credential: true
OPTIONS /1.jpg HTTP/1.1
Host: b.com
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://a.com
Access-Control-Allow-Credential: true
Access-Control-Allow-Methods: GET, PUT, DELETE
PUT /1.jpg HTTP/1.1
Host: b.com
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://a.com
Access-Control-Allow-Credential: true
跨域 GET 访问
无需 Preflight 操作
GET /1.jpg HTTP/1.1
跨域 POST/PUT 访问
需要进行 Preflight 操作
PUT /1.jpg HTTP/1.1

跨域访问 (CORS)

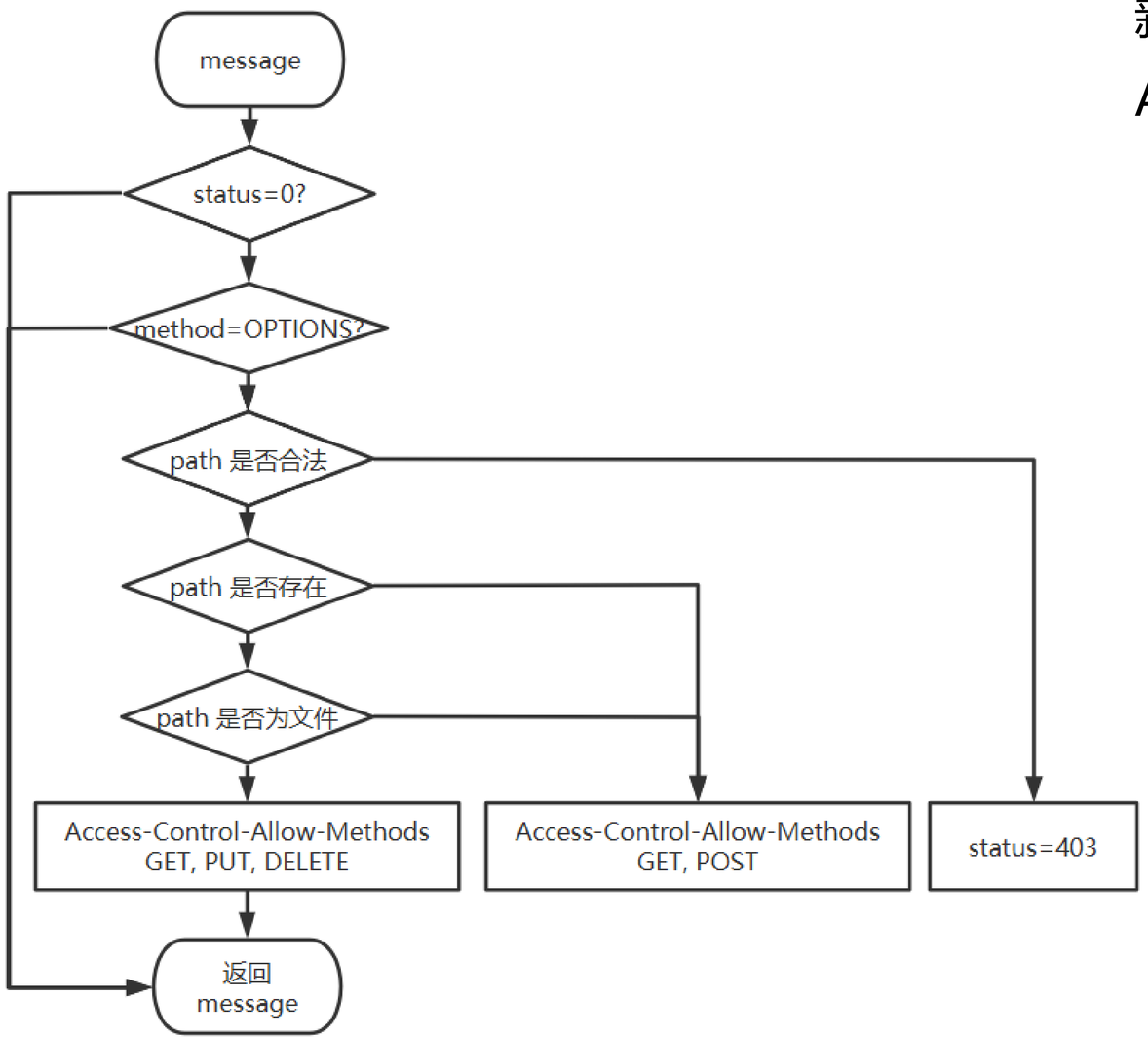


新知识点

Access-Control-Allow-Origin

Access-Control-Allow-Credentials

跨域访问 (Preflight)



新知识点

Access-Control-Allow-Methods

浏览器缓存控制

Status Code：304

HTTP/1.1 304 Not Modified

说明无需再次传输请求的内容，也就是说可以使用缓存的内容。

可以使用 304 通知浏览器直接使用缓存内容来代替请求的资源

Header：Cache-Control

通过指定指令来实现缓存机制。

是否允许缓存资源 (Response)，或者请求允许缓存资源 (Request)

见于 Request 和 Response 之中

例如：

Cache-Control: max-age=<seconds>

表示缓存有效期

Header：ETag

资源的指纹（版本标识符）

见于 Response 之中

用法：

ETag: W/"<etag_value>"

ETag: "<etag_value>"

W 表示大小写敏感

Header：Last-Modified

资源的最后修改时间

见于 Response 之中

用法：

Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT

Header：If-None-Match

用于向服务器传递缓存资源的 ETag 值用于判断资源是否发生改变，如果没有匹配则返回资源内容。

见于 Request 之中

用法：

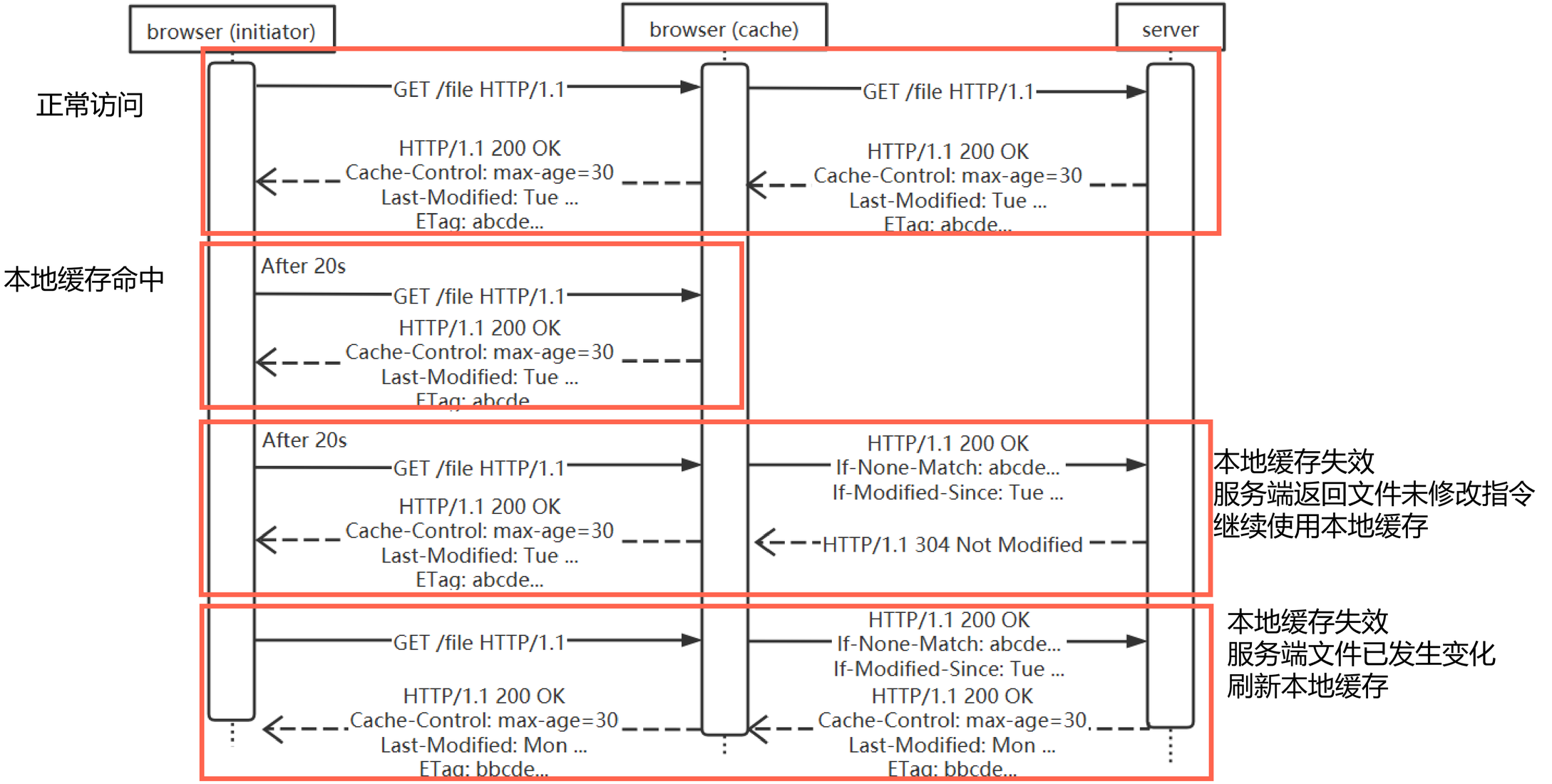
If-None-Match: <etag_value>

If-None-Match: <etag_value>, <etag_value>, ...

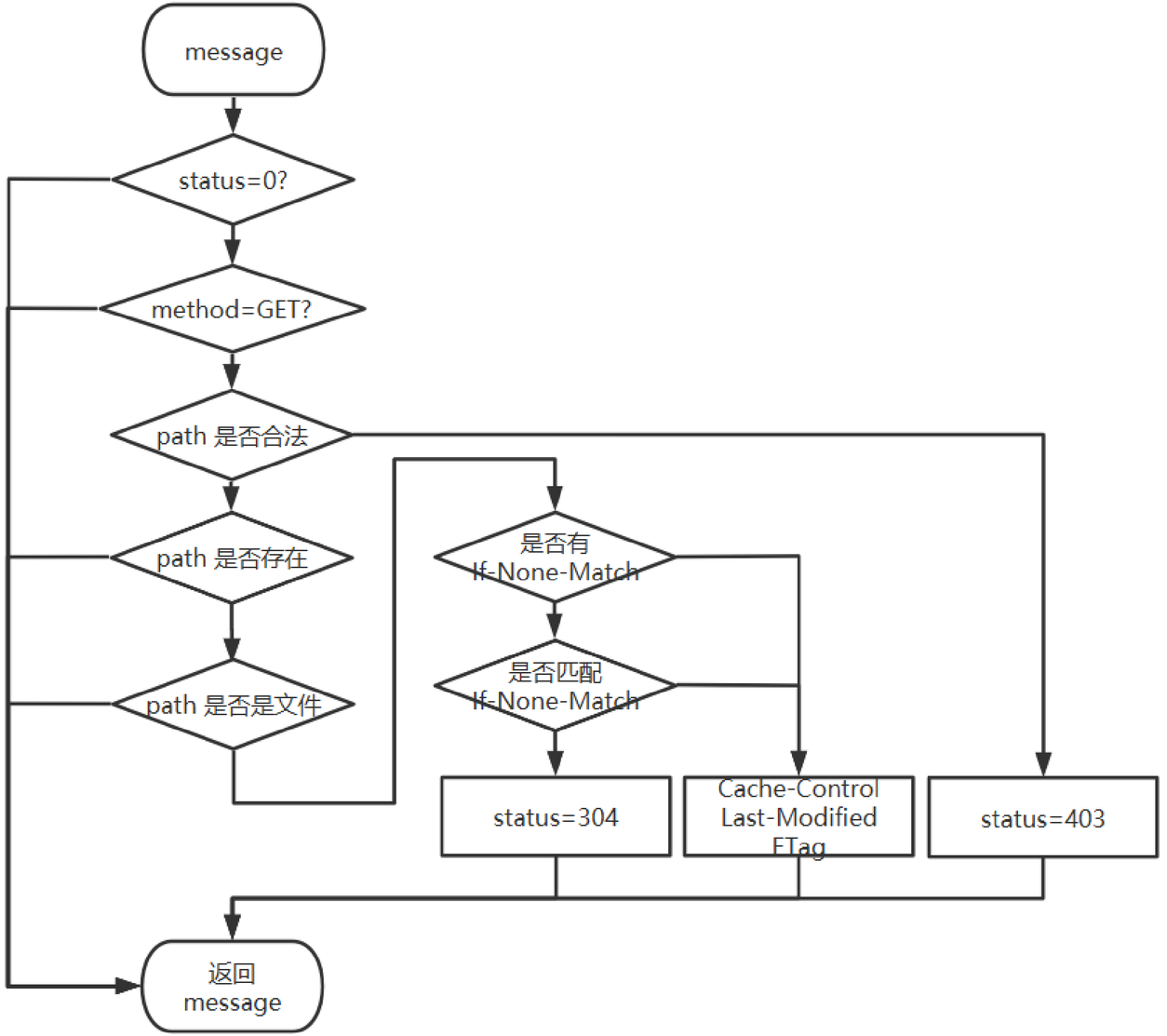
If-None-Match: *

本例只实现【用法 1】

浏览器缓存控制 (原理)


browser (initiator)
browser (cache)
server
正常访问
GET /file HTTP/1.1
GET /file HTTP/1.1
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Tue ...
ETag: abcde...
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Tue ...
ETag: abcde...
本地缓存命中
After 20s
GET /file HTTP/1.1
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Tue ...
ETag: abcde
After 20s
GET /file HTTP/1.1
HTTP/1.1 200 OK
If-None-Match: abcde...
If-Modified-Since: Tue ...
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Tue ...
ETag: abcde...
HTTP/1.1 304 Not Modified
本地缓存失效
服务端返回文件未修改指令
继续使用本地缓存
GET /file HTTP/1.1
HTTP/1.1 200 OK
If-None-Match: abcde...
If-Modified-Since: Tue ...
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Mon ...
ETag: bbcde...
HTTP/1.1 200 OK
Cache-Control: max-age=30
Last-Modified: Mon ...
ETag: bbcde...
本地缓存失效
服务端文件已发生变化
刷新本地缓存

浏览器缓存控制



新知识点

- Cache-Control
- ETag
- If-None-Match
- Last-Modified
- Status Code: 304

作业和练习

1. 根据本次课程中的时序图和流程图完成 HTTP Server 的 缓存、CORS、鉴权 和 Preflight 模块

2. 启动完成的 HTTP Server，并使用浏览器访问验证所有模块功能

扩展和提高

1. 阅读和学习 MDN HTTP 部分的参考，深入了解 HTTP 协议
   链接：https://developer.mozilla.org/en-US/docs/Web/HTTP

2. 深入理解 HTTP 协议，尝试为自己的 HTTP Server 增加新的特性

3. 可以参考的特性如下：

- 扩展 MIME，根据文件扩展名返回不同的 Content-Type
- 为鉴权模块的 Cookie 增加有效时间
- 服务端缓存验证在仅有 ETag 验证的基础的新增最后修改时间验证
- … 你的脑洞

THANKS