HTTP Accept 头驱动的内容协商机制

黄梓淳

Aug 26, 2026

在 Web 世界里，同一个资源往往需要以多种形式呈现给不同客户端。浏览器可能希望得到 HTML，移动应用可能期待 JSON，爬虫则可能只关心纯文本。这种「同一 URI、多种表示」的需求，催生了 HTTP 内容协商机制。内容协商分为主动协商与被动协商两种形式。主动协商由服务端根据请求头自动选择最合适的表示，被动协商则让客户端在多个可选响应中二次选择。本文聚焦主动协商中扮演核心角色的 Accept 系列请求头，剖析其语法、实现策略与工程实践，并探讨性能、安全与未来演进。

1　HTTP 内容协商基础

RFC 7231 明确了主动协商的流程：服务端在收到请求后，先解析 Accept、Accept-Language、Accept-Encoding 等头字段，再结合自身能力与资源可用形式，选出最优表示并返回 200 响应，同时在响应头中携带 Content-Type、Content-Language、Content-Encoding 以告知客户端实际使用的表示。Vary 响应头则用于告知缓存系统哪些请求头会影响响应结果，从而避免错误命中。

Accept 头声明客户端可接受的 MIME 类型及其优先级；Accept-Language 声明自然语言偏好；Accept-Encoding 则用于表达对 gzip、Brotli 等压缩方式的支持。Accept-Charset 已在现代实践中被废弃，因为 UTF-8 已成事实标准。服务端返回的 Content-* 系列头与请求中的 Accept 系列头形成镜像，共同构成完整的协商闭环。

2　Accept 头的完整解析

Accept 头的语法遵循「媒体范围」列表，每个范围可附加 q 参数表示质量因子。典型示例 Accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, /;q=0.8 表达了客户端对 HTML、XHTML、XML、WebP 图片以及任意类型的偏好顺序。质量因子取值范围为 0 到 1，默认 1。RFC 7231 第 5.3.2 节规定：当多个媒体范围匹配同一资源表示时，q 值最高者胜出；若 q 值相同，则按出现顺序决定。

星号通配符 / 表示接受任意类型，image/* 表示接受任意图片子类型。参数如 profile= 或 charset= 可进一步约束模式，例如 Accept: application/ld+json;profile=https://schema.org/ 用于请求符合 Schema.org 的 JSON-LD。浏览器通常发送包含多种现代格式的 Accept 列表，而 API 客户端往往只声明 application/json；爬虫则可能只发送 / 以获取最通用的文本形式。

3 服务端实现策略

服务端解析 Accept 头一般遵循「拆分—排序—过滤—打分—选择」五步。算法先将逗号分隔的媒体范围拆成数组，按 q 值降序排列，再逐一与服务端支持的 MIME 类型做匹配。匹配成功后累加 q 值与特异度得分，最终得分最高者即为协商结果。边界情况包括：请求不携带 Accept 头时，默认使用服务端首选类型；q=0 表示显式拒绝；多个类型得分相同时可按服务端偏好顺序或响应大小决定。

以 Node.js 的 negotiator 库为例，代码 negotiator = require('negotiator'); available = ['text/html','application/json']; best = negotiator.mediaType(available) 完成了从请求头到最优类型的映射。Go 语言可用 github.com/julienschmidt/httprouter 结合 mime 包实现类似逻辑。Spring 框架则通过 ContentNegotiationManager 统一管理，在 WebMvcConfigurer 中重写 configureContentNegotiation 方法即可声明扩展名、参数或 Accept 头的优先级。

与路由框架集成时，Express 可将协商逻辑封装为中间件，放在路由之前执行；Gin 在 Go 中可通过 c.Negotiate 系列方法直接返回 HTML 或 JSON；Spring MVC 则依靠 @ResponseBody 与 HttpMessageConverter 自动选择序列化器。无论哪种实现，缓存系统都需要看到 Vary: Accept，否则可能把 JSON 响应错误地提供给请求 HTML 的客户端。

4 实战案例

多语言文档站通常同时依赖 Accept-Language 与 Accept 头。服务端可先按 Accept-Language 确定语言，再按 Accept 选择 HTML 或 JSON 格式的 API 响应。URL 设计上，子路径方案 /zh-cn/docs 直观但需维护多套路由；内容协商方案 URI 唯一但缓存碎片化严重；Cookie 方案则在无状态场景下难以被 CDN 理解。

同一资源同时提供 HTML 与 JSON 的 HATEOAS API，可通过 Accept: application/json;profile=https://example.com/hateoas 与 Accept: text/html 区分不同表示。profile 参数可映射到不同 JSON Schema，从而在不改变媒体类型的前提下实现版本演进。

渐进式图片场景中，服务端可检查 Accept 是否包含 image/webp 或 image/avif，若命中则由 Cloudflare 或对象存储的边缘函数实时转换并返回对应格式。GraphQL 端点通常只声明 application/json，但仍需处理 Accept: application/graphql-response+json 等实验性类型。

5 性能与可维护性

Accept 解析本身开销极低，但 Vary: Accept 会把不同 Accept 值的请求分散到不同缓存条目，导致公共资源缓存命中率下降。缓解方法包括：标准化客户端 Accept 列表、忽略无影响的参数、或在反向代理层对 Accept 做归一化后再转发。

Nginx 可通过 map 指令将 Accept 映射为简短标记，再结合 try_files 实现轻量协商；Cloudflare Workers 则能在边缘执行完整协商逻辑，减少源站压力；Varnish 的 VCL 语言同样支持基于 Accept 的分支缓存。无论哪种方案，均需记录协商决策日志以便 A/B 测试不同 q 值策略对用户体验的影响。

6 安全与隐私

Accept 头可与其他指纹信息组合，提升跨站追踪精度。研究表明，Accept + User-Agent + Accept-Language 三元组在未登录用户中具备较高唯一性。服务端应避免将敏感信息写入 Content-Type 参数，如将内部版本号或用户 ID 编码其中。拒绝服务方面，构造数千个媒体范围的 Accept 头可放大解析开销，需在网关层设置字段长度上限或使用流式解析器。

7 演进与替代方案

传统 Accept 头无法表达「只返回部分字段」或「字段级压缩」等细粒度需求。RFC 7240 引入 Prefer 与 Preference-Applied 头，可声明 return=minimal 或 respond-async 等偏好。链接头中的 profile 与 describes 参数则提供了更语义化的资源描述方式。

HTTP/3 与 QUIC 对内容协商的影响主要体现在 0-RTT 与连接复用层面，但协商算法本身保持不变。RFC 8942 定义的 Variants 与 Variant-Key 头让服务端在响应中预先声明资源变体，客户端可据此直接选择，避免二次协商。

8 最佳实践清单

始终返回 Vary: Accept，以确保缓存正确区分不同表示。提供 406 Not Acceptable 兜底响应，并在响应体中列出服务端支持的类型列表。Accept 解析逻辑必须幂等，避免产生副作用或触发写操作。公开文档中应明确声明支持的 MIME 类型及默认 q 值策略。最后，结合 ETag 或 Last-Modified 实现条件请求，可在客户端缓存有效时跳过协商，降低延迟。

内容协商是「一个 URI，多种表示」的基石，在可维护性、安全与性能之间持续权衡。随着语义 Web 与机器可读描述的演进，未来或将出现 Accept-Schema 等新头字段，让客户端能够声明对数据结构的精确期望。理解 Accept 机制的细节，有助于构建更优雅、更高效的 Web 服务。