浏览器中的文件传输技术

李睿远

May 22, 2026

浏览器作为最广泛使用的客户端平台，其文件传输能力已经从最初的简单文件选择控件，逐步演进为支撑复杂网络应用的核心技术。早期用户只能通过 input 元素完成有限的上传操作，而今天的需求已扩展到大文件传输、高速并发、点对点直连以及离线场景。文章旨在系统梳理从传统到现代的浏览器文件传输技术栈，探讨不同技术在实际场景中的选择策略、性能优化要点与潜在副作用，并展望未来的发展方向。

1 传统文件传输方法

传统文件上传主要依靠 input 元素配合 FormData 对象实现。开发者在 HTML 中声明 input 元素并设置 type 属性为 file，即可让用户从本地磁盘选取文件。选取后，脚本可将该文件对象附加到 FormData 实例，再通过 fetch 或 XMLHttpRequest 发送至服务端。整个过程使用标准的 multipart/form-data 编码，服务器端可按普通表单字段的方式接收文件流。

当需要限制文件类型或大小以提升用户体验时，可在 input 元素上添加 accept 属性或在 JavaScript 中进行二次校验。需要注意的是，accept 仅为提示性质，用户仍可手动选择任意文件，因此后端必须再次验证 MIME 类型和文件大小，以防止恶意上传。

2 现代文件传输方法

随着 Web 平台能力的扩展，现代浏览器提供了 File、Blob、ArrayBuffer 等原生对象，允许开发者以更细粒度的方式操控文件数据。通过 FileReader 可以将文件内容异步读取为文本、Data URL 或 Array Buffer，从而实现前端预处理或分片计算。Blob 对象则支持对已有二进制数据进行切片、合并或创建新对象，为大文件分片上传提供了基础。

分片上传的核心思路是将大文件拆分为若干固定大小的块，每块独立上传，服务端按顺序或并行方式重组。典型做法是先调用 slice 方法得到若干 Blob，再依次使用 fetch 发送。分片策略可显著降低单次请求超时风险，并支持断点续传：服务端记录已接收的分片编号，客户端仅重传缺失部分即可。

3 点对点传输

WebRTC 为浏览器间的点对点文件传输提供了新途径。建立连接前，双方需通过信令服务器交换 SDP 和 ICE 候选信息。连接成功后，可创建 RTCDataChannel 用于传输任意二进制数据。文件首先被读入 ArrayBuffer，随后按 MTU 大小分块发送；接收方按序重组后，通过 URL.createObjectURL 生成临时链接供用户下载。

这种架构消除了对中心化服务器的依赖，显著降低延迟与带宽成本。但 WebRTC 受 NAT 穿越、防火墙策略影

响，实际连通率并非百分之百，因此生产环境常保留信令服务器作为备选中转。

4 离线文件传输与存储

当网络不可用时，浏览器可借助 IndexedDB 或 Cache Storage 实现本地文件暂存。开发者先将文件转换为 Blob，再写入 IndexedDB 的对象存储；待网络恢复后，再从数据库读出并执行上传。Service Worker 可拦截网络请求，结合 Background Sync API，在恢复连接时自动重试失败的上传任务，实现近似原生的离线体验。需要注意，浏览器对存储配额有严格限制，超出后会抛出 QuotaExceededError。因此在写入前应通过 navigator.storage.estimate 查询可用空间，并对超大文件采用分片压缩策略以降低存储压力。

5 性能优化策略

大文件传输的性能瓶颈通常出现在网络与 CPU 两个维度。采用 HTTP/2 或 HTTP/3 多路复用可并行发送多个分片，减少队头阻塞。客户端可使用 Web Worker 将哈希计算、压缩等 CPU 密集型任务移出主线程，避免阻塞用户界面。服务端则可利用零拷贝技术，直接将内核缓冲区映射到用户空间，降低内存拷贝开销。

此外，合理设置分片大小至关重要。过小的分片会增加请求头与握手开销；过大的分片则易触发超时或内存溢出。实测表明，5 MiB 左右的分片在大多数场景下能取得较好的吞吐与稳定性平衡。

6 安全与隐私考量

文件传输涉及敏感数据，必须在传输与存储两端实施安全措施。客户端应强制使用 HTTPS，防止中间人窃听或篡改。上传前可对文件进行客户端哈希校验，服务端二次验证以检测传输损坏或恶意替换。浏览器同源策略限制跨域访问，需通过 CORS 头或 postMessage 配合 ArrayBuffer 传递实现受控跨域。

隐私层面，开发者应最小化收集文件元数据，仅在必要时请求 FileSystemHandle 权限，并及时释放不再使用的对象 URL，以避免内存泄漏与用户文件信息泄露。

7 未来方向

WebTransport 基于 QUIC 协议，为浏览器提供更低延迟、支持双向流的可靠与不可靠传输通道，有望进一步提升大文件传输体验。File System Access API 的普及将允许网页直接读写用户授权目录，实现近似原生应用的本地文件协作。结合 WebAssembly 的高性能编解码能力，浏览器文件传输正朝着零拷贝、端到端加密、边缘计算的方向持续演进。