|  |  |  |
| --- | --- | --- |
| **Feature** | **HTTP/1.1** | **HTTP/2** |
| **Resource Loading Optimization** | Requires inlining and spriting for resource bundling. | Allows for more granular optimization with individual resource loading, minimizing redundant transfers. |
| **Header Requests** | Sends separate header requests for each resource. | Combines multiple requests into a single header, reducing overhead and improving efficiency. |
| **Flow Control** | Lacks a standardized mechanism for flow control. | Implements flow control, allowing for better management of data transmission between client and server. |
| **Error Handling** | Errors in one resource request can block others. | Isolates errors to individual streams, preventing them from affecting the loading of other resources. |
| **TCP Connections** | Establishes multiple TCP connections per origin. | Uses a single TCP connection, reducing the number of connections and associated overhead. |
| **Backward Compatibility** | Maintains compatibility with older servers and clients. | Is backward-compatible, allowing for a gradual transition and support for older systems. |
| **Pipelining** | Supports pipelining, but it is often not fully utilized due to limitations. | Pipelining is obsolete, as multiplexing in HTTP/2 provides a more efficient alternative. |
| **Resource Prioritization** | Limited control over the priority of resource loading. | Allows for more granular control with weight and dependency settings for each resource. |
| **Round Trip Optimization** | Requires multiple round trips for parallel resource fetching. | Reduces the need for additional round trips, improving efficiency with concurrent loading. |
| **Header Size Limit** | Large header size due to redundancy and lack of compression. | Implements header compression, reducing header size and mitigating the impact of large headers. |
| **Cache Handling** | Has limitations on cache handling, leading to redundant requests. | Improves cache handling, minimizing redundant requests through more effective cache utilization. |
| **Response Prioritization** | Limited ability to prioritize critical responses. | Supports response prioritization, ensuring the timely loading of critical content. |
| **Header Duplication** | Headers are duplicated across multiple requests. | Minimizes header duplication through compression, conserving bandwidth. |
| **WebSocket Support** | Requires a separate WebSocket connection. | Shares the same connection with other resources, reducing the need for additional connections. |
| **Pushed Resource Control** | Lacks control over resources pushed by the server. | Provides client control over pushed resources, enhancing flexibility and efficiency. |
| **Idle Connections** | Maintains idle connections for extended periods. | Optimizes idle connections for resource efficiency and reduced server load. |
| **Resource Dependencies** | Does not explicitly handle dependencies between resources. | Supports dependency tracking, ensuring that resources are loaded in the correct order. |
| **Security Considerations** | Slightly less emphasis on securing non-HTTPS connections. | Encourages secure connections (HTTPS), with many implementations requiring TLS for added security. |
| **Request and Response Headers** | Requests and responses often contain redundant information. | Minimizes redundancy in headers through efficient compression, optimizing data transfer. |
| **HOL Blocking Resolution** | Relies on domain sharding or alternative workarounds. | Eliminates head-of-line blocking through multiplexing, enhancing performance. |