1. Difference between HTTP1.1 vs HTTP2

|  |  |  |
| --- | --- | --- |
| Aspect | HTTP1.1 | HTTP2 |
| 1)Multiplexing | Separate connections for each file, leading to slower loading times. | Uses multiplexing, allowing multiple files to be transferred over a single connection simultaneously, speeding up page loading. |
| 2)Header Compression | Headers are not compressed, leading to increased overhead and slower transmission. | Implements header compression, reducing redundant header data and improving performance. |
| 3)Server Push | Servers can only respond to client requests, leading to additional round trips for required resources. | Introduces server push, where servers can push resources to clients before they're requested, reducing latency and speeding up page loading. |
| 4)Binary Protocol | Uses plaintext for data exchange, which can be inefficient for parsing | Adopts a binary protocol, optimizing communication efficiency and reducing parsing overhead. |
| 5)Stream Prioritization | Requests are processed in the order they're received, without considering their importance. | Implements stream prioritization, allowing clients to specify the importance of each resource request, ensuring critical resources are fetched first. |
| 6)Connection Reuse | Requires opening and closing multiple connections for each file, leading to connection setup overhead. | Supports connection reuse, allowing multiple requests and responses to be exchanged over a single connection, reducing latency and resource consumption. |
| 7)Flow Control | Lacks explicit flow control mechanisms, potentially leading to congestion and degraded performance. | Implements flow control mechanisms, enabling clients and servers to regulate data transmission rates, preventing congestion and ensuring optimal performance. |
| 8)Server Load | Each file request can potentially tie up a server connection, limiting scalability. | Multiplexing and connection reuse reduce the number of connections needed, allowing servers to handle more requests efficiently. |
| 9)Header Fields | Headers are sent with each request and response, even if they remain unchanged. | Introduces header field compression, reducing redundant header transmission and conserving bandwidth. |
| 10)Adoption and Compatibility | Widely supported and compatible with most web servers and clients. | Requires support from both servers and clients, with considerations for compatibility and performance influencing adoption decisions. |

2.Write about objects and its internal representation in Javascript

1. **Object Literal Notation:** Objects in JavaScript can be created using literal notation, where key-value pairs are enclosed within curly braces **{}**. For example:

var person = { name: "John", age: 30 };

1. **Properties:** Objects in JavaScript are collections of key-value pairs, where keys are known as properties. Properties can hold various data types, including primitive values, functions, or other objects.
2. **Prototypes:** Each object in JavaScript has a prototype, which serves as a fallback mechanism for property access. If a property is not found in the object itself, JavaScript looks for it in the object's prototype chain.
3. **Internal Representation:** Internally, objects in JavaScript are typically implemented using a combination of hash tables and internal slots. Hash tables are used to store the properties of an object, allowing for efficient property lookup and access.
4. **Property Access:** Properties of an object can be accessed using dot notation (**object.property**) or bracket notation (**object["property"]**). Bracket notation is useful when the property name is dynamic or contains special characters.
5. **Object Methods:** Objects can contain methods, which are functions stored as properties. These methods can perform operations on the object's data or interact with other objects in the system.
6. **Dynamic Properties:** JavaScript objects are dynamic, meaning properties can be added, modified, or deleted at runtime. This flexibility allows for dynamic object behavior and is a fundamental aspect of JavaScript's dynamic nature.
7. **Object Identity:** Each object in JavaScript has a unique identity, even if their properties are identical. This allows for comparison of objects by reference, using operators like **===** and **!==**.
8. **Cloning and Copying:** Objects can be cloned or copied using various techniques, such as the spread operator (**{ ...obj }**), **Object.assign()**, or serialization/deserialization methods like **JSON.parse(JSON.stringify(obj))**.
9. **Object Serialization:** JavaScript objects can be serialized into JSON (JavaScript Object Notation) format using **JSON.stringify()**. This process converts JavaScript objects into strings, which can then be transmitted over the network or stored in databases. Deserialization can be done using **JSON.parse()** to convert JSON strings back into JavaScript objects.