|  |  |
| --- | --- |
| HTTP1.1 | HTTP2 |
| * Uses a single connection for each request/response, which means that multiple requests need multiple connections. | Introduces multiplexing, allowing multiple requests and responses to be sent in parallel over a single connection. |
| * Headers are not compressed, and each request and response must carry the full set of headers, leading to increased overhead. | Implements header compression, which significantly reduces the amount of data that needs to be transmitted |
| * Uses a plain text protocol, which is human-readable but can be inefficient in terms of parsing and transmitting data. | Adopts a binary protocol, which is more efficient for machines to parse and reduces the amount of data transmitted. |
| * Clients must explicitly request each resource, and the server cannot proactively push resources to the client. | Supports server push, allowing the server to push additional resources to the client before they are explicitly requested. |
| * Requires multiple connections for parallelism, leading to increased resource usage and potential latency. | Uses a single, multiplexed connection, reducing the need for multiple connections and improving resource utilization. |
| * No built-in mechanism for prioritizing requests, leading to potential inefficiencies. | Introduces stream prioritization, allowing clients to assign priority levels to different resources. |
| * Relies on the TCP layer for flow control. If there's congestion or slow delivery of data, it may lead to suboptimal performance. | With multiplexing and other optimizations, HTTP/2 minimizes the number of round-trips needed, reducing latency and improving overall performance. |

1. Write a blog on difference between HTTP1.1 (vs) HTTP2

|  |  |
| --- | --- |
| * Each new connection requires a "warm-up" period to reach its optimal performance, which can impact small or short-lived connections. | Multiplexing allows for more efficient use of connections, reducing the need for a warm-up period. |
| * Requires careful management of multiple connections, especially in scenarios with limited resources. | Reduces the need for multiple connections, simplifying connection management and resource allocation. |
| * Servers can't proactively push resources to the client without explicit requests. | Introduces the "Push Promise" feature, enabling servers to push resources to the client before they are requested, potentially reducing latency. |
| * Caching is primarily based on the entire resource, and if one part of the resource is updated, the entire resource needs to be re-fetched. | Supports more granular content-based caching due to multiplexing, allowing for more efficient cache utilization. |

2.Write a blog about objects and its internal representation in Javascript ..?

## Understanding Objects in JavaScript:

In JavaScript, an object is a complex data type that allows developers to store and organize data in key-value pairs. Unlike primitive data types such as numbers or strings, objects can hold a collection of values and functions, making them versatile and powerful for building complex applications.

### Creating Objects:

Objects in JavaScript can be created using various methods. The most common approach is through object literals:

// Object literal

const person = {

name: 'John Doe',

age: 30,

occupation: 'Web Developer',

greet: function() {

console.log(`Hello, I'm ${this.name}!`);

}

};

## Internal Representation of Objects:

### 1. Properties and Values:

Each property of an object is stored as a key-value pair in memory. When you access a property, the JavaScript engine looks up the corresponding value based on the provided key.

### 2. Prototype Chain:

JavaScript is a prototypal language, meaning objects can inherit properties and methods from other objects through a mechanism called the prototype chain. This chain forms a hierarchy of objects, allowing for a more organized and reusable code structure.

### 3.Hidden Classes:

JavaScript engines optimize object creation and property access using hidden classes. These classes define the shape of an object and help improve performance by reducing the time needed for property lookups.

### 4. Object Descriptors:

Each property of an object has an associated object descriptor, containing information about whether the property is writable, enumerable, or configurable. This allows developers to control the behavior of object properties.

### 5. Property Access Efficiency:

Accessing properties directly is more efficient than using dynamic property names or expressions. It helps JavaScript engines optimize property lookups, leading to faster code execution.

### 6. Minimizing Object Mutations:

Objects are mutable in JavaScript, but excessive mutations can lead to performance issues. When possible, prefer creating new objects or using immutable data structures to enhance code maintainability and performance.

### 7. Prototype Pollution Awareness:

Be cautious about unintended changes to the prototype of objects, as this can lead to unexpected behavior and bugs. Avoid manipulating built-in prototypes and consider using Object.create() for safe object inheritance.