1. **Write a blog on Difference between HTTP1.1 vs HTTP2**

**1.INTRODUCTION:**

**1.1 HTTP1.1**

In 1989, Tim Berners-Lee invented HTTP. HTTP/1.1 was its 1st standardized version that was available for use in the year 1997 for the end-users. HTTP/1 was known to have poor response time. With websites becoming more resource-intensive, the protocol was losing its efficiency. It progressively became essential to minimize latency and boost page load speeds.

Years later, the IETF, Google, Microsoft, and Facebook released the fully comprehensive and well-tested newer version of HTTP in 2015.

**1.2 HTTP2:**

|  |  |
| --- | --- |
| **HTTP1.1** | **HTTP2** |
| **Multiplexing:** |  |
| Utilizes a single request-response connection at a time, leading to a phenomenon known as head-of-line blocking, where one slow-loading resource can delay the loading of subsequent resources. | Introduces multiplexing, allowing multiple streams of data to be sent and received concurrently on a single connection. This significantly improves the efficiency of data transfer, reducing latency and enhancing page load times. |

HTTP/2 is the second version of HTTP with most of the shortcomings of its predecessor addressed in it. It has come with advancements in efficiency, speed, and security. Till the date, HTTP/2 is supported on almost all popular web browsers, such as Chrome, Firefox, Internet Explorer, and Safari.

HTTP/2 aims at simplifying, speeding up, and empowering the applications across the internet. To achieve the same, the protocol emphasizes on page load time, resource optimization, and round-trip time (RTT) reduction.

For resource-heavy pages, it supports gradual downloading on the user’s end to improve user experience.

**Exploring the Differences Between HTTP/1.1 and HTTP/2**



|  |  |
| --- | --- |
| **Header Compression:** | |
| Headers are not compressed, leading to increased overhead and slower data transmission. | Implements header compression, reducing the size of headers and minimizing the amount of data that needs to be transferred. This results in improved performance and reduced latency. |
| **Binary Protocol:** | |
| Uses a plain text protocol, which is human-readable but less efficient in terms of parsing and processing. | Adopts a binary protocol, which is more compact and efficient for both machines to process and for network transmission. This contributes to faster and more reliable communication. |
| **Resource Prioritization:** | |
| Lacks built-in support for explicit resource prioritization, often relying on techniques like domain sharding to parallelize resource loading. | Introduces the ability to assign priority to resources, allowing critical resources to be loaded first. This ensures a more efficient use of available network resources. |
| **Server Push:** | |
| Relies on the client to request each resource individually, leading to additional round trips for necessary assets. | Supports server push, enabling the server to proactively send resources to the client before they are requested. This can significantly reduce latency and improve page load times. |
| **Connection Handling:** | |
| Requires multiple connections for parallel downloads, leading to increased complexity. | Efficiently manages connections, reducing the need for multiple parallel connections and improving resource utilization. |
| **Error Handling:** | |
| Uses a single error response, impacting the entire connection. | Permits independent error handling for each stream, minimizing the impact on other resources. |
| **Security:** | |
| Security is often dependent on additional protocols like TLS. | Encourages the use of TLS by making it a requirement, enhancing overall security. |
| **Backward Compatibility:** | |
| Remains widely used, but with limitations in terms of performance and efficiency. | Designed to be backward-compatible, allowing for a smooth transition without breaking existing systems. |
| **Resource Optimization:** | |
| Faced challenges with loading numerous small resources due to its limitations. | Optimizes the loading of multiple resources, particularly beneficial for modern, complex web applications. |

A screenshot of a computer screen

Description automatically generated

**CONCLUSION:**

The transition from HTTP/1.1 to HTTP/2 marks a significant leap in the world of web protocols. With improvements in speed, efficiency, and security, HTTP/2 addresses many of the shortcomings of its predecessor, providing a more streamlined and enhanced web browsing experience for users and developers alike.

**2.WRITE A BLOG ABOUT OBJECTS AND ITS INTERNAL REPRESENTATION IN JAVASCRIPT:**

**INTRODUCTION:**

Objects, in JavaScript, is it’s most important data-type and forms the building blocks for modern JavaScript. These objects are quite different from JavaScript’s primitive data-types (Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex, and each object may contain any combination of these primitive data-types as well as reference data-types. An object is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don’t store the value.

Objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of “key: value” pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like updateAddress, updateName, etc.

**INTERNAL REPRESENTATION OF OBJECTS:**

Under the hood, JavaScript engines implement objects using various data structures. One common approach is using a combination of dictionaries (or hash maps) and hidden classes.

**1. DICTIONARIES**

Objects in JavaScript can be thought of as dictionaries where each property access involves a hash table lookup. This allows for flexible key-value pairs but might impact performance for frequent property access or manipulation.

**2. HIDDEN CLASSES (OR SHAPES)**

JavaScript engines employ hidden classes to optimize property access. When an object is created, a hidden class is associated with it. When properties are added or removed, the hidden class might change, affecting how the object is optimized internally.

**OBJECT REPRESENTATION AND PERFORMANCE**

Understanding object representation is crucial for writing efficient JavaScript code. Property access and manipulation can impact performance, especially in performance-sensitive applications.

**BEST PRACTICES:**

**Property Access:** Minimize property access within loops or critical sections for better performance.

**Object Reuse:** Reuse objects where possible to maintain consistent hidden classes and optimize performance.

**Avoid Dynamic Property Addition:** Dynamically adding properties might cause hidden class changes, impacting performance.

**CREATING OBJECTS IN JAVASCRIPT :**

One of easiest way to create a JavaScript object is object literal, simply define the property and values inside curly braces as shown below

let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};

**console.log(bike.name);**

# CREATE JAVASCRIPT OBJECT WITH CONSTRUCTOR

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavour as shown below,

function Vehicle(name, maker) {  
 this.name = name;  
 this.maker = maker;  
}  
let car1 = new Vehicle(’Fiesta’, 'Ford’);  
let car2 = new Vehicle(’Santa Fe’, 'Hyundai’)  
console.log(car1.name); //Output: Fiesta  
console.log(car2.name); //Output: Santa Fe

**CONCLUSION:**

Objects form the backbone of JavaScript, offering flexibility and versatility. The internal representation of objects using dictionaries and hidden classes highlights the complexities involved in optimizing JavaScript code for performance.

By understanding how JavaScript handles objects internally, developers can write more efficient code, improving performance and enhancing the overall user experience.

JavaScript's object representation is a rich topic, delving into the nuances of language design and performance optimization. Mastering these concepts empowers developers to create more efficient and scalable applications in JavaScript.