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

## 

## **Exploring the Difference Between HTTP/1.1 and HTTP/2**

The Hypertext Transfer Protocol, or HTTP, is the foundation of data communication on the World Wide Web. Over the years, HTTP has evolved to become more efficient and adaptable to the demands of modern web applications. HTTP/1.1 was the widely used version for many years, but with the advent of HTTP/2, significant improvements were introduced. This blog will explore the differences between HTTP/1.1 and HTTP/2, highlighting the advantages of the newer protocol.

## **HTTP/1.1: The Old Workhorse**

HTTP/1.1, first standardized in 1997, has served as the backbone of the web for more than two decades. It functions on a simple request-response model, where a client (usually a web browser) sends a request to a server, and the server responds with the requested resources (such as HTML, CSS, JavaScript, and images).

### **1. Head-of-Line Blocking**

One of the significant limitations of HTTP/1.1 is head-of-line blocking. In this protocol, if multiple resources are requested, they are fetched sequentially. If a resource higher in the queue experiences delay, it can block the loading of subsequent resources, slowing down the entire page load.

### **2. High Latency**

Another drawback is high latency. In HTTP/1.1, a new TCP connection is established for each request-response cycle. This incurs the overhead of connection setup and teardown for each resource, resulting in higher latency.

### **3. No Support for Server Push**

HTTP/1.1 lacks native support for server push. In server push, the server can preemptively send resources to the client without a prior request. This reduces the need for multiple round trips between the client and the server.

## **HTTP/2: A Modern Solution**

HTTP/2, introduced in 2015, was designed to address the limitations of HTTP/1.1. It retains compatibility with the older protocol but offers several performance improvements.

### **1. Multiplexing**

One of the most significant enhancements in HTTP/2 is multiplexing. Multiple requests and responses can be sent and received in parallel over a single TCP connection. This eliminates the head-of-line blocking issue, allowing for faster and more efficient loading of resources.

### **2. Header Compression**

HTTP/2 uses header compression to reduce overhead. In HTTP/1.1, headers are sent with each request and response, which can be substantial. HTTP/2 compresses these headers, reducing the data size and improving overall performance.

### **3. Server Push**

HTTP/2 supports server push, which enables the server to send resources that it anticipates the client will request. This can significantly reduce the number of round trips needed to load a webpage.

### **4. Prioritization**

HTTP/2 allows for the prioritization of requests. This means that critical resources can be designated as high-priority, ensuring that they are loaded before less critical ones.

## **Which Should You Choose?**

HTTP/2 is undoubtedly an improvement over HTTP/1.1, offering better performance and more efficient resource loading. If you have the flexibility to choose which protocol to use, HTTP/2 is the way to go.

However, it's important to note that not all websites and servers support HTTP/2. The good news is that modern web browsers typically support both HTTP/1.1 and HTTP/2, so they can negotiate and use the best available protocol when fetching resources from a server.

In conclusion, HTTP/2 represents a significant step forward in the world of web protocols. Its ability to multiplex requests, compress headers, and support server push makes it the better choice for faster and more efficient web communication. While HTTP/1.1 still lingers, HTTP/2 is the present and future of web performance optimization.

The choice between HTTP/1.1 and HTTP/2 is no longer a choice between two equal options; it's a choice between a legacy standard and a modern solution designed to meet the demands of today's web.

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

## **Objects and Their Internal Representation in JavaScript**

## JavaScript, one of the most popular programming languages, is renowned for its versatile and flexible approach to handling data. Central to JavaScript's data structure is the object, which serves as a cornerstone for organizing and manipulating data in the language. In this blog, we'll explore what objects are in JavaScript and delve into their internal representation.

## **Understanding JavaScript Objects**

In JavaScript, an object is a complex data type that allows you to store and manage collections of key-value pairs. This data structure is fundamental to JavaScript and is used for a wide range of purposes, including representing real-world entities, creating data structures, and implementing many core language features.

### **Creating Objects**

Objects can be created in several ways, but the most common is using the object literal notation:

javascript

Copy code

const person = {

name: "John Doe",

age: 30,

email: "john@example.com"

};

In this example, we've created an object named person with properties like name, age, and email. The properties are key-value pairs, with keys representing property names and values representing the associated data.

## **Internal Representation of JavaScript Objects**

While the syntax and usage of JavaScript objects are straightforward, understanding how they are internally represented sheds light on how objects work behind the scenes.

### **1. Property Bag Model**

JavaScript objects can be conceptualized as a "property bag." This means they are essentially collections of key-value pairs. Each property (key) maps to a value. This simplicity is at the heart of the language's flexibility, allowing you to add, modify, or delete properties dynamically.

### **2. Prototypal Inheritance**

In JavaScript, objects are not just standalone entities. They can also inherit properties and behaviors from other objects through a mechanism known as "prototypal inheritance." This allows you to create a chain of objects, with each object inheriting from another.

javascript

Copy code

const parent = { a: 1 };

const child = Object.create(parent);

child.b = 2;

console.log(child.a); // Output: 1

console.log(child.b); // Output: 2

In this example, child inherits the property a from parent through its prototype chain.

### **3. Object Descriptors**

In JavaScript, object properties have associated descriptors. These descriptors define the behavior of the property, including whether it is writable, enumerable, or configurable. You can manipulate these descriptors using methods like Object.defineProperty and Object.getOwnPropertyDescriptor.

javascript

Copy code

const obj = { name: "Alice" };

// Make the 'name' property read-only

Object.defineProperty(obj, "name", { writable: false });

obj.name = "Bob"; // This will have no effect in strict mode

### **4. Memory Management**

JavaScript engines, such as V8 (used in Chrome) or SpiderMonkey (used in Firefox), implement objects differently under the hood. These engines employ memory management techniques like garbage collection to reclaim memory occupied by objects that are no longer in use.

## **Conclusion**

JavaScript objects are a fundamental part of the language, allowing you to represent, store, and manipulate data effectively. Understanding their internal representation helps you make more informed decisions about object manipulation and memory management in your JavaScript applications. Whether you're creating simple key-value pairs or building complex data structures, objects provide the foundation for organizing and working with data in JavaScript.