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

**Introduction**

The Hypertext Transfer Protocol (HTTP) is the backbone of data exchange on the World Wide Web. It's responsible for delivering web pages, images, videos, and other resources from servers to clients (usually web browsers). HTTP has evolved over the years, with HTTP/1.1 and HTTP/2 being two major versions. In this blog post, we'll delve into the significant differences between HTTP/1.1 and HTTP/2, exploring how the latter has brought significant improvements to web performance and efficiency.

**HTTP/1.1: The Old Workhorse**

HTTP/1.1, introduced in 1997, has been the backbone of the web for more than two decades. While it served its purpose admirably, the changing landscape of the internet led to certain limitations becoming apparent:

**Resource Loading:** In HTTP/1.1, each resource request typically required a separate connection. This resulted in a phenomenon known as the "head-of-line blocking," where one slow-loading resource could block the loading of subsequent resources, slowing down the page load.

**Header Overhead:** HTTP/1.1 sent headers for each request and response, which introduced significant overhead, especially for small resources.

**Compression:** While HTTP/1.1 allowed for header compression (e.g., gzip), this was optional and not consistently implemented across the web.

**HTTP/2: The Evolution**

HTTP/2 was developed to address the shortcomings of HTTP/1.1 and to better suit the demands of modern web applications. Here are the key differences that make HTTP/2 stand out:

**Multiplexing:** Perhaps the most significant improvement in HTTP/2 is multiplexing. Instead of relying on a single connection for each resource, HTTP/2 allows multiple resources to be requested and delivered in parallel over a single connection. This eliminates the head-of-line blocking issue, significantly speeding up page loads.

**Header Compression:** HTTP/2 mandates header compression, reducing the overhead associated with sending headers for each request and response. This results in more efficient use of network resources.

**Binary Protocol**: Unlike the plain text protocol of HTTP/1.1, HTTP/2 uses a binary protocol, which is more efficient for both parsing and transmission. This leads to faster and more compact communication between the client and server.

**Server Push:** HTTP/2 introduces server push, allowing the server to proactively send resources to the client before they are explicitly requested. This can be a game-changer for web performance, as it reduces the number of round trips required to load a page.

**Stream Prioritization:** HTTP/2 introduces the concept of stream prioritization, allowing the client to specify the importance of different resources. This ensures that critical resources are loaded first, further improving perceived page load times.

**Backward Compatibility:** HTTP/2 is designed to be backward-compatible with HTTP/1.1. Servers and clients that do not support HTTP/2 can still communicate using the older protocol.

**Conclusion**

In summary, HTTP/2 represents a significant leap forward in terms of web performance and efficiency compared to HTTP/1.1. With features like multiplexing, header compression, server push, and stream prioritization, HTTP/2 is optimized for the demands of modern web applications. However, the adoption of HTTP/2 has been gradual, and not all websites and servers support it yet. As web developers and users, understanding these differences can help us appreciate the benefits of HTTP/2 and advocate for its broader adoption, ultimately leading to faster and more efficient web experiences for everyone.

**Write a blog about objects and its internal representation in JavaScript**

**Introduction**

JavaScript is a versatile and powerful programming language used extensively in web development and beyond. One of its fundamental data structures is the object. In this blog post, we'll dive deep into JavaScript objects, exploring their internal representation and how to work with them effectively.

**What is a JavaScript Object?**

At its core, a JavaScript object is an unordered collection of key-value pairs. Each key (also known as a property or member) is a string or symbol, and each value can be of any data type, including other objects. Objects in JavaScript are incredibly flexible and versatile, making them a fundamental part of the language.

**Internal Representation of Objects**

To understand how JavaScript objects are represented internally, let's break down their key components:

**Properties and Methods:** Objects have properties, which are essentially variables attached to the object. These properties can hold values of any data type, including other objects. Objects can also have methods, which are functions associated with the object.

**Prototypes:** JavaScript uses a prototype-based inheritance model. Each object has an associated prototype object, which acts as a fallback for property and method lookups. If an object doesn't have a specific property or method, JavaScript will search for it in its prototype chain.

**Hidden Classes:** JavaScript engines use hidden classes (also called shapes or maps) to optimize property access. When you add properties to an object, the JavaScript engine creates a hidden class for that object's structure. This speeds up property access, as the engine can optimize memory and lookups based on the hidden class.

**Property Descriptors:** Each property in a JavaScript object has a property descriptor that defines its behavior. This descriptor includes attributes like whether the property is writable, enumerable, or configurable.

Creating Objects in JavaScript

There are several ways to create objects in JavaScript:

**Object Literal:**

const person = {

name: "John",

age: 30,

};

**Constructor Functions:**

function Person(name, age) {

this.name = name;

this.age = age;

}

const person = new Person("John", 30);

**Class Syntax (ES6):**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

}

const person = new Person("John", 30);

**Accessing Object Properties**

You can access object properties using dot notation or bracket notation:

const person = {

name: "John",

age: 30,

};

console.log(person.name); // Using dot notation

console.log(person["age"]); // Using bracket notation

**Modifying Objects**

Objects in JavaScript are mutable, meaning you can add, modify, or delete properties after creation:

person.city = "New York"; // Adding a property

person.age = 31; // Modifying a property

delete person.age; // Deleting a property

**Conclusion**

JavaScript's objects are at the heart of the language's versatility and power. Understanding their internal representation, how to create them, and how to work with their properties and methods is crucial for effective JavaScript programming. Whether you're building web applications, server-side code, or even desktop applications using frameworks like Electron, mastering JavaScript objects will empower you to create dynamic and sophisticated software solutions.

**Read about IP address, port, HTTP methods, MAC address**

**IP Address (Internet Protocol Address):**

An IP address is a unique numerical label assigned to each device connected to a computer network that uses the Internet Protocol for communication. IP addresses serve two main purposes:

**Identification:** They identify a specific device or host on a network, allowing data to be sent to and received from the correct destination.

Location Addressing: IP addresses are used for location addressing, helping routers and switches determine where to send data packets within a network.

There are two main versions of IP addresses in use today: IPv4 (Internet Protocol version 4) and IPv6 (Internet Protocol version 6). IPv4 addresses are 32-bit numerical addresses (e.g., 192.168.1.1), while IPv6 addresses are 128-bit hexadecimal addresses (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334). The transition from IPv4 to IPv6 is ongoing due to the depletion of available IPv4 addresses.

**Port:**

A port is a communication endpoint in a computer's operating system that is used to identify a specific process or service to which data can be sent. Ports are associated with IP addresses to enable network communication between devices.

Ports are categorized into two main types:

Well-Known Ports: These are reserved port numbers commonly used by well-known services and protocols. For example, HTTP typically uses port 80, and HTTPS uses port 443.

Dynamic (or Ephemeral) Ports: These are temporary port numbers used by client-side applications to establish connections with server-side services. They are typically assigned dynamically by the operating system.

The combination of an IP address and a port number forms a socket, which is the foundation for communication between devices over a network.

**HTTP Methods (Hypertext Transfer Protocol Methods):**

HTTP methods are commands or verbs that define the actions to be performed on a resource identified by a URL (Uniform Resource Locator). They dictate how the client and server should interact when making requests and responding to them. The most common HTTP methods include:

**GET:** Used to retrieve data from a specified resource. It should have no side effects on the server.

**POST:** Used to submit data to be processed to a specified resource. It can create new resources on the server or modify existing ones.

**PUT:** Used to update a resource or create it if it doesn't exist at a specific URL.

**DELETE:** Used to request the removal of a resource at a specific URL.

**PATCH:** Used to apply partial modifications to a resource.

These HTTP methods, along with others, enable the CRUD (Create, Read, Update, Delete) operations necessary for web applications to interact with servers and databases.

**MAC address (Media Access Control Address):**

A MAC address is a hardware address assigned to network interface controllers (NICs) for network communication on a physical network segment. MAC addresses are unique identifiers burned into the network hardware by the manufacturer.

Unlike IP addresses, which can change as devices move between networks, MAC addresses remain constant. They are used at the data link layer of the OSI model to ensure that data is delivered to the correct device within a local network. Routers use MAC addresses to make decisions about how to forward data between devices on different local networks.

A typical MAC address is represented as a sequence of six pairs of hexadecimal digits, such as "00:1A:2B:3C:4D:5E."

Understanding these fundamental concepts is essential for anyone working with computer networks, web development, or any field involving networked devices and services.