1. **Write a blog on the difference between HTTP1.1 and HTTP2.**

**Introduction:**

HTTP stands for hypertext transfer protocol & it is used in client-server communication. By using HTTP user sends the request to the server & the server sends the response to the user. There are several stages of development of HTTP but we will focus mainly on HTTP/1.1 which was created in 1997 & the new one is HTTP/2 which was created in 2015.

| **HTTP/1.1** | **HTTP/2** |
| --- | --- |
| It works in the textual format. | It works on the binary protocol. |
| There is head-of-line blocking that blocks all the requests behind it until it doesn’t get its all resources. | It allows multiplexing so one TCP connection is required for multiple requests. |
| It uses requests resource Inlining for use getting multiple pages | It uses a PUSH frame by the server that collects all multiple pages |
| It compresses data by itself. | It uses HPACK for data compression. |

The digital landscape is constantly evolving, and at the heart of this evolution are the protocols that govern communication between web browsers and servers. The transition from HTTP/1.1 to HTTP/2 is a crucial step in enhancing the performance and efficiency of web applications. In this blog post, we'll delve into the key distinctions between HTTP/1.1 and HTTP/2, shedding light on how these changes are reshaping the way we experience the web.

1. **Multiplexing:**
   * **HTTP/1.1:** Operates on a single, sequential request-response model. Each request is sent one at a time, and subsequent requests must wait for the completion of the previous ones. This can result in slower load times and inefficiencies.
   * **HTTP/2:** Introduces multiplexing, allowing multiple requests and responses to be sent concurrently over a single connection. This eliminates the bottleneck effect, significantly improving the speed and responsiveness of web applications.
2. **Header Compression:**
   * **HTTP/1.1:** Sends headers in plaintext for every request and response. This can lead to redundant data transmission and increased latency, especially when dealing with large headers.
   * **HTTP/2:** Implements header compression, which reduces the overhead associated with transmitting headers. By compressing headers, HTTP/2 minimizes data redundancy, resulting in faster load times and more efficient use of network resources.
3. **Binary Protocol:**
   * **HTTP/1.1:** Relies on a textual protocol that is human-readable but can be less efficient for machines to process. Parsing textual data can introduce unnecessary complexities.
   * **HTTP/2:** Adopts a binary protocol, streamlining the processing of information by machines. The binary format is more compact, making data transmission more efficient and reducing the potential for parsing errors.
4. **Server Push:**
   * **HTTP/1.1:** Lacks the capability for servers to proactively push resources to the client before they are requested. This can result in additional round trips and increased latency.
   * **HTTP/2:** Introduces server push, allowing servers to push resources to the client without waiting for explicit requests. This proactive approach enhances the loading speed of web pages by reducing the need for multiple round trips.
5. **Connection Handling:**
   * **HTTP/1.1:** Requires multiple connections to achieve parallelism, which can lead to increased resource consumption and greater complexity in managing connections.
   * **HTTP/2:** Enables the use of a single connection for parallelism, allowing multiple streams within that connection. This optimizes resource utilization, simplifies connection management, and contributes to overall performance improvements.

**Conclusion:**

The shift from HTTP/1.1 to HTTP/2 represents a significant advancement in web protocols, addressing the limitations of its predecessor and ushering in a new era of web performance. With features like multiplexing, header compression, a binary protocol, server push, and improved connection handling, HTTP/2 sets the stage for faster, more efficient, and more responsive web experiences. As the digital landscape continues to evolve, embracing these advancements becomes paramount for developers and businesses seeking to deliver optimal user experiences.

1. Write a blog about objects and their internal representation in JavaScript.

Objects, in JavaScript, are its most important data type and form 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 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 actually store the value.

Loosely speaking, 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, updateNam, etc.

**Objects and properties**

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object. Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot notation:

objectName.propertyName

Like all JavaScript variables, both the object name (which could be a normal variable) and the property name are case-sensitive. You can define a property by assigning it a value. For example, let’s create an object named myCar and give it properties named make, model, and year as follows:

var myCar = new Object();

myCar.make = 'Ford';

myCar.model = 'Mustang';

myCar.year = 1969;

Unassigned properties of an object are undefined (and not null).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see property accessors). Objects are sometimes called associative arrays, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the myCar object as follows:

myCar['make'] = 'Ford';

myCar['model'] = 'Mustang';

myCar['year'] = 1969;

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

// four variables are created and assigned in a single go,

// separated by commas

var myObj = new Object(),

str = 'myString',

rand = Math.random(),

obj = new Object();

myObj.type = 'Dot syntax';

myObj['date created'] = 'String with space';

myObj[str] = 'String value';

myObj[rand] = 'Random Number';

myObj[obj] = 'Object';

myObj[''] = 'Even an empty string';console.log(myObj);

You can also access properties by using a string value that is stored in a variable:

var propertyName = 'make';

myCar[propertyName] = 'Ford';propertyName = 'model';

myCar[propertyName] = 'Mustang';

You can use the bracket notation with for...in to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:

function showProps(obj, objName) {

var result = ``;

for (var i in obj) {

// obj.hasOwnProperty() is used to filter out properties from the object's prototype chain

if (obj.hasOwnProperty(i)) {

result += `${objName}.${i} = ${obj[i]}\n`;

}

}

return result;

}

So, the function call showProps(myCar, "myCar") would return the following:

myCar.make = Ford

myCar.model = Mustang

myCar.year = 1969

**Creating Objects in JavaScript:**

**Create JavaScript Object with Object Literal**

One of the 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'};

**Create JavaScript Object with Constructor**

Constructor is nothing but a function and with the help of a new keyword, the constructor function allows to creation of multiple objects of the same flavor 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

**Using the JavaScript Keyword new**

The following example also creates a new JavaScript object with four properties:

Example

var person = new Object();

person.firstName = “John”;

person.lastName = “Doe”;

person.age = 50;

person.eyeColor = “blue”;

**Using the Object.create method**

Objects can also be created using the Object.create() method. This method can be very useful, because it allows you to choose the prototype object for the object you want to create, without having to define a constructor function.

// Animal properties and method encapsulation

var Animal = {

type: 'Invertebrates', // Default value of properties

displayType: function () { // Method which will display type of Animal

console.log(this.type);

}

};

// Create a new animal type called animal1

var animal1 = Object.create(Animal);

animal1.displayType(); // Output:Invertebrates

// Create new animal type called Fishes

var fish = Object.create(Animal);

fish.type = 'Fishes';

fish.displayType();

// Output:Fishes

1. **Write about IP address, port, HTTP methods, MAC address**

In the intricate world of computer networking, several fundamental concepts play pivotal roles in facilitating communication between devices. IP addresses, ports, HTTP methods, and MAC addresses are among these essentials. In this blog post, we'll unravel the significance of each and how they contribute to the seamless functioning of the digital realm.

1. **IP Addresses:**

Internet Protocol (IP) addresses are the backbone of network communication. They uniquely identify devices on a network, allowing them to send and receive data. There are two main types of IP addresses: IPv4 (32-bit) and IPv6 (128-bit). IPv4 addresses, such as "192.168.1.1," have been widely used, but due to the increasing number of connected devices, IPv6 is becoming more prevalent.

IP addresses are hierarchical, with different classes serving various purposes. They enable routers to route data packets across the vast network infrastructure, ensuring that information reaches its intended destination.

1. **Ports:**

While IP addresses identify devices, ports specify the application or service on a device. Ports act as endpoints for communication. A port number is a 16-bit unsigned integer, allowing for a range of 0 to 65535. Ports are categorized into three ranges: well-known (0-1023), registered (1024-49151), and dynamic or private (49152-65535).

For example, when you access a website, your browser communicates with the server using specific port numbers. HTTP typically uses port 80, while HTTPS uses port 443. Understanding port numbers is crucial for efficient data exchange between applications.

1. **HTTP Methods:**

Hypertext Transfer Protocol (HTTP) governs the communication between web browsers and servers. HTTP methods define the actions a client can request. Common methods include:

* GET: Retrieve data from the server.
* POST: Submit data to be processed to a specified resource.
* PUT: Update a resource on the server.
* DELETE: Remove a resource from the server.

These methods enable versatile interactions, allowing web applications to perform various operations based on user input and requirements.

1. **MAC Addresses:**

Media Access Control (MAC) addresses operate at the data link layer of the OSI model and are assigned to network interfaces for communication within a local network. MAC addresses are unique identifiers burned into the hardware, ensuring every device on a network has a distinct address.

While IP addresses facilitate global communication, MAC addresses are vital for local network operations. They are commonly displayed as six groups of two hexadecimal digits (e.g., 00:1A:2B:3C:4D:5E).

Conclusion:

In the intricate tapestry of networking, understanding the roles of IP addresses, ports, HTTP methods, and MAC addresses is essential for anyone delving into the world of web development or network administration. These concepts form the foundation of digital communication, enabling the seamless transfer of data across the vast expanse of the internet.

Top of Form