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

Ans: HTTP (Hypertext Transfer Protocol) is a fundamental protocol used for communication on the World Wide Web. As the web has evolved, so has the need for more efficient and faster communication between web servers and clients. In this blog post, we'll explore the key differences between two major versions of HTTP: HTTP/1.1 and HTTP/2.

**HTTP/1.1: The Traditional Approach**

1. Serial Request Processing:

In HTTP/1.1, only one request can be processed at a time over a single TCP connection.

Resources, such as images, scripts, and stylesheets, are requested sequentially, leading to potential delays and inefficiencies.

2. Header Overhead:

Each request and response carries its own set of headers, leading to redundant information being sent for each resource.

This redundancy increases the amount of data transmitted and contributes to slower page loading times.

3. Resource Inlining and Spriting:

Techniques like inlining small images or combining multiple images into sprites are employed to reduce the number of requests.

However, these workarounds come with their own set of challenges and may not be ideal for all scenarios.

4. Connection Limitation:

Browsers limit the number of simultaneous connections to a single domain, often resulting in the need for domain sharding or resource concatenation to overcome this limitation.

**HTTP/2: A Modernized Approach**

1. Multiplexing:

One of the most significant improvements in HTTP/2 is the introduction of multiplexing.

Multiple requests and responses can be sent concurrently over a single TCP connection, eliminating the need for resource prioritization or connection optimization.

2. Header Compression:

HTTP/2 uses header compression techniques like HPACK to reduce redundancy.

This leads to a more efficient use of bandwidth, as headers are only sent when there are actual changes.

3. Binary Framing:

HTTP/2 employs a binary framing mechanism, enabling more compact and efficient data transfer compared to the plain text format used in HTTP/1.1.

This binary framing contributes to faster parsing by both browsers and servers.

4. Server Push:

HTTP/2 introduces server push, allowing the server to proactively send resources to the client without waiting for explicit requests.

This can significantly improve page loading times by reducing the need for additional round-trip requests.

5. Stream Prioritization:

HTTP/2 enables stream prioritization, allowing more critical resources to be loaded first.

Prioritization helps ensure a better user experience by delivering essential content faster.

**Conclusion**

While HTTP/1.1 has served as the backbone of the World Wide Web for many years, HTTP/2 represents a substantial leap forward in terms of efficiency and speed. The introduction of multiplexing, header compression, and other optimizations in HTTP/2 addresses the limitations of its predecessor, resulting in a more responsive and faster web.

However, it's essential to note that the adoption of HTTP/2 doesn't render HTTP/1.1 obsolete. Many websites and servers still use HTTP/1.1, and both protocols continue to coexist. The choice between them often depends on factors such as server and browser support, as well as the specific requirements of the web application.

As the web continues to evolve, it's clear that the advancements brought by HTTP/2 are pivotal in enhancing the overall performance and user experience of web applications. Whether you're a developer, system administrator, or end-user, understanding these differences can provide insights into the technologies that power the web.

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

Ans: JavaScript, being a versatile and dynamic programming language, employs a wide range of data types to handle different kinds of information. One of the fundamental and powerful data types in JavaScript is the object. In this blog post, we'll explore the concept of objects, their internal representation, and how they play a crucial role in building complex applications.

Objects in JavaScript

**Definition:**

In JavaScript, an object is a composite data type that allows you to store and organize data using a key-value pair mechanism. Objects provide a flexible way to represent real-world entities and their properties.

Syntax:

// Creating an object using literal notation

const person = {

name: 'John',

age: 30,

profession: 'Developer'

};

**Key Characteristics:**

Key-Value Pairs: Objects consist of key-value pairs where keys are strings or symbols, and values can be any valid JavaScript data type.

Properties and Methods: Keys in an object are called properties, and values associated with those keys can be data or functions, which are then referred to as methods.

Dynamic Nature: Objects can be modified, extended, or altered during runtime, making them highly adaptable.

**Internal Representation of Objects**

Memory Allocation:

When an object is created, JavaScript allocates memory to store its properties and methods. Each property and method has its memory location, and the object itself is a reference to those locations.

Object Prototype:

JavaScript is a prototype-based language, and each object is linked to a prototype object from which it inherits properties. The prototype linkage forms a chain, allowing objects to inherit methods and properties from their prototypes.

Property Descriptors:

Each property in an object has an associated property descriptor, which defines attributes like whether the property is writable, enumerable, or configurable. These descriptors provide control over the behavior of the properties.

**Object Creation Patterns**

Literal Notation:

const person = {

name: 'Alice',

age: 25

};

Constructor Function:

function Person(name, age) {

this.name = name;

this.age = age;

}

const person = new Person('Bob', 30);

Object.create():

const personPrototype = {

greet: function() {

console.log(`Hello, my name is ${this.name}.`);

}

};

const person = Object.create(personPrototype);

person.name = 'Charlie';

person.age = 35;

**Objects and Modern JavaScript Applications**

Objects play a pivotal role in modern JavaScript applications, especially with the advent of ES6 and beyond. Concepts like classes, destructuring, and enhanced object literals further enhance the capabilities of working with objects.

// ES6 Class

class Animal {

constructor(name, species) {

this.name = name;

this.species = species;

}

makeSound() {

console.log('Some generic animal sound.');

}

}

// Creating an instance

const lion = new Animal('Simba', 'Lion');

lion.makeSound();

**Conclusion**

Understanding objects and their internal representation is essential for JavaScript developers. Objects provide a versatile way to structure and organize data, enabling the creation of complex applications. With various object creation patterns and modern JavaScript features, developers have powerful tools at their disposal for efficient and maintainable code.

As you delve deeper into JavaScript development, mastering the nuances of objects will empower you to build scalable and robust applications. Whether you're working with prototypes, constructor functions, or modern ES6 classes, the principles of objects remain a cornerstone in the JavaScript language.