**ACTIVITIES DAY – 1**

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

**HTTP 1.1:**

HTTP 1.1 loads resources one after the other, so if one resource cannot be loaded, it blocks all the other resources behind it.

**HTTP 2:**

 HTTP 2 can use a single TCP connection to send multiple streams of data at once so that no one resource blocks any other resource.



The Hypertext Transfer Protocol (HTTP) is the foundation of any data exchange on the Web, and it has undergone several revisions since its inception. HTTP/1.1, introduced in 1997, has been the workhorse of the web for decades. However, as the web evolved, the limitations of HTTP/1.1 became apparent, leading to the development and release of HTTP/2 in 2015. This blog post explores the key differences between HTTP/1.1 and HTTP/2 and how these changes impact web performance and user experience.

**1. Multiplexing and Concurrency**

**HTTP/1.1**

HTTP/1.1 allows only one request per TCP connection. This means that if multiple resources are needed (which is usually the case with web pages that include images, CSS files, JavaScript, etc.), multiple TCP connections must be opened. This can lead to significant overhead and inefficiency due to the latency involved in establishing multiple connections.

**HTTP/2**

HTTP/2 introduces multiplexing, which allows multiple requests and responses to be sent simultaneously over a single TCP connection. This reduces the latency and overhead associated with establishing multiple connections, leading to faster load times and more efficient use of network resources.

**2. Header Compression**

**HTTP/1.1**

HTTP/1.1 sends headers as plain text, and since headers are sent with every request, this can lead to significant redundancy and larger payloads, especially when many requests are being made to the same server.

**HTTP/2**

HTTP/2 uses HPACK header compression to reduce the size of header data. This significantly decreases the amount of data that needs to be transmitted, improving the efficiency and speed of data transfer.

**3. Server Push**

**HTTP/1.1**

In HTTP/1.1, the server can only respond to requests made by the client. This means that if a web page requires additional resources (like CSS or JavaScript files), the client must explicitly request each one, resulting in additional round trips and increased load times.

**HTTP/2**

HTTP/2 introduces server push, a feature that allows the server to send resources to the client proactively. If the server knows that certain resources will be needed, it can push them to the client before the client even requests them. This reduces the number of round trips required and speeds up the loading process.

**4. Stream Prioritization**

**HTTP/1.1**

HTTP/1.1 does not support prioritization of requests. All requests are treated equally, which can lead to inefficient use of bandwidth and delays in loading critical resources.

**HTTP/2**

HTTP/2 supports stream prioritization, allowing clients to specify the priority of different streams. This enables more important resources to be loaded first, improving the overall user experience by ensuring that critical content is available as soon as possible.

**5. Binary Protocol**

**HTTP/1.1**

HTTP/1.1 uses a text-based protocol for communication, which can be more prone to errors and less efficient in terms of parsing and processing.

**HTTP/2**

HTTP/2 uses a binary protocol, which is more compact and easier for machines to parse and process. This leads to faster and more efficient communication between the client and server.

**6. Connection Management**

**HTTP/1.1**

In HTTP/1.1, each request/response pair requires its own connection, leading to a high number of connections and increased overhead. Persistent connections were introduced to mitigate this, but they still have limitations.

**HTTP/2**

HTTP/2 improves connection management by allowing a single connection to handle multiple requests and responses concurrently. This reduces the overhead and complexity associated with managing multiple connections, leading to more efficient use of network resources.

**Conclusion**

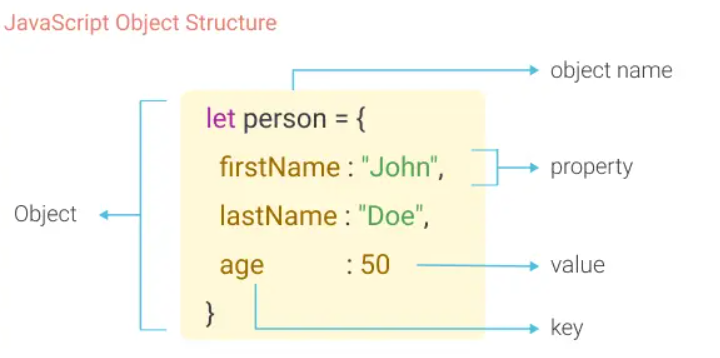
HTTP/2 represents a significant evolution from HTTP/1.1, addressing many of the performance and efficiency limitations of its predecessor. By introducing features like multiplexing, header compression, server push, stream prioritization, and a binary protocol, HTTP/2 offers faster load times, reduced latency, and a better overall user experience. As the web continues to grow and evolve, the adoption of HTTP/2 is likely to become increasingly important for ensuring optimal performance and efficiency.

Understanding these differences is crucial for web developers, network engineers, and anyone involved in the creation and maintenance of web applications, as it enables them to make informed decisions about optimizing their web services and improving user satisfaction.

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

Objects are important data types in javascript. Objects are different than primitive datatypes (i.e. number, string, boolean, etc.). Primitive data types contain one value, but Objects can hold many values in form of Key: value pair. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

Every object has some property associated with some value. These values can be accessed using these properties associated with them.



JavaScript is a powerful and versatile language, and objects are one of its core features. They allow for the creation of complex data structures, making it possible to manage and manipulate data in a flexible and efficient manner. In this blog, we'll explore what objects are in JavaScript, how they are represented internally, and some of the key features and behaviours that make them so essential.

**What is an Object in JavaScript?**

An object in JavaScript is a collection of properties, where each property is a key-value pair. The keys are strings (or symbols) that act as identifiers, and the values can be of any data type, including other objects, arrays, functions, and primitive values (such as numbers and strings). This key-value structure allows objects to represent complex data models.

Here is an example of a simple JavaScript object:

let person = {

name: "John Doe",

age: 30,

greet: function() {

console.log("Hello, my name is " + this.name);

}

};

In this example, person is an object with three properties: name, age, and greet.

**Internal Representation of Objects:**

Internally, JavaScript represents objects using a structure called a hash table or dictionary. This allows for efficient property lookup, addition, and deletion. Here are some key aspects of the internal representation:

1. Property Storage: Properties in an object are stored in a table-like structure that allows for quick access using the property keys. This structure is optimized to handle common operations like property retrieval, addition, and deletion efficiently.
2. Hidden Classes and Shapes: JavaScript engines, such as V8 (used in Chrome and Node.js), use hidden classes or shapes to optimize object property access. When an object is created, the engine assigns it a hidden class based on its properties. As properties are added or removed, the hidden class may change, but the engine uses these classes to streamline property access.
3. Prototype Chain: Objects in JavaScript can inherit properties and methods from other objects via the prototype chain. Each object has an internal link to another object called its prototype. When a property or method is accessed, the engine first looks at the objects own properties. If it doesn't find it there, it looks up the prototype chain until it finds the property or reaches the end of the chain (usually null).

**Example: (js)**

let animal = {

species: "Unknown",

makeSound: function() {

console.log("Some sound");

}

};

let dog = Object.create(animal);

dog.bark = function() {

console.log("Woof!");

};

console.log(dog.species); // Output: Unknown

dog.bark(); // Output: Woof!

In this example, dog inherits the species property and make Sound method from animal via the prototype chain.

**Key Features of JavaScript Objects**

1. Dynamic Nature: Objects in JavaScript are dynamic, meaning properties can be added, modified, or deleted at runtime. This flexibility is a cornerstone of JavaScript's dynamic behavior.
2. Methods: Functions can be assigned as properties of objects, making them methods. This allows for the encapsulation of behavior within objects.
3. Constructor Functions and Classes: Objects can be created using constructor functions or the class syntax, which provides a blueprint for creating objects with predefined properties and methods.

**Example: (js)**

function Person(name, age) {

this.name = name;

this.age = age;

}

Person.prototype.greet = function() {

console.log("Hello, my name is " + this.name);

};

let john = new Person("John Doe", 30);

john.greet(); // Output: Hello, my name is John Doe

With the introduction of ES6, the class syntax provides a more intuitive way to create objects and handle inheritance:

**Example: (js)**

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log("Hello, my name is " + this.name);

}

}

let jane = new Person("Jane Doe", 25);

jane.greet(); // Output: Hello, my name is Jane Doe

**Conclusion:**

Objects are a fundamental part of JavaScript, enabling the creation of complex and dynamic data structures. Their internal representation as hash tables and the use of hidden classes or shapes make property access and manipulation efficient. Understanding how objects work and how they are represented internally can help developers write more efficient and effective JavaScript code. As the cornerstone of object-oriented programming in JavaScript, mastering objects opens up a world of possibilities for creating sophisticated and high-performance web applications.