JavaScript - Day –1

Q1.) **Write a blog on Difference between HTTP1.1 vs HTTP2**

Answer:

In the realm of the internet, where every click and every request is powered by underlying protocols, the evolution of these protocols significantly impacts the performance and user experience. Two such pivotal protocols are HTTP1.1 and HTTP2, both instrumental in shaping the way we interact with the web. In this blog post, we'll delve into the key differences between HTTP1.1 and HTTP2, exploring how they've revolutionized web communication.

HTTP1.1 has been the backbone of the World Wide Web for over two decades. It introduced numerous features and optimizations over its predecessor, HTTP1.0, making web browsing faster and more efficient. However, as web technologies evolved, HTTP1.1 started showing signs of strain, particularly in handling the increasing complexity of modern websites.

Key Features of HTTP1.1:

1. **Persistent Connections**: HTTP1.1 introduced the concept of persistent connections, allowing multiple requests and responses to be sent over a single TCP connection. This helped reduce the overhead of establishing and tearing down connections for each resource request.

2. **Request Pipelining:** In theory, HTTP1.1 supported request pipelining, where multiple requests could be sent without waiting for each response. However, in practice, implementations varied, and issues like head-of-line blocking often negated the benefits.

3. **Header Compression**: HTTP1.1 didn't offer native header compression, resulting in redundant data being sent with each request and response. This overhead became more pronounced with the proliferation of mobile devices and limited bandwidth.

4. **Multiplexing:** Unlike HTTP2, HTTP1.1 lacked true multiplexing capabilities. Each request had to wait in line, leading to inefficient resource utilization and slower page load times, especially for sites with numerous assets.

HTTP2: The Modern Protocol

Recognizing the limitations of HTTP1.1, the Internet Engineering Task Force (IETF) introduced HTTP2 in 2015. Built upon the foundation of SPDY, an experimental protocol developed by Google, HTTP2 aimed to address the performance bottlenecks of its predecessor while maintaining compatibility with existing web infrastructure.

Key Features of HTTP2:

1. **Multiplexing**: HTTP2 introduced true multiplexing, allowing multiple requests and responses to be interleaved over a single TCP connection. This eliminated the head-of-line blocking problem and significantly improved resource utilization, leading to faster page loads.

2. **Binary Protocol**: HTTP2 employs a binary framing layer instead of plain text, reducing overhead and improving parsing efficiency. This streamlined format facilitates faster data transmission and reduces latency, particularly on high-latency connections.

3. **Header Compression**: HTTP2 features built-in header compression using the HPACK algorithm. By dynamically indexing and referencing previously sent headers, HTTP2 reduces redundant data transmission, optimizing network utilization and improving performance.

4. **Server Push**: One of the most significant enhancements of HTTP2 is server push, which allows servers to proactively send resources to the client before they are explicitly requested. This capability enables more efficient resource loading and faster page rendering, especially for assets associated with the initial request.

Scenario: Consider a webpage containing HTML, CSS, JavaScript, and several images.

HTTP/1.1:

- **Request**: The browser sends a request to the server for the HTML document.
- **Response**: The server responds with the HTML document.
- **Parsing**: The browser parses the HTML and identifies additional resources (CSS, JavaScript, images).
- **Requests for Resources**: For each identified resource, the browser sends individual requests to the server.
- **Sequential Loading**: Due to the lack of multiplexing, resources are fetched sequentially. Each subsequent request waits for the previous one to be completed before it can start.

- **Rendering**: Once all resources are fetched, the browser renders the page.

HTTP/2:

- **Request**: The browser sends a single request for the HTML document.
- **Response**: The server responds with the HTML document and initiates server push for associated resources (CSS, JavaScript, images).
- **Parallel Loading**: While receiving the HTML document, the browser also receives pushed resources in parallel. Additionally, it can request other necessary resources using the same TCP connection, benefiting from multiplexing.
- **Header Compression**: Headers are compressed, reducing overhead and improving efficiency.
- **Rendering**: With resources arriving more quickly and efficiently, the browser can render the page faster, resulting in a quicker user experience.

Conclusion:

HTTP2 represents a significant leap forward in web communication, addressing many of the shortcomings of HTTP1.1 and delivering faster, more efficient browsing experiences. By introducing features like multiplexing, binary framing, header compression, and server push, HTTP2 enhances performance, reduces latency, and improves resource utilization across the board. While the transition from HTTP1.1 to HTTP2 requires server and client-side support, the benefits it offers make it a compelling upgrade for any modern web application striving for optimal performance and user satisfaction.

Q2.) **Write a blog about objects and its internal representation in Javascript**

Answer:

In JavaScript, objects are collections of key-value pairs where keys are strings (or symbols) and values can be any data type, including other objects, functions, arrays, and primitives like numbers and strings. Objects in JavaScript are dynamic and flexible, allowing properties to be added, modified, or removed at runtime.

Internal Representation:

Internally, JavaScript engines use various techniques to represent objects efficiently. One common approach is using hash tables (also known as hash maps or dictionaries) to store object properties. In a hash table, each property's key is hashed to generate an index, allowing for fast access to properties.

Property Lookup:

When accessing or modifying properties of an object, JavaScript engines perform a property lookup process. This involves traversing the object's internal property map to find the corresponding property based on its key. If the property is found, its value is returned or updated; otherwise, undefined is returned.

Object Manipulation:

JavaScript provides several ways to manipulate objects, allowing developers to create, modify, and interact with objects dynamically.

Creating Objects:

Objects can be created using object literals, constructors, or by instantiating classes in modern JavaScript.

```
// Object literal
const person = {
  name: "John",
  age: 30
};

// Constructor
const car = new Object();
car.make = "Toyota";
car.model = "Camry";
```

Adding and Modifying Properties:

Properties can be added or modified using dot notation or square bracket notation.

```
person.gender = "Male"; // Dot notation
car["year"] = 2022; // Square bracket notation
```

Removing Properties:

Properties can be removed using the delete operator.

```
delete person.age;
```

Conclusion:

Understanding the internal representation of objects in JavaScript provides insights into how JavaScript engines manage and optimize object manipulation. By leveraging this knowledge, developers can write more efficient and performant code when working with objects. Whether it's creating, accessing, modifying, or deleting properties, having a solid understanding of JavaScript objects' internals empowers developers to build robust and scalable applications effectively.

Q3.) **Codekata practice**

Answer: Practicing javascript coding questions on a daily basis. Earned 1781 Guvi points currently.

Q4.) Read about IP address, port, HTTP methods, MAC address

Answer: Will revise the above topics.