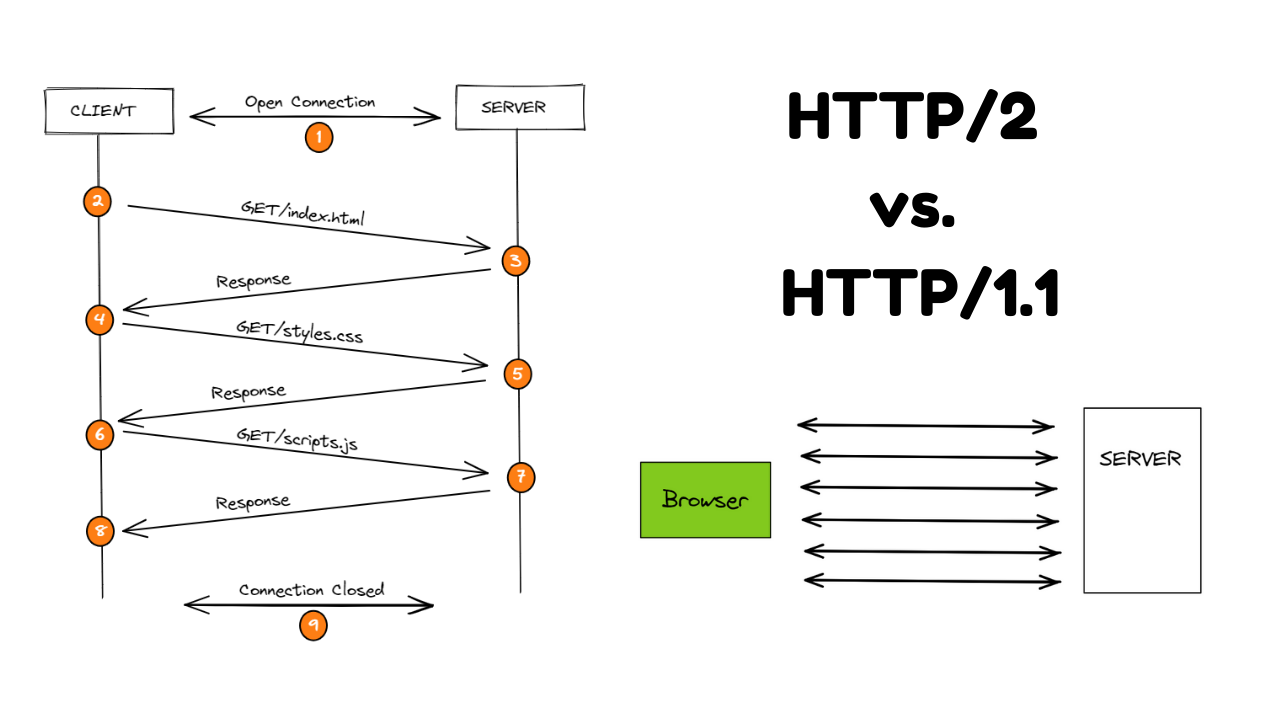
**Q.no: 1) Write a blog on Difference between HTTP1.1 vs HTTP2**

* **HTTP – Hypertext transfer protocol**



Hypertext Transfer Protocol (HTTP) is an application protocol that is, currently, **the foundation** of data communication for the World Wide Web.

**HTTP is based on** the Client/Server model. Client/Server model can be explained as two computers, Client (receiver of service) and Server (provider of service) that are communicating via requests and responses.

A simple and abstract example would be a **restaurant guest and a waiter**. The guest (**Client**) asks (**sends** **request**) waiter (**Server**) for a meal, then the waiter gets the meal from the restaurant chef (**your application logic**) and brings the meal to the guest.

| **Aspect** | **HTTP/1.1** | **HTTP/2** |
| --- | --- | --- |
| **Request/Response** | Uses multiple connections for parallel requests and responses. | Utilizes a single, multiplexed connection for parallel requests and responses. |
| **Header Compression** | Headers are not compressed, leading to high overhead. | Employs header compression (HPACK) to reduce overhead. |
| **Binary Protocol** | Text-based protocol, which is human-readable but less efficient. | Binary protocol, optimized for machines, enhancing efficiency. |
| **Prioritization** | Lacks built-in support for request prioritization. | Includes native support for prioritization of requests. |
| **Concurrency** | Limited parallelism, with a maximum of six connections per domain. | High level of concurrency, as multiple requests can be processed simultaneously over a single connection. |
| **Latency Reduction** | High latency due to head-of-line blocking. | Lowers latency by reducing head-of-line blocking through multiplexing. |
| **Server Push** | No support for server push. | Supports server push, allowing the server to send resources to the client proactively. |
| **Error Handling** | Errors can block subsequent requests. | Errors are isolated and do not impact other requests. |
| **TLS Requirement** | Not mandatory, but recommended. | Encourages the use of TLS (Transport Layer Security). |
| **Backward Compatibility** | Compatible with older browsers and systems. | Requires modern browsers and systems to fully benefit from its features. |
| **Implementation Complexity** | Simpler to implement and debug. | More complex due to multiplexing and header compression. |
| **Adoption** | Widespread and well-supported across various platforms. | Gaining adoption, but not universally supported by older systems. |

**HTTP/1.1** was a significant improvement over its predecessor, HTTP/1.0, but it had its limitations. One of the primary issues was that it required multiple connections to handle parallel requests and responses, which led to inefficiencies and increased latency. Moreover, the lack of header compression resulted in substantial overhead.

**HTTP/2**, on the other hand, addressed these limitations by introducing a binary protocol, header compression, and multiplexing, allowing multiple requests and responses to share a single connection. This greatly improved performance, reduced latency, and enhanced the overall user experience. Additionally, HTTP/2 introduced features like request prioritization and server push, which further optimized web communication.

Despite the advantages of HTTP/2, HTTP/1.1 is still in use today due to its compatibility with older systems and browsers. However, as the web continues to evolve, HTTP/2 is becoming the new standard, especially for websites and applications that prioritize speed and efficiency.

In conclusion, HTTP/2 is a significant improvement over HTTP/1.1 in terms of performance, efficiency, and user experience. While the transition to HTTP/2 may require some adjustments, its benefits make it a compelling choice for modern web development.

**Q.no: 2). Write a blog about objects and its internal representation in javascript**

**Title**: Objects and Their Internal Representation in JavaScript

**Introduction**:

JavaScript, one of the most widely used programming languages for web development, relies heavily on objects. In fact, nearly everything in JavaScript is an object or can be treated as an object. Understanding how objects are represented internally in JavaScript is fundamental for any developer aiming to master the language. In this blog post, we will delve into the internal representation of objects in JavaScript to provide you with a clear understanding of how they work behind the scenes.

**JavaScript Objects: The Basics**

In JavaScript, an object is a composite data type that can hold various data types, such as strings, numbers, functions, and even other objects. Objects are collections of key-value pairs, where keys are strings (or Symbols in ES6+) and values can be of any data type. To create an object, you can use the curly braces **{}** or the **new Object()** constructor.

**Creating Objects in JavaScript:**

1. By object literal
2. By creating instance of Object directly (using new keyword)

**By object literal:**

The syntax of creating object using object literal is given below:

https://miro.medium.com/v2/resize:fit:655/1*BPfZK2NMr7-nsIXYyXNFew.png

Property and value is separated by colon(:)

Here's a simple example of creating an object:

**const person = {**

**firstName: "John",**

**lastName: "Doe",**

**age: 30,** };

In this example, **person** is an object with three properties: **firstName**, **lastName**, and **age**. These properties are stored internally in a specific way.

**Internal Representation of Objects**

JavaScript engines, such as V8 (used in Chrome) and SpiderMonkey (used in Firefox), have their own way of internally representing objects. However, we can discuss a common model that helps in understanding how objects are typically structured behind the scenes.

**Property Descriptor**

Each property of an object is represented by a property descriptor. This descriptor holds information about the property, including its value, whether it's writable, enumerable, and configurable.

* **Value**: This is where the actual data associated with the property is stored.
* **Writable**: If set to **true**, the property's value can be changed. If set to **false**, the value is read-only.
* **Enumerable**: If set to **true**, the property can be iterated over in a **for...in** loop.
* **Configurable**: If set to **true**, the property's attributes can be changed, and the property can be deleted.

Hidden Classes (for V8 Engine)

In V8, a popular JavaScript engine, objects are organized using hidden classes. These hidden classes define the structure and behavior of objects, making property access faster. When you create an object in JavaScript, V8 associates it with a hidden class. As you add properties to the object, V8 optimizes its internal representation to ensure faster property access.

Here's a simplified example:

**class Point {**

**constructor(x, y) {**

**this.x = x;**

**this.y = y;**

**}**

**}**

**const p1 = new Point(1, 2);**

**const p2 = new Point(3, 4);**

In this case, **p1** and **p2** will have the same hidden class because they share the same constructor and property structure.

**Conclusion**

Understanding how objects are internally represented in JavaScript is crucial for writing efficient and performant code. While JavaScript abstracts many complexities away from developers, having insight into how objects work behind the scenes can help you write more optimized code and troubleshoot performance issues when they arise.

Remember that JavaScript engines may differ in their internal representations, so it's essential to focus on writing clean, maintainable code while keeping performance considerations in mind.