**Difference between HTTP1.1 vs HTTP2**

**HTTP/1.1: The Old Standard**

HTTP/1.1, which has been around since 1999, was a major improvement over its predecessor, HTTP/1.0. However, as web usage and complexity grew, it started to show its limitations.

1. **Multiplexing**: In HTTP/1.1, browsers could only open a limited number of concurrent connections to a server (usually around 6), which meant that if a web page had numerous resources to load (e.g., images, scripts, stylesheets), it had to wait for these connections to complete sequentially. This resulted in slow page load times.
2. **Header Inefficiency**: HTTP/1.1 would send headers with each request and response. This redundancy increased the amount of data sent over the network, leading to higher latency and increased bandwidth usage.
3. **No Prioritization**: HTTP/1.1 treated all resources equally, even if some were more critical to rendering a page. This lack of prioritization could lead to inefficient loading, where non-essential resources delayed the rendering of a page.

**HTTP/2: The Modern Marvel**

HTTP/2 was developed to address the shortcomings of HTTP/1.1, bringing significant improvements in terms of performance and efficiency.

**Multiplexing:** HTTP/2 introduced multiplexing, which allows multiple requests and responses to be interleaved on a single connection. This means that a browser can request and receive multiple resources in parallel over a single connection, reducing latency and speeding up page load times.

**Header Compression:** In HTTP/2, header data is compressed, reducing the overhead associated with headers. This results in a significant reduction in the amount of data transmitted, improving overall efficiency.

**Server Push:** One of the most exciting features of HTTP/2 is server push. With server push, a server can proactively send resources to the client before the client requests them. For example, when a browser requests an HTML file, the server can also push associated CSS and JavaScript files, saving round-trip time and further improving load times.

**Stream Prioritization:** HTTP/2 introduces stream prioritization, allowing developers to specify the importance of different resources. This ensures that critical resources are loaded before less important ones, optimizing page rendering.

**Security:** While not exclusive to HTTP/2, there is a general push towards HTTPS in conjunction with HTTP/2. Most modern browsers only support HTTP/2 over HTTPS, which helps improve security on the web.

**Objects and its Internal Representation in JavaScript**

**Objects are fundamental to JavaScript, as they enable developers to organize and manipulate data efficiently.**

**Understanding Objects in JavaScript**

In JavaScript, objects are a composite data type, allowing you to store and manipulate collections of data. Unlike primitive data types such as numbers and strings, objects are more complex and versatile, as they can store various data types and even functions. Objects are everywhere in JavaScript, and they play a pivotal role in creating structured and dynamic applications.

**The Anatomy of Objects**

JavaScript objects consist of key-value pairs. Each key, also known as a property, is associated with a value, forming the basic structure of an object. These properties can store various data types, including other objects, arrays, functions, or primitive values like strings and numbers.

**Internal Representation of Objects**

To truly understand how objects work in JavaScript, it's essential to peek under the hood and explore their internal representation.

**Hash Tables and Objects**

JavaScript engines implement objects using a data structure called a hash table. A hash table is an associative array that stores key-value pairs. When you create an object, the JavaScript engine allocates memory for it and uses a hash table to manage its properties. These properties are stored as keys (property names) and values in the hash table.

**Property Access**

When you access a property of an object, JavaScript uses the property's name to calculate a hash code. This hash code is used to locate the property's value in the hash table efficiently. This mechanism allows for fast property access and manipulation, even for objects with many properties.

**Prototype Chain**

JavaScript objects also have a concept called the prototype chain. Every object in JavaScript is linked to a prototype object. If a property is not found on an object, JavaScript looks for it in the object's prototype and continues up the prototype chain until it either finds the property or reaches the end of the chain.

**Conclusion**

JavaScript objects are versatile and powerful data structures used to store and manipulate data in web applications. Internally, they are implemented as hash tables, enabling fast property access. Understanding how objects work behind the scenes, including their use of hash tables and prototype chains, is crucial for JavaScript developers to leverage their full potential. Whether you're building web pages, web apps, or server-side applications, objects are at the heart of JavaScript, making them a fundamental concept to master for any developer.

Top of Form