**A Comparative Analysis of HTTP 1.1 and HTTP 2**

**Introduction:**

The evolution of the World Wide Web has seen significant advancements in its underlying protocols, with HTTP (Hypertext Transfer Protocol) serving as the foundation for communication between clients and servers. Over time, the need for faster, more efficient data transfer led to the development of HTTP2, building upon the strengths and addressing the limitations of its predecessor, HTTP1.1. In this blog, we'll delve into the key differences between HTTP1.1 and HTTP2, exploring how each protocol shapes the modern web experience.

**The Traditional Backbone of the Web**

HTTP1.1 has been the workhorse of the internet for decades, facilitating the exchange of vast amounts of data between clients (such as web browsers) and servers. However, its design, optimized for an earlier era of the web, comes with inherent limitations that hinder performance in today's dynamic web landscape.

Head-of-Line Blocking: In HTTP1.1, only one request per connection can be processed at a time, leading to head-of-line blocking. This means that if a large resource, like an image or a script, is being transferred, other requests have to wait in line, causing delays in loading web pages.

**Multiple Connections:**

To work around the limitations of head-of-line blocking, browsers often open multiple parallel connections to the same server, allowing for concurrent downloads of resources. However, this approach can lead to inefficiencies, as each connection incurs its own overhead in terms of TCP handshakes and resource allocation.

**Uncompressed Headers:**

HTTP1.1 headers are sent with each request and response, and they are typically uncompressed. This can result in unnecessary overhead, especially when dealing with multiple requests or responses containing redundant header information.

**HTTP2: A Leap Forward in Web Performance**

Recognizing the shortcomings of HTTP1.1, HTTP2 was developed to provide a more efficient and responsive web experience. It introduces several key features and optimizations that address the limitations of its predecessor.

**Multiplexing:**

One of the most significant improvements in HTTP2 is multiplexing, which allows multiple requests and responses to be sent and received on the same connection simultaneously. This eliminates head-of-line blocking and optimizes the use of network resources, leading to faster and more efficient data transfer.

**Binary Framing Layer:**

HTTP2 introduces a binary framing layer that replaces the text-based format of HTTP1.1. By encoding data in binary format, HTTP2 reduces overhead and improves parsing efficiency, resulting in lower latency and higher throughput.

**Header Compression:**

HTTP2 employs header compression techniques to reduce the overhead associated with transmitting header information. This helps minimize the amount of data sent over the network, further improving performance, especially for websites with numerous small assets.

**Server Push:**

Another notable feature of HTTP2 is server push, which allows servers to proactively send resources to the client before they are requested. This can significantly reduce the number of round trips required to load a web page, further enhancing performance and responsiveness.

**Conclusion:**

In summary, while HTTP1.1 has served as the backbone of the web for many years, its limitations in terms of performance and efficiency have become increasingly apparent in today's web ecosystem. HTTP2 represents a significant leap forward, offering improvements such as multiplexing, binary framing, header compression, and server push, all of which contribute to a faster, more responsive web experience. As websites and web applications continue to evolve, HTTP2 stands poised to play a central role in shaping the future of the internet, delivering enhanced performance and efficiency for users and developers alike.

**OBJECTS ANS ITS INTERNAL REPRESENTATION IN JAVASCRIPT**

**Introduction:**

JavaScript, as one of the most widely used programming languages for web development, relies heavily on objects. Objects are fundamental to JavaScript's syntax and functionality, serving as the building blocks for data structures and providing a flexible means of organizing and manipulating data. In this blog, we'll explore the concept of objects in JavaScript, delving into their internal representation and shedding light on how they function under the hood.

**Understanding Objects in JavaScript:**

In JavaScript, objects are collections of key-value pairs, where each key is a string (or Symbol) and each value can be of any data type, including other objects, functions, arrays, and primitive values. Objects can be created using either object literals, constructor functions, or the class syntax introduced in ECMAScript 2015 (ES6).

**Javascript (code)**

// Object literal

const person = {

name: 'John',

age: 30,

greet() {

console.log(`Hello, my name is ${this.name} and I'm ${this.age} years old.`);

}

};

// Constructor function

function Person(name, age) {

this.name = name;

this.age = age;

}

// Class syntax

class Animal {

constructor(name) {

this.name = name;

}

}

**Internal Representation of Objects:**

Internally, JavaScript engines use various mechanisms to represent objects efficiently. While the exact details may vary between different engines (such as V8 in Chrome, SpiderMonkey in Firefox, and JavaScriptCore in Safari), the underlying principles remain similar.

**Object Properties and Prototypes:**

Each JavaScript object has an internal property called [[Prototype]], which references another object known as its prototype. If a property is not found on the object itself, the JavaScript engine looks up the prototype chain until it finds the property or reaches the end of the chain (the prototype of Object.prototype).

This mechanism enables inheritance and property delegation in JavaScript.

**Property Descriptors:**

Each property of an object is associated with a property descriptor, which contains information about the property's attributes such as value, writable, enumerable, and configurable. These attributes determine the behavior of the property and how it can be accessed and modified.

**Hidden Classes and Inline Caching**:

JavaScript engines optimize property access and method calls using techniques like hidden classes and inline caching. These optimizations improve performance by reducing the overhead of property lookup and method invocation.

**Memory Management:**

JavaScript engines employ garbage collection algorithms to manage the memory used by objects and reclaim memory no longer needed by unreachable objects. Modern garbage collectors, such as the generational garbage collector used in V8, are designed to minimize pauses and optimize memory usage.

**Conclusion:**

Objects are central to JavaScript programming, providing a versatile and powerful means of organizing and manipulating data. Understanding how objects are represented internally in JavaScript can help developers write more efficient and performant code. By grasping concepts such as prototypes, property descriptors, hidden classes, and memory management, developers can gain insight into the inner workings of JavaScript engines and leverage this knowledge to write faster and more optimized JavaScript applications.