M S BASKARAN - FSD - B52WD2 TAMIL

Javascript Day 1 Task

horizontal line

Task 1:

# HTTP/1.1 vs. HTTP/2

Introduction

The World Wide Web is built upon a complex network of protocols, and one of the most critical ones is the Hypertext Transfer Protocol (HTTP). It's the foundation for data communication on the web, enabling browsers to retrieve web pages from servers. Over the years, HTTP has undergone significant changes and improvements to keep up with the growing demands of the internet. In this blog, we will dive into the key differences between HTTP/1.1 and HTTP/2 and how this evolution has shaped the web we know today.

HTTP/1.1: The Classic Protocol

HTTP/1.1, the predecessor of HTTP/2, has been the workhorse of the internet for more than a decade. While it served the web well during its time, it did have its limitations.

1.Sequential Processing:

In HTTP/1.1, each request-response cycle was processed sequentially. If a page required multiple resources, such as images, scripts, and style sheets, the browser would send requests for these resources one at a time, leading to delays in page loading.

2.High Latency:

The protocol suffered from high latency due to the so-called "head-of-line blocking." If a single resource took longer to load, it could block the loading of other resources, even if they were readily available.

3.Redundant Headers:

HTTP/1.1 used textual headers for every request, which led to redundant data transmission. For each request, the same headers had to be sent again and again, causing additional overhead.

HTTP/2: A New Era

In response to the shortcomings of HTTP/1.1, HTTP/2 was introduced. This updated protocol brought about several significant changes, making web browsing faster and more efficient.

1.Multiplexing:

One of the most significant changes in HTTP/2 is the ability to send multiple requests and responses in parallel over a single TCP connection. This means that multiple resources can be fetched simultaneously, reducing latency and significantly speeding up page loading times.

2.Binary Protocol:

HTTP/2 uses a binary protocol as opposed to the textual one used in HTTP/1.1. This makes it more efficient and compact, as it reduces redundancy and minimizes the overhead associated with textual headers.

3.Header Compression:

HTTP/2 uses header compression, which reduces the overhead of sending headers for each request and response. This not only saves bandwidth but also speeds up communication.

4.Server Push:

HTTP/2 introduced server push, a feature that allows a server to proactively send resources to the client before they are requested. This can further reduce the number of round trips needed to load a web page, enhancing performance.

Key Differences Summarized

Here's a summary of the main differences between HTTP/1.1 and HTTP/2:

1.Multiplexing:

HTTP/2 allows multiple requests and responses to be sent in parallel over a single connection, reducing latency.

2.Binary Protocol:

HTTP/2 uses a binary protocol for more efficient and compact data transfer.

3.Header Compression:

Headers are compressed in HTTP/2, reducing redundancy and saving bandwidth.

4.Server Push:

HTTP/2 supports server push, enabling proactive resource delivery to the client.

Conclusion

The transition from HTTP/1.1 to HTTP/2 marked a significant step forward in web performance and efficiency. HTTP/2's ability to multiplex requests, use binary protocol, compress headers, and support server push has made the web faster, more responsive, and more suitable for modern web applications. However, it's important to note that the adoption of HTTP/2 is not uniform across the web, and the support for this protocol may vary among websites. As the internet continues to evolve, we can expect further improvements in web protocols to enhance our online experiences even more.

TASK 2:

# Understanding Objects and Their Internal Representation in JavaScript

Introduction:

JavaScript is an incredibly versatile and powerful programming language, known for its capacity to handle complex data structures. Objects are a fundamental concept in JavaScript, allowing developers to structure and manage data effectively. In this blog, we'll explore the concept of objects in JavaScript and delve into their internal representation.

What Are JavaScript Objects?

In JavaScript, objects are collections of key-value pairs. Each key, often referred to as a property, maps to a value. This flexible structure makes objects suitable for representing real-world entities, from simple data structures to complex entities. Objects are used extensively in the language, and you've likely encountered them when dealing with variables, functions, and libraries.

Internal Representation of Objects:

To understand the internal representation of objects in JavaScript, you need to grasp how they are stored in memory. JavaScript engines, which execute your code, use various data structures and optimizations to manage objects efficiently.

1.Property-Value Pairs:

The most fundamental part of an object's internal representation is its property-value pairs. These pairs are stored in an object's memory allocation. Properties serve as keys that map to values. For instance, consider an object representing a person:

```javascript

const person = {

name: 'John',

age: 30,

isEmployed: true

};

```

In this object, "name," "age," and "isEmployed" are properties, and their corresponding values are 'John,' 30, and true.

2.Property Descriptors:

Each property has associated property descriptors, which provide additional information about the property. Property descriptors include attributes such as whether a property is writable, enumerable, or configurable. These descriptors are part of an object's hidden internal structure and are managed by the JavaScript engine.

3.Hidden Classes and Shapes:

JavaScript engines use optimization techniques, such as hidden classes and shapes, to improve object access and memory management. These mechanisms determine how properties are stored in memory and are crucial for optimizing property access.

4.Prototypes:

Objects can also have a prototype, which is a reference to another object. When a property is accessed on an object, and the property doesn't exist on the object itself, JavaScript will look up the prototype chain to find the property in the object's prototype.

Object Creation:

JavaScript provides several ways to create objects:

1.Object Literal Notation:

As shown in the example above, you can create an object using curly braces and specifying property-value pairs.

2.Constructor Functions:

You can create objects using constructor functions. These functions act as blueprints for creating objects with shared properties and methods.

3.Object.create():

This method creates a new object with the specified prototype object.

Object Manipulation:

To interact with objects in JavaScript, you can use various methods and syntax, including dot notation (e.g., `object.property`) or bracket notation (e.g., `object['property']`) to access and manipulate properties.

Conclusion:

Objects are a fundamental concept in JavaScript, serving as the building blocks for structuring and managing data. Understanding their internal representation, including property-value pairs, descriptors, hidden classes, and prototypes, is crucial for writing efficient and performant JavaScript code. As you delve deeper into JavaScript development, mastering objects and their internal workings will become essential in creating robust and scalable applications.