HTTP/1.1 and HTTP/2 are two different versions of the HTTP (Hypertext Transfer Protocol), which is the protocol used for transmitting data over the internet. They have several key differences in terms of performance, features, and how they handle data. Here are some of the main differences between HTTP/1.1 and HTTP/2:

1. **Multiplexing**:
   - **HTTP/1.1**: In HTTP/1.1, requests and responses are processed sequentially. This means that if multiple resources (e.g., images, scripts, stylesheets) need to be loaded on a web page, the browser has to open multiple connections to the server, which can be inefficient and slow.
   - **HTTP/2**: HTTP/2 supports multiplexing, which allows multiple requests and responses to be sent and received in parallel over a single connection. This greatly improves the efficiency of data transfer and reduces latency, as it eliminates the need to open multiple connections.

2. **Header Compression**:
   - **HTTP/1.1**: Each HTTP request and response includes headers that provide metadata about the request or response. In HTTP/1.1, these headers are sent as plaintext, and the headers can be quite large, leading to increased overhead.
   - **HTTP/2**: HTTP/2 uses header compression techniques (HPACK) to reduce the size of headers, which helps minimize overhead and improve performance.

3. **Binary Protocol**:
   - **HTTP/1.1**: HTTP/1.1 is a text-based protocol, which means that both headers and data are transmitted as plain text. This can be less efficient in terms of parsing and processing.
   - **HTTP/2**: HTTP/2 is a binary protocol, which means that headers and data are encoded in binary format. Binary encoding is more efficient for both transmission and parsing by machines.

4. **Server Push**:
   - **HTTP/1.1**: HTTP/1.1 does not support server push. The client must explicitly request each resource it needs, even if the server knows that additional resources will be required.
   - **HTTP/2**: HTTP/2 supports server push, which allows the server to push resources to the client before the client requests them. This can reduce the number of round-trip requests required to load a web page and improve page load times.

5. **Prioritization**:
   - **HTTP/1.1**: In HTTP/1.1, there is no built-in mechanism for prioritizing requests. All requests are treated equally.
   - **HTTP/2**: HTTP/2 allows for request prioritization, so more important or critical resources can be given higher priority, ensuring faster delivery.

6. **Backward Compatibility**:
   - **HTTP/1.1**: HTTP/1.1 is widely supported and is compatible with older web servers and clients.
   - **HTTP/2**: HTTP/2 is designed to be backward compatible with HTTP/1.1. If a client or server does not support HTTP/2, they can still communicate using HTTP/1.1.

In summary, HTTP/2 offers significant performance improvements over HTTP/1.1, primarily due to features like multiplexing, header compression, binary encoding, server push, and request prioritization. These improvements make web pages load faster and more efficiently, which is crucial for modern web applications and websites.

Title: Demystifying Objects and Their Internal Representation in JavaScript

Introduction:
JavaScript is a versatile programming language known for its extensive use of objects. In this blog post, we will delve into the fascinating world of objects in JavaScript and explore how they are internally represented. Understanding the internals of objects is essential for writing efficient and performant code.

Objects: The Cornerstone of JavaScript

1. **What Are Objects?**
   - In JavaScript, objects are fundamental data structures used to store and organize data.
   - They are collections of key-value pairs, where keys (also called properties) are strings, and values can be of any data type.

2. **Creating Objects**
   - Objects can be created using object literals, constructors, or classes (introduced in ES6).
   - Object literals are the simplest and most commonly used way to create objects.

The Internal Representation of Objects

3. **Property Storage**
   - JavaScript engines use different techniques to store object properties. One common approach is using a hash table-like structure, where keys are hashed to access values quickly.

4. **Property Attributes**
   - Each property in JavaScript objects has associated attributes, such as writable, enumerable, and configurable, which determine their behavior.
   - These attributes affect property access, modification, and deletion.

5. **Primitive Wrapper Objects**
   - JavaScript has primitive data types (e.g., numbers, strings), and when you access their properties, temporary wrapper objects are created and discarded during the operation.

Object Prototypes and Inheritance

6. **Prototypal Inheritance**
   - Objects in JavaScript are linked together through a prototype chain.
   - When you access a property on an object, if it's not found, JavaScript looks up the prototype chain until it finds the property or reaches the end (Object.prototype).

7. **`__proto__` vs. `prototype`**
   - `__proto__` is an object's reference to its prototype, used for property lookup.
   - `prototype` is used by constructor functions to establish the prototype for objects created with `new`.

Object Cloning and Immutability

8. **Shallow vs. Deep Copy**
   - Cloning objects in JavaScript can be tricky. `Object.assign()` creates a shallow copy, while deep copying requires custom solutions.

9. **Immutability**
   - Achieving immutability can be essential for avoiding unintended side effects in JavaScript. Libraries like Immutable.js help with this.

Conclusion

Objects are the building blocks of JavaScript, and understanding their internal representation is crucial for writing efficient and bug-free code. Knowing how properties are stored, how prototypes work, and how to achieve immutability empowers JavaScript developers to harness the full potential of this versatile language. Explore and experiment with objects to become a JavaScript master.