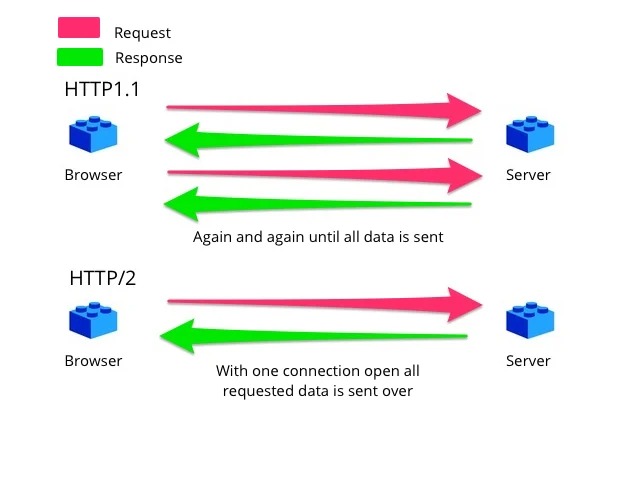
DIFFERENCE BETWEEN HTTP1.1 VS HTTP2

## **What is HTTP?**

* [HTTP](https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/) stands for hypertext transfer protocol, and it is the basis for almost all web applications.
* It is used to load webpages using hypertext links.
* HTTP (Hypertext Transfer Protocol) is the foundation of web communication, and it has undergone several revisions to improve speed, efficiency, and security.
* Two major versions that have shaped the modern web are HTTP/1.1 and HTTP/2.
* 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.

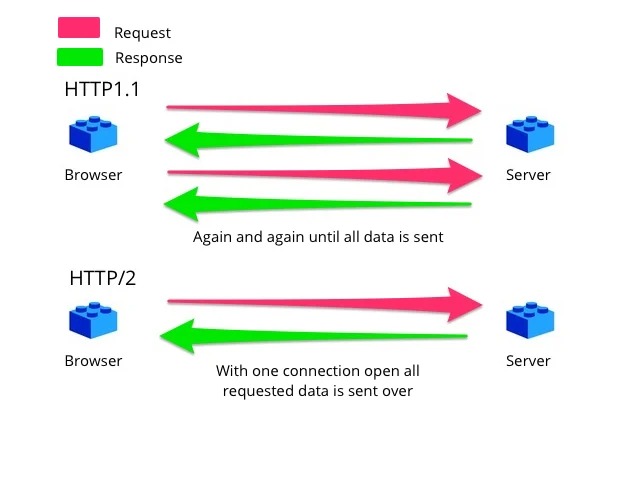
#### **What is HTTP/1.1?**

The first usable version of HTTP was created in 1997. Because it went through several stages of development, this first version of HTTP was called HTTP/1.1. This version is still in use on the web.



#### **What is HTTP/2?**

In 2015, a new version of HTTP called HTTP/2 was created. HTTP/2 solves several problems that the creators of HTTP/1.1 did not anticipate. In particular, HTTP/2 is much faster and more efficient than HTTP/1.1. One of the ways in which HTTP/2 is faster.



**Difference between HTTP/1.1 and HTTP/2 are:**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **HTTP1.1** | **HTTP2** |
| 1 | **Protocol Type** | |
|  | HTTP/1.1 is a text-based protocol. | HTTP/2 is a binary protocol, |
| 2 | **Multiplexing** | |
|  | HTTP/1.1 uses a single request-response at a time | HTTP/2 supports multiplexing, allowing multiple requests and responses simultaneously on a single connection. |
| 3 | **Header Compression** | |
|  | HTTP/1.1 sends headers with each request and response | HTTP/2 uses header compression to reduce unnecessary header data. |
| 4 | **Latency** | |
|  | HTTP/1.1 creates new connections for each resource request, incurring higher latency. | HTTP/2 uses a single connection for multiple resource requests, reducing latency. |
| 5 | **Resource Loading** | |
|  | HTTP/1.1 loads resources sequentially, potentially slowing down page rendering | HTTP/2 loads resources in parallel, improving page loading speed. |
| 6 | **Prioritization** | |
|  | HTTP/1.1 lacks built-in resource prioritization. | HTTP/2 allows for stream prioritization, ensuring critical resources are loaded first. |
| 7 | **Server Push** | |
|  | HTTP/1.1 relies on the client to request each resource. | HTTP/2 supports server push, allowing the server to send resources proactively. |
| 8 | **Connection Handling:** | |
|  | HTTP/1.1 often requires opening multiple connections to load resources in parallel. | HTTP/2 uses a single connection for all resources, reducing resource contention |
| 9 | **Error Handling:** | |
|  | HTTP/1.1 handles errors individually, often requiring the retransmission of multiple resources. | HTTP/2 can identify and handle errors at the stream level without affecting other resources. |
| 10 | **Security:** | |
|  | Both HTTP/1.1 and HTTP/2 can be used over secure connections (HTTPS), but security mechanisms are generally similar. |  |
| 11 | **Compatibility:** | |
|  | HTTP/1.1 is widely supported and compatible with older systems. | HTTP/2's binary format is more complex but efficient |

# **Objects And Its Internal Representation In JavaScript**

**INTRODUCTION OF JAVASCRIPT:**

* JavaScript is a dynamic, versatile programming language used for web development,
* objects play a pivotal role.
* JavaScript is designed on a simple object-based paradigm.
* Data types are broadly classified into 2 types:

Primitive: - (String, Boolean, Number, Null, Undefined, Symbol)

Non-Primitive: - Object (array, functions) also called object references.

**UNDERSTANDING OF JAVASCRIPT:**

* object is like a container that holds different property.
* object store their element as a key-value pairs, each key value pair is termed as property

var car = {

**Key**   **value**

color: "white",

material: "fibre",

wheels: 4,

milege: 20

}

**INTERNAL REPRESENTATION OF OBJECTS:**

Objects are instances of a built-in Object constructor, and they are implemented as collections of key-value pairs. The keys, also known as property names, are strings or symbols, and the values can be of any data type, including other objects or functions.

Here's a basic example of creating an object:

For example, this example creates an object named myCar, with properties named make, model, and year, with their values set to "Ford", "BMW", and 2010:

Var myCar = {

make: "BMW ",

model: "BMWX7",

year: 2019,

};

Unassigned properties of an object are undefined (and not null).

Car.color; // **undefined**

### **A**[**CCESSING PROPERTIES**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_objects#accessing_properties)

* You can access an object's properties and methods using two main notations:
* [Property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Property_accessors) come in two syntaxes:
* Dot notation and
* Bracket notation

For example, you could access the properties of the Car object as follows:

**// Dot notation**

Car.make = "BMW";

Car.model = "BMW" X7;

Car.year = 2019;

**// Bracket notation**

Car["make"] = "BMW";

Car["model"] = "";

Car["year"] = “2019”;

An object property name can be any JavaScript string or symbol including an empty string. However, you cannot use dot notation to access a property whose name is not a valid JavaScript identifier

// variables are created and assigned   
// separated by commas

Var Car = {

make: "BMW ",

model: "BMW X7",

year: 2019,

};

car.type = 'Dot syntax';  
car['date created'] = 'String with space';  
car[str] = 'String value';  
car[obj] = 'Object';  
car[''] = 'Even an empty string';

console.log(car);

**CONCLUSION**

JavaScript objects are the backbone of the language's flexibility and extensibility. They allow developers to create complex data structures and build modular, maintainable code. Understanding how objects are internally represented and how they can be manipulated, combined with knowledge of prototypes and inheritance, is crucial for any JavaScript developer. With objects, you can unlock the full potential of JavaScript, enabling you to create sophisticated applications for the web and beyond.