


Difference between HTTP/2 and HTTP/1.1

Last Updated : 28 Mar, 2023



HTTP stands for hypertext transfer protocol & it is used in client-server communication. By using HTTP user sends the request to the server & the server sends the response to the user. There are several stages of development of HTTP but we will focus mainly on HTTP/1.1 which was created in 1997 & the new one is HTTP/2 which was created in 2015.

HTTP/1.1: For better understanding, let's assume the situation when you make a request to the server for the `geeksforgeeks.html` page & server responds to you as a resource `geeksforgeeks.html` page. before sending the request and the response there is a TCP connection established between client & server. again you make a request to the server for image `img.jpg` & the server gives a response as an image `img.jpg`. the connection was not lost here after the first request because we add a keep-alive header which is the part of the request so there is an open connection between the server & client. there is a persistent connection which means several requests & responses are merged in a single connection. These are the drawbacks that lead to the creation of HTTP/2: The first problem is HTTP/1.1 transfer all the requests & responses in the plain text message form. The second one is head of line blocking in which TCP connection is blocked all other requests until the response does not receive. all the information related to the header file is repeated in every request.

HTTP/2: HTTP/2 was developed over the SPDY protocol. HTTP/2 works on the binary framing layer instead of textual that converts all the messages in binary format. it works on fully multiplexed that is one TCP connection is used for multiple requests. HTTP/2 uses HPACK which is used to split data from header. it compresses the header. The server sends all the other files like CSS & JS without the request of the client using the PUSH frame.

Difference between HTTP/1.1 and HTTP/2 are:

HTTP/1.1	HTTP/2
It works on the textual format.	It works on the binary protocol.
There is head of line blocking that blocks all the requests behind it until it doesn't get its all resources.	It allows multiplexing so one TCP connection is required for multiple requests.
It uses requests resource Inlining for use getting multiple pages	It uses PUSH frame by server that collects all multiple pages

It compresses data by itself.	It uses HPACK for data compression.
-------------------------------	-------------------------------------

Want to be a Software Developer or a Working Professional looking to enhance your **Software Development Skills**? Then, master the concepts of Full-Stack Development. Our [Full Stack Development - React and Node.js Course](#) will help you achieve this quickly. Learn everything from **Front-End to Back-End Development** with hands-on **Projects** and real-world examples. This course enables you to build scalable, efficient, dynamic web applications that stand out. Ready to become an expert in Full-Stack? Enroll Now and Start Creating the Future!

v Objects And Its Internal Representation In JavaScript

Objects, in JavaScript, is it's most important data-type and forms the building blocks for modern JavaScript. These objects are quite

different from JavaScript's primitive data-types(Number, String, Boolean, null, undefined and symbol) in the sense that while these primitive data-types all store a single value each (depending on their types).

Objects are more complex and each object may contain any combination of these primitive data-types as well as reference data-types.

An object, is a reference data type. Variables that are assigned a reference value are given a reference or a pointer to that value. That reference or pointer points to the location in memory where the object is stored. The variables don't actually store the value.

Loosely speaking, objects in JavaScript may be defined as an unordered collection of related data, of primitive or reference types, in the form of "key: value" pairs. These keys can be variables or functions and are called properties and methods, respectively, in the context of an object.

For Eg. If your object is a student, it will have properties like name, age, address, id, etc and methods like `updateAddress`, `updateNam`, etc.

Objects and properties

A JavaScript object has properties associated with it. A property of an object can be explained as a variable that is attached to the object.

Object properties are basically the same as ordinary JavaScript variables, except for the attachment to objects. The properties of an object define the characteristics of the object. You access the properties of an object with a simple dot-notation:

```
objectName.propertyName
```

Like all JavaScript variables, both the object name (which could be a normal variable) and property name are case sensitive. You can define a property by assigning it a value. For example, let's create an object named `myCar` and give it properties named `make`, `model`, and `year` as follows:

```
var myCar = new Object();
```

```
myCar.make = 'Ford';
```

```
myCar.model = 'Mustang';
```

```
myCar.year = 1969;
```

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

```
myCar.color; // undefined
```

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](#)). Objects are sometimes called *associative arrays*, since each property is associated with a string value that can be used to access it. So, for example, you could access the properties of the `myCar` object as follows:

```
myCar['make'] = 'Ford';
```

```
myCar['model'] = 'Mustang';
```

```
myCar['year'] = 1969;
```

An object property name can be any valid JavaScript string, or anything that can be converted to a string, including the empty string. However, any property name that is not a valid JavaScript identifier (for example, a property name that has a space or a hyphen, or that starts with a number) can only be accessed using the square bracket notation. This notation is also very useful when property names are to be dynamically determined (when the property name is not determined until runtime). Examples are as follows:

```
// four variables are created and assigned in a single go,
```

```
// separated by commas
```

```
var myObj = new Object(),
```

```
    str = 'myString',
```

```
    rand = Math.random(),
```

```
obj = new Object();
```

```
myObj.type = 'Dot syntax';
```

```
myObj['date created'] = 'String with space';
```

```
myObj[str] = 'String value';
```

```
myObj[rand] = 'Random Number';
```

```
myObj[obj] = 'Object';
```

```
myObj[''] = 'Even an empty  
string'; console.log(myObj);
```

You can also access properties by using a string value that is stored in a variable:

```
var propertyName = 'make';
```

```
myCar[propertyName] = 'Ford'; propertyName = 'model';
```



```
myCar[propertyName] = 'Mustang';
```

You can use the bracket notation with [for...in](#) to iterate over all the enumerable properties of an object. To illustrate how this works, the following function displays the properties of the object when you pass the object and the object's name as arguments to the function:

```
function showProps(obj, objName) {  
  
    var result = ``;  
  
    for (var i in obj) {  
  
        // obj.hasOwnProperty() is used to filter out properties from  
        the object's prototype chain  
  
        if (obj.hasOwnProperty(i)) {  
  
            result += `${objName}.${i} = ${obj[i]}\n`;  
  
        }  
    }  
}
```

```
}
```

```
return result;
```

```
}
```

So, the function call `showProps(myCar, "myCar")` would return the following:

```
myCar.make = Ford
```

```
myCar.model = Mustang
```

```
myCar.year = 1969
```

Creating Objects In JavaScript :

Create JavaScript Object with Object Literal

One of easiest way to create a javascript object is object literal, simply define the property and values inside curly braces as shown below

```
let bike = {name: 'SuperSport', maker:'Ducati', engine:'937cc'};
```

Create JavaScript Object with Constructor

Constructor is nothing but a function and with help of new keyword, constructor function allows to create multiple objects of same flavor as shown below

```
function Vehicle(name, maker) {
```

```
    this.name = name;
```

```
    this.maker = maker;
```

```
}
```

```
let car1 = new Vehicle('Fiesta', 'Ford');
```

```
let car2 = new Vehicle('Santa Fe', 'Hyundai')
```

```
console.log(car1.name);    //Output: Fiesta
```

```
console.log(car2.name); //Output: Santa Fe
```

Using the JavaScript Keyword new

The following example also creates a new JavaScript object with four properties:

Example

```
var person = new Object();  
  
person.firstName = "John";  
  
person.lastName = "Doe";  
  
person.age = 50;  
  
person.eyeColor = "blue";
```

Using the `Object.create` method

Objects can also be created using the [Object.create\(\)](#) method. This method can be very useful, because it allows you to choose the

prototype object for the object you want to create, without having to define a constructor function.

```
// Animal properties and method encapsulation

var Animal = {

  type: 'Invertebrates', // Default value of properties

  displayType: function() { // Method which will display type of
Animal

    console.log(this.type);

  }

};

// Create new animal type called animal1

var animal1 = Object.create(Animal);
```

```
animal1.displayType(); // Output:Invertebrates
```

```
// Create new animal type called Fishes
```

```
var fish = Object.create(Animal);
```

```
fish.type = 'Fishes';
```

```
fish.displayType();
```

```
// Output:Fishes
```