# Difference between HTTP1.1 vs HTTP2

HTTP - 1.1

supports network re-use i.e. in all TCP communications there may be multiple requests and responses, as well as plumbing where the client can request multiple services on the server at once. However, pipe loading was difficult to use due to problems such as head blockage and was not a possible solution.

Introducing the warning header field to contain more information about message status. It can define 24 status codes, reporting errors is fast and very efficient.

It is relatively secure to HTTP - 1.0 as it uses alarm authentication and NTLM authentication.

Expands on the caching support by using additional headers like cache-control, conditional headers like If-Match and by using entity tags.

HTTP / 1.1 provides faster delivery of web pages and reduces web traffic compared to HTTP / 1.0. However, TCP starts slowly and with domain splitting (resources can be downloaded simultaneously using multiple domains), re-use of connections and pipelines, there is an increased risk of network congestion.

HTTP - 2.0

It uses multiplexing, in which more than one TCP connecting service to be delivered varies and reaches the client at about the same time. It is done using pre-stream streams and can be dependent on and control each flow. It also provides a feature called server push that allows the server to send the data a client will need but has not yet requested.

Lower HTTP semantics like titles and status codes remain the same.

Security concerns from previous versions will continue to appear on HTTP / 2. However, it is better equipped to deal with them due to new TLS features such as Inadequate\_Security communication error.

HTTP/2 does not change much in terms of caching. With the server push feature, if the client finds the resources are already present in the cache, it can cancel the pushed stream.

HTTP/2 utilizes multiplexing and server push to effectively reduce the page load time by a greater margin along with being less sensitive to network delays.

# Objects And Its Internal Representation In JavaScript

Objects, in JavaScript, are the most important data type and form the building blocks of modern JavaScript. These items are very different from older JavaScript data types (Number, Thread, Blind, null, unspecified and symbol) in the sense that these older data types all store the same value individually (depending on their types).

Objects are very complex and each object may contain any combination of these old data types and reference data types. An object is a type of reference data. Variables assigned to a reference value are given a reference or reference to that value. That reference or indicator points to the location in the memory where the object is stored. Variables actually do not limit value.

JavaScript objects can be defined as a non-ordered collection of related, classic or reference data, in the form of “key: pairs”. These keys can be flexible objects or functions and are called structures and methods, respectively, in the context of the object.

For example. If your item is a student, it will have properties such as name, age, address, id, etc. and methods such as Update Address, Update Name, etc.

### Objects and properties

The JavaScript object has related features. The structure of an object can be defined as the flexibility attached to an object. Object structures are basically the same as standard JavaScript variables, with the exception of objects being attached to them. The properties of an object define the characteristics of the object. You access object features with simple dot-notation:

objectName.propertyName

Like all JavaScript variables, both the object name and property name are case sensitive. You can define a property by assigning it a value.

var myCar = new Object();

myCar.make = 'Ford';

myCar.model = 'Mustang';

myCar.year = 1969;

Unassigned properties of an object are undefined.

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation. 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;

The object property name can be any valid JavaScript string unit and anything that can be converted to a string, including an empty character unit. However, any design name that is not a valid JavaScript identifier can only be accessed using square brackets notice. This notice is especially useful if the names of the properties are to be determined alternately (where the property name can be determined until the time of operation). 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