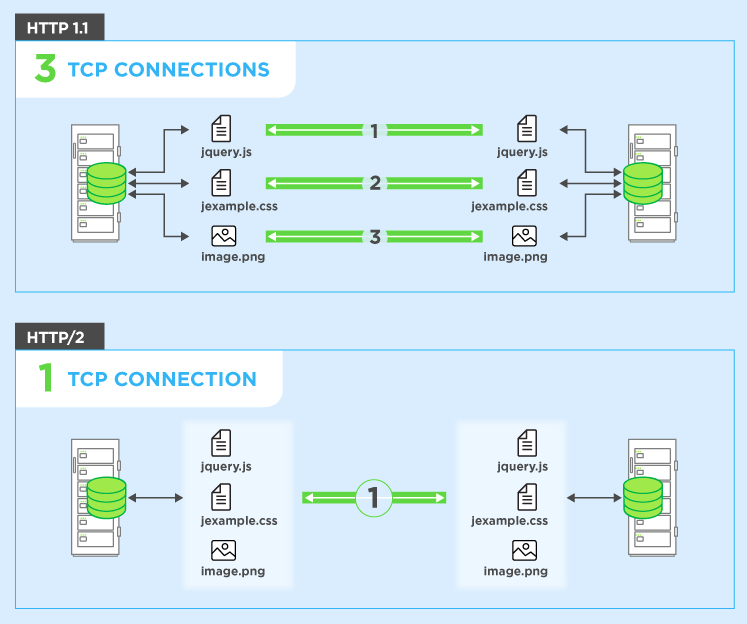
|  |  |  |
| --- | --- | --- |
|  | **HTTP1.1** | **HTTP2** |
| **Year** | First version of http created in 1997 | New version of http created in 2015 |
| **Protocol** | It uses textual protocol, which can be read and interpreted by humans | It uses binary protocol, which is more efficient to parse, more compact in size, and less error-prone. |
| **Multiplexing** | Each transfer requires a new TCP connection or uses connection keep-alive, which can still only process one request and response at a time. | Multiple requests and responses can be sent simultaneously and asynchronously over a single TCP connection. This reduces latency significantly, especially for sites with numerous small resources. |
| **Header Compression** | Headers are sent in plain text and can become quite large in size. | Employs HPACK compression, which reduces overhead, especially for repetitive headers. |
| **Stream Prioritization** | No built-in mechanism for resource prioritization. | Allows the client to specify the priority of a request, enabling the server to send the most important resources first. |
| **Server Push** | No push capability. | Introduces “server push”, where a server can send resources proactively to the client’s cache before the client requests them. |
| **Connection Use** | Browsers often open multiple TCP connections to fetch resources in parallel, which can add overhead. | A single connection per origin is multiplexed, eliminating the need for multiple connections. |
| **Connection Handshake** | Each new connection requires a handshake. | After the initial connection, additional streams do not require a full SSL/TLS handshake. |
| **Improved Security** | HTTPS is optional. | Most implementations support or even require encryption, thereby promoting a more secure web. |
| **Reduced Latency** | Due to the head-of-line blocking and lack of multiplexing, latency can be an issue. | Multiplexing and prioritization significantly reduce latency. |



**Objects and its representation**

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.

notation:

objectName.propertyName

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

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';

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 the4Object.create()4method.

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

