1. Difference between http1.1 and http2.0

DEFINTION

The Hypertext Transfer Protocol, or HTTP, is an application protocol that has been the de facto standard for communication on the World Wide Web since its invention in 1989. From the release of HTTP/1.1 in 1997 until recently, there have been few revisions to the protocol. But in 2015, a reimagined version called HTTP/2 came into use, which offered several methods to decrease latency, especially when dealing with mobile platforms and server-intensive graphics and videos. HTTP/2 has since become increasingly popular, with some estimates suggesting that around a third of all websites in the world support it. In this changing landscape, web developers can benefit from understanding the technical differences between HTTP/1.1 and HTTP/2, allowing them to make informed and efficient decisions about evolving best practices.

The HyperText Transfer Protocol (HTTP) is a protocol resides in the application layer. HTTP defines how the client and the server exchange the message, which means how the client requests a Web page and how the server responds the web page to the client. When the client clicks on the link, the browser sends the HTTP request to the server, the server receives the request and responses back to the client.

HTTP/2 allows clients to make **multiple requests** over one TCP connection while HTTP/1.1 allows only **one request** (even with request pipelining).

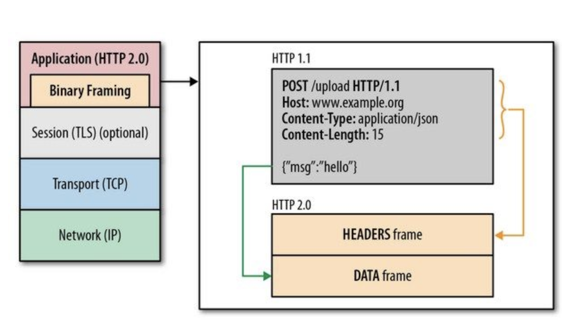
* HTTP/2 is a **binary protocol**. This is potentially much more compressed than HTTP/1.1 and this fewer bytes on-the-wire. This means less bandwidth usage and faster load times - benefits which are amplified with many smaller requests.
* HTTP/2 is **multiplexed**. This means many requests can leverage an existing connection in parallel. Coming from HTTP/1.1’s one connection per request this is a huge leap in efficiency.
* HTTP/2 requests can be **pushed.**This means requests can be pre-sent (and be ready before they've even been requested). While that creates some interesting questions (mostly revolving around caches) it's a potentially exciting area of development.
* HTTP/2 must be **encrypted.**At least for browsers. This is a valuable best practice and I'm all for seeing browser vendors flex their muscle where it benefits users.

**The main differences between HTTP 1.1 vs HTTP 2.0**

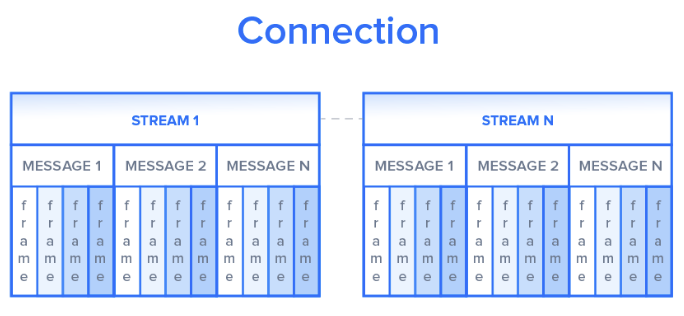
There are mainly four differences between HTTP 1.1 and HTTP 2.0. Firstly, HTTP 2.0 has a binary format layer. Secondly, HTTP 2.0 introduces multiplexing. Thirdly, HTTP 2.0 adds the header compression. Fourthly, HTTP 2.0 has the server push function. Let me explain them one by one.

**1. Binary format layer**

HTTP 2.0 adds a binary format layer between the application layer and the transport layer. HTTP 1.0 transfers the plain-text message, but HTTP 2.0 transfers the binary frame. Thus, the message format is different. As is shown in the image below, you can see the binary frame layer breaks the message into frames. The header of HTTP 1.1 would be encapsulated into the HEADERS frame, the response body would be encapsulated into the DATA frame.



HTTP 2.0 sets up a TCP connection, within this connection, there are amounts of the bilateral streams of data. Each stream consists of multiple messages in the request/response format, each message can be split into small units called the frame. The image below showcases the format of the stream.

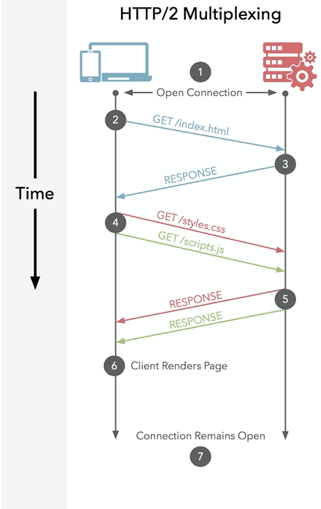


The advantage of the binary format layer is that it increases the flexibility of data transfer. The reason is plain-text is diversified, thus, it is much more difficult for HTTP 1.1 to deal with the robustness. However, HTTP 2.0 transfer binary format, this would be much more helpful for maintaining the robustness.

**2. Multiplexing**

HTTP 1.0 requests and responses in a stop-and-wait way, which has pretty low efficiency. HTTP 1.1 introduces pipelining, we mentioned above. Although the persistent connection with pipelining improves the performance over stop-and-wait, this optimization strategy has the bottleneck, which would cause Head of Line (HOL) Blocking. Head of Line (HOL) Blocking means if the head of the packet cannot be passed by the destination port when it arrives at the destination, it will cause other packets behind to be blocked.

To improve this, HTTP 2.0 introduces multiplexing and tags each frame. Multiplexing allows the client to construct the multiple streams in parallel, these streams share a single TCP connection. In the following graph, you can see the process of multiplexing.



In HTTP 2.0, each frame would be tagged to a specific stream, the tag allows the connection to interleave these frames during transfer and reassemble them at the other side. Thus, the request and response frames can be transferred in parallel without blocking the behind messages.

**3. Header compression**

Let’s see an example before explaining the header compression. Suppose we have the following two requests, the headers of these two requests are as follow.

A screenshot of a computer

Description automatically generated

Everything is the same except path field. In HTTP 1.1, we have to send these two request messages twice, which means the same fields will be sent twice. Thus, the size of the messages will be pretty large. However, in HTTP 2.0. these two headers will be encapsulated into the header frames. HTTP 2.0 can compress the header frames. The result header frames are as follow:

A screenshot of a computer screen

Description automatically generated

When sending the request 2, we only encode the path field and we can reconstruct the header with other common fields. This is the process of header compression.

**4. Server push**

Before I explain the server push of HTTP 2.0, let me introduces how to improve the speed of requesting and resource lining of HTTP 1.1.

HTTP 1.1 uses resource lining to send the client objects that they might need before the client asks for them. For example, if a client requests for a CSS file, the server includes other objects that the client might need into the file, and package them together and send back the client. Resource lining will reduce the total number of requests, but it also has some drawbacks. Firstly, the server put all the resources together and the client cannot separate or decline to receive them if it doesn’t need them. Secondly, if putting too many objects within an HTML file, then the HTML file will be very large, which decreases the connection speed.

To improve these problems, HTTP 2.0 introduces the server push. Since HTTP 2.0 enables sending concurrent responses to a client’s get request, the server can send other objects separately along with the requested HTML file. In this way, the client can choose to cache them or decline them.

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

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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined) (and not [null](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/null)).

myCar.color; // undefined

Properties of JavaScript objects can also be accessed or set using a bracket notation (for more details see [property accessors](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/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](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/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()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/create" \t "_blank) 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