**What is HTTP**

Hypertext Transfer Protocol (HTTP) is an application protocol that is, currently, **the foundation** of data communication for the World Wide Web.

**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.

A simple and abstract example would be a **restaurant guest and a waiter.** The guest (**Client**) asks (**sends request**) waiter (**Server**) for a meal, then the waiter gets the meal from the restaurant chef (**your application logic**) and brings the meal to the guest.

**Difference between HTTP 1.1 and HTTP 2.0**

|  |  |
| --- | --- |
| 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 |

# Features of HTTP 2.0

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

# Request multiplexing

HTTP/2 can send **multiple requests** for data in parallel over a **single** TCP connection. This is **the most advanced feature** of the HTTP/2 protocol because it **allows you to download web files asynchronously from one server**. Most modern browsers limit TCP connections to one server.

# 2. Header compression

HTTP/2 compress a large number of redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests.

HPACK compresses the individual value of each header before it is transferred to the server, which then looks up the encoded information in a list of previously transferred header values to reconstruct the full header information.

# 3. Binary protocol

The latest HTTP version has evolved significantly in terms of capabilities and attributes such as transforming from a text protocol to a binary protocol. HTTP1.1 used to process text commands to complete request-response cycles. HTTP/2 will use binary commands (in 1s and 0s) to execute the same tasks. This attribute eases complications with framing and simplifies implementation of commands that were confusingly intermixed due to commands containing text and optional spaces.

# 4. HTTP/2 Server Push

This capability allows the server to send additional cacheable information to the client that isn’t requested but is anticipated in future requests. For example, if the client requests for the resource X and it is understood that the resource Y is referenced with the requested file, the server can choose to push Y along with X instead of waiting for an appropriate client request.

**OBJECTS**

In JavaScript, an object is an unordered collection of key-value pairs. Each key-value pair is called a property.

The key of a property can be a string. And the value of a property can be any value, e.g., a [string](https://www.javascripttutorial.net/javascript-string-type/), a [number](https://www.javascripttutorial.net/javascript-number/), an [array](https://www.javascripttutorial.net/javascript-array/), and even a [function](https://www.javascripttutorial.net/javascript-function/).

JavaScript provides you with many ways to create an object. The most commonly used one is to use the object literal notation.

1. The following example creates an empty object using the object literal notation:

Let Sachin = { }

1. To create an object with properties, we use the (Key: Value) within the curly braces. For example, the following creates a new Person object:

let Sachin={

firstName: ”Sachin”

lastName: ”Yadav”

};

**Ways to access property of objects:**

1. Dot notation

**General representation: objectName.propertyName**

**Example: Sachin.firstName**

**Complete example**

let Sachin={

firstName: ”Sachin”,

lastName: ”Yadav”

};

Console.log(Sachin.firstName)

Console.log(Sachin.lastName)

**(The above example will print first and last name of person)**

### 2) Array-like notation ([ ])

**General representation: objectName[“propertyName”]**

**Example: Sachin[“firstName”]**

**Complete example**

let Sachin={

firstName: ”Sachin”,

lastName: ”Yadav”

};

Console.log(Sachin[“firstName”])

Console.log(Sachin[“lastName”])

**(The above example will print first and last name of person)**