Difference between HTTP1.1 and HTTP2

**HTTP1.1**

Version 1.1 of HTTP was released in 1997 and is an enhancement of HTTP 1.0, providing several extensions.

Following are some important enhancements:

1. Host header: HTTP 1.0 does not officially require the host header. HTTP 1.1 requires it by the specification.
2. Persistent connections: in HTTP 1.0, each request/response pair requires opening a new connection. In HTTP 1.1, it is possible to execute several requests using a single connection.
3. New methods: 1.1 version added six extra methods: PUT, PATCH, DELETE, CONNECT, TRACE, and OPTIONS

**HTTP2**

HTTP2 was released in 2015 and is a patch of enhancements to solve the problems and limitations of the HTTP1.1

It implemented several features to improve connections and data exchange. They are

Request multiplexing: HTTP 1.1 is a sequential protocol. So, we can send a single request at a time. In HTTP2, we can do multiple requests at the same time using a single connection.

Request prioritization: with HTTP2, we can set a numeric prioritization in a batch of requests.

Automatic compressing: we must explicitly require the compression of requests and responses in HTTP1.1. HTTP2, executes a GZip compression automatically

Server push: HTTP2 introduced a server push functionality which proactively pushes these resources to the client cache.

|  |  |  |
| --- | --- | --- |
|  | HTTP1.1 | HTTP2 |
| Delivery Model | Works on text format | Works on binary protocol |
| Predicting Resource Request | It uses requests resource Inlining for getting multiple pages | It uses PUSH frame by server that collects all multiple pages |
| Buffer Overflow | 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. |
| Compression | It compresses data by itself. | It uses HPACK for data compression. |

Objects and Internal representation in JavaScript

JavaScript objects are a collection of properties in a key-value pair. These objects can be understood with real-life objects, like similar objects have the same type of properties, but they differ from each other.

Some important points about objects are –

1. Object contains properties separated with a comma( , ).
2. Each property is represented in a key-value pair.
3. Key and value are separated using a colon( : ).
4. The key can be a string or variable name that does not contain any special characters, except underscore( \_ ).
5. The value can contain any type of data - primitive, non-primitive, and even a function.
6. The objects are passed by reference to a function.

**Creating an Object**

The object can be created in two ways –

Using curly brackets - We can create empty object as –

var obj = {}; and

an object with some initial properties as –

var obj = {key1: value1, ..., keyN: valueN}

Using new operator - We can create empty object as –

var obj = new Object(); and

an object with properties as –

var obj = new Object({key1: value1, ..., keyN: valueN})

**Creating and Accessing Properties**

There are two ways to create and access properties:

Using a dot operator –

Property can be accessed like - obj.propertyName.

Similarly, you can create property like –

obj.propertyName = value

Using a square bracket –

Property is accessed like –

obj["propertyName"].

Similarly, you create property like –

obj["propertyName"] = value

**Deleting Property**

the property of the object can be removed using the 'delete' operator followed by the property name (can either use dot operator or square bracket notation).

delete obj["objectName"] OR

delete obj.objectName

**How Objects Are Stored**

* Objects are stored in heap.
* Objects are reference types.

Object variables point to the location where they are stored. This means that more than one variable can point to the same location.