1. **Difference between HTTP1.1 vs HTTP2**

|  |  |
| --- | --- |
| **HTTP1.1** | **HTTP2** |
| It was released in **1997** | It was released in **2015** |
| **HTTP1.1** was developed by **Timothy Berners-Lee** as a communication standard for World Wide Web. | **HTTP2** began as the SPDY protocol, developed primarily at **Google** with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. |
| It keeps all requests and responses in **plain text format** | It uses the binary framing layer to encapsulate all messages in **binary format**, while still maintaining HTTP semantics, such as verbs, methods, and headers. |
| The delivery models used is Pipelining and Head-of-Line Blocking (HOL) i.e. request at the head of the queue that cannot retrieve its required resource will block all the requests behind it. | The delivery **model** used is **Binary framing layer**. It encodes requests/responses and cuts them up into smaller packets of information, greatly increasing the flexibility of data transfer. |
| Adding separate**, parallel TCP (Transmission control protocol) connections to minimize the effect of HOL blocking**, but there are limits to the number of concurrent TCP connections possible between a client and server, and each new connection requires significant resources. | A process called **multiplexing resolves the HOL blocking** which means that servers and clients can send concurrent requests and responses, allowing for greater control and more efficient connection management using single TCP connection. |

|  |  |
| --- | --- |
| **HTTP1.1** | **HTTP2** |
| It **relies on the transport layer to avoid buffer overflow**, each new TCP connection requires a separate flow control mechanism. | It multiplexes streams of data within a single TCP connection by allowing the **client and server to implement their own flow controls**, rather than relying on the transport layer. |
| It uses a technique called**resource inlining**to include the required resource directly within the HTML document that the server sends in response to the initial ***GET*** request. | A process called **Server Push** enables multiple concurrent responses to a client’s initial ***GET***request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. |
| A major drawback of resource inlining, is that the client **cannot separate the resource and the document**. | In the Server Push process the client can decide to cache or decline the pushed resource **separate from the main HTML document,** fixing the major drawback of resource inlining. |
| The header component of a message is always sent as plain text. The **uncompressed header** data weighs heavier and heavier on the connection as more requests are made. | It can split headers from their data, resulting in a header frame and a data frame using specific compression program  [**HPACK**](https://tools.ietf.org/html/draft-ietf-httpbis-header-compression-12)**can compress this header frame.** |
| It does not allowdevelopers to gain better control over web page rendering. | A method called **Stream Prioritization**  allows developers to customize the relative weight of requests to better optimize application performance. |

**2. Objects and its internal representation in javascript**

**Javascript Object**

* A **Javascript Object** is a collection of properties and methods. It is a way to group data together and store it in a structured way.
* The data in objects are stored in key-value pairs. Where each key-value pair is called a property. The key is the name of the property and the value is the value of the property.
* The key of an object is a string while the value can be any **data type**, including another object.

**Example:**

var person = {

firstName: "John",

lastName : "Doe",

age : 50

};

**Creating Object in Javascript**

Javascript objects can be created in many different ways. Some of the ways are:

1. Javascript object literal
2. Javascript object.create method
3. Javascript object constructor
4. Object using class
5. **Javascript Object Literal**

* The simplest way to create an object in javascript is using the **object literal**.
* It is a comma separated key-value pair enclosed in curly braces ({}) assigned to a variable.
* To create an empty object just assign a curly brace to a variable.
* The key-value pairs are separated by a comma (,).

**Example:**

let person = {

// adding properties to an object

firstName: "John",

lastName : "Doe",

age : 50

};

// checking type

console.log(typeof person);

1. **Javascript object.create method**

* The Object.create() method is used to create an object from an existing object. It creates a new object with the same properties as the existing object.
* It uses an existing object as a prototype of the newly created object.

**Example:**

// prototype object

let person = {

firstname: "steve",

lastname: "jobs",

fullName: function() {

return "My name is " + this.firstname + " " + this.lastname;

}

}

// create a new object

let newPerson = Object.create(person);

// change the value of the new object

newPerson.firstname = "John";

newPerson.lastname = "Smith";

console.log(newPerson.fullName());

1. **Javascript object constructor**

* The object constructor is a function that is used to create an object. It is defined using the **new** keyword.
* The object constructor is used to create an object with a specific set of properties and methods.
* If we want to create multiple objects of the same type then it is better to use the **object constructor**.

**Example:**

// creating a function to create an object

function person(firstname, lastname, age) {

this.firstname = firstname;

this.lastname = lastname;

this.age = age;

}

// creating an object using the function

let person1 = new person("steve", "jobs", 25);

console.log(person1);

let person2 = new person("john", "smith", 35);

console.log(person2);

1. Object using class

* Class  was introduced in ES6. The class is used to create an object constructor. It is defined using the ***class*** keyword.
* There is difference between the class and the object constructor. All the difference is in their syntax.

**Example:**

// creating a class to create an object

class person {

constructor(firstname, lastname) {

this.firstname = firstname;

this.lastname = lastname;

}

fullName() {

return "My name is " + this.firstname + " " + this.lastname;

}

}

// creating an object

let person1 = new person('John', 'Smith');

console.log(person1.fullName());

## Accessing properties of Object

There are two ways to access the properties of an object

1. Using the dot operator
2. Using the bracket operator
3. **Using the dot operator**

* Using the **dot operator**, you can access the properties of an object by writing the object name followed by a dot and the property name.

**Example:**

let car = {

name: "Ford",

color: "Black",

}

// accessing the property

console.log(car.name);

console.log(car.color);

1. **Using the bracket operator**

* It accesses the properties of an object by writing the object name followed by the bracket and the property name as a string.
* If you want to access the value of a property whose name is a variable or property name is a space separated string, then you can use the bracket operator.

**Example:**

let car = {

name: "Ford",

color: "Black"

}

// accessing the property

console.log(car["name"]);

console.log(car["color"]);

## Adding properties to an Object

* You can add a new property to an existing object simply by object name followed by a dot and the property name and assigning the value to the property.

**Example:**

let car = {

name: "Ford",

color: "Black"

}

console.log(car);

// adding a new property

car.model = "Mustang";

console.log(car.model);

* Alternatively, you can use square bracket notation to add a new property to an object.

**Example:**

let car = {

name: "Ford",

color: "Black"

}

console.log(car);

// adding a new property

car['model'] = "Mustang";

console.log(car.model);

## Deleting properties from an Object

* You can delete a property from an object by using the **delete** operator.

**Example:**

let car = {

name: "Ford",

color: "Black",

model: "Avenger"

}

console.log(car.model);

// deleting a property

delete car.model;

console.log(car.model);

## Looping through an Object

1. **Using for...in loop**

The **for...in** loop access the properties of an object by iterating through the object.

**Example:**

let car = {

name: "Ford",

color: "Black",

model: "Avenger"

}

// looping through the object

for (let key in car) {

console.log(key, car[key]);

}

1. **Using Object.keys() loop**

The **Object.keys()** method returns an array of a given object's property names. Which can be used to loop through the object.

**Example:**

let car = {

name: "Ford",

color: "Black",

model: "Avenger"

}

// looping through the object

for (let key of Object.keys(car)) {

console.log(key, car[key]);

}

1. **Using Object.values() loop**

The **Object.values()** method returns an array of a given object's property values. Which gives us direct access to the values of the object.

**Example:**

let car = {

name: "Ford",

color: "Black",

model: "Avenger"

}

// looping through the object

for (let value of Object.values(car)) {

console.log(value);

}

1. **Using Object.entries() loop**

The **Object.entries()** method returns an array of a given object's property names and values.

**Example:**

let car = {

name: "Ford",

color: "Black",

model: "Avenger"

}

// looping through the object

for (let entry of Object.entries(car)) {

console.log(entry);

}

## Javascript array of objects

* An array of objects is a collection of objects. Apart from a single object, you can create an array of objects to store multiple objects.
* To access the properties of these objects you can simply loop through the array.

**Example:**

var students = [

{

name: "std1",

roll: 1,

marks: 66

},

{

name: "std2",

roll: 2,

marks: 58

},

{

name: "std3",

roll: 3,

marks: 76

},

{

name: "std4",

roll: 4,

marks: 80

}

];

// looping through array

for (let i = 0; i < students.length; i++) {

console.log(students[i].name);

}

## this keyword in javascript object

* this keyword in an object represents the object itself. It is used to access the properties of the object. When you say **this.propertyName** it means **object.propertyName.**
* In the example below the sum is a method in 'number' object which is using this keyword to access properties of its parent element.

**Example:**

var number = {

num1: 10,

num2: 15,

sum: function(){

return this.num1 + this.num2

}

}

console.log(number.sum());