Objects and its internal representation in JavaScript

Object in JavaScript can be defined as a collection of same or different primitive data types organized in key-value pairs. Object in JS, like most programming languages represents a real-world object and contains states and functions/methods.

**Standard notation of a JS Object:**

var obj={

key1:value1,

key2:value2

};

**Both key and value can be of any primitive type.**

Eg:

var employee={

name: ‘Ashley’,

age:24,

18:’yes’};

**Can contain a member function also**

var employee={

name: ‘Ashley’,

age:24,

18:’yes’,

job: function(){

console.log(‘code all day’);

};

**Accessing values**

Values can be accessed through their keys using the dot(.) operator or using the square brackets([ ])

Eg:

var employee={

name: ‘Ashley’,

age:24,

18:’yes’,

job: function(){

console.log(‘code all day’);

};

employee.name //Ashley

employee[‘18’] //yes

employee.job() //code all day

**Methods of creating object**

A] Object literal syntax

Using curly braces

var employee={

name: ‘Ashley’,

age:24,

18:’yes’,

job: function(){

console.log(‘code all day’);

};

B] Object constructor

Using the new keyword and Object constructor to create object and assign values

var employee=new Object();

employee.name: ‘Ashley’,

employee.age=24;

C] Constructor function

Used to initialize the common properties of all objects using ‘this’ keyword

function employeeMaker(name,age){

this.name=name;

this.age=age;

}

var employee=new employeeMaker(‘Ashley’,24);

D] Class constructor

Constructors are special methods used to initialize objects in classes. The object inherits all properties initialized by the constructor prototype object.

class employees{

constructor(){

this.job=function(){

console.log(‘code all day’);}

let employee=new employees();

employee.job(); //code all day

E] Prototypes

Every prototype function has its own prototype object. When we create an object based on the prototype function, we assign all properties of its object prototype to the object we are making.

let employees={

job: function(){

console.log(‘code all day’);

}

}

let employee = Object.create(employees);

employee.job(); //code all day

Authentication & CORS

Cross-Origin-Resource-Sharing is a mechanism which allows client browser to access resources from an origin other than the point where it is running. In other words, a browser/web application makes a cross-origin request when it requests for a resource on a server/origin other than the one it is executing at.

However, this is only possible when HTTP requests made by client browsers also include headers that are CORS specific. If these headers are not added along with the standard HTTP request headers, the request for CORS will be blocked since XMLHTTP follow only the same-origin policy.

The following fields are added in the HTTP request headers in both the client and server side communication;

**Origin Access:**

Client header:

Origin : <https://example.com>

Server response header:

Access-Control-Allow-Origin: <https://example.com>

**Pre-flight Access:**

In this procedure, the client makes a pre-request to the server to confirm if it can make the request. The response to this pre-flight request by the server gives a confirmation whether the actual request can be made or not.

Apart from the standard contents given in an HTTP request, the pre-flight request contains the following fields in the client side request.

Client header:

Origin: http://foo.example

Access-Control-Request-Method: POST

Access-Control-Request-Headers: X-PINGOTHER, Content-Type

The response of the server on request in case of allowing CORS, will be as follows:

Server SAccess-Control-Allow-Origin: https://foo.example

Access-Control-Allow-Methods: POST, GET, OPTIONS

Access-Control-Allow-Headers: X-PINGOTHER, Content-Type.

Once the pre-flight request is accepted, the real request is then made.