Object in JavaScript - Object Creation

Anil Singh 4:56 AM

How to create a JavaScript object?

The JavaScript object is a collection of properties and the each property associated with the name-value pairs. The object can contain any data types (numbers, arrays, object etc.)

The example looks like,

```
Var myObject= {empId : “001”, empCode :”X0091”};
```

In the above example, here are two properties one is empId and other is empCode and its values are “001” and “X0091”.

The properties name can be string or number. If a property name is number i.e.

```
Var numObject= {1 : “001”, 2 :”X0091”};

Console.log(numObject.1);  //This line throw an error!

Console.log(numObject[“1”]);  // will access to this line not get any error!
```

As per my thought, the number property name should be avoided.

Types of creating an object (There are two types)

1. Object literals
2. Object constructor

Object Literals: This is the most common way to create an object with object literal and the example as given below.

The empty object initialized using object literal i.e.

```
var emptyObj= {};
```

This is an object with 4 items using object literal i.e.

```
var emptyObj ={
  empId:”Red”,
  empCode: “X0091”,
 empDetail : function(){
    alert(“Hi”);
};
};
```

Object Constructor: The second way to create object using object constructor and the constructor is a function used to initialize new object.

The example looks like,

```
Var obj = new Object();
Obj.empId=”001”;
Obj.empCode=”X0091”;
Obj.empAddressDetai = function(){
    Console.log(“Hi, I Anil”);
};
```

Summary:

There is no best way to create an object. It’s depending on your uses case. You can choose all one as per your case. All have some benefits.

There are some styles are available to create an objects,

1. Object Constructor,
2. Literal Constructor,
3. Function Based,
4. Prototype Based,
5. Function and
6. Prototype Based Singleton Based.

Prototype in JavaScript

Anil Singh 5:00 AM
What is Prototype in JavaScript?
How to create prototype in JavaScript?

The prototype is a fundamental concept of JavaScript and its must to known JavaScript developers and all the JavaScript objects have an object and its property called prototype and its used to add and the custom functions and property.

The example looks like,

```
var employee = function () {
    //This is a constructor  function.
}

//Crate the instance of above constructor  function and assign in a variable
var empInstance = new employee();
empInstance.deportment = "IT";

console.log(empInstance.deportment);//The output of above is IT.
The example with prototype as given below.
var employee = function () { //This is a constructor  function.}

employee.prototype.deportment = "IT";//Now, for every instance employee will have a
deportment.

//Crate the instance of above constructor function and assign in a variable
var empInstance = new employee();
empInstance.deportment = "HR";

console.log(empInstance.deportment);//The output of above is HR not IT.
```

What is ‘this’ in JavaScript?

Anil Singh 9:09 PM

Hello everyone, today's I am going to share a basic and very confusing concepts that is called 'this' keyword :)

The 'this' keyword behaves a little differently in JavaScript compared to other languages.

In most of the other languages, 'this' keyword is a reference to the current object instantiated by the classes and methods.

In the JavaScript languages, 'this' keyword refers to the object which 'owns' the method, but it depends on how a function is called.

The examples in details as given below.

```
//Global Scope in JavaScript
//In the below example, ‘this’ keyword refers to the global object.

window.sms = "Hi, I am window object";

console.log(window.sms);
console.log(this.sms); // Hi, I am window object.
console.log(window === this); // Its return true.

//Calling a Function in JavaScript
//In the below example, ‘this’ keyword remains the global object if we are calling a
function.

window.sms = "Hi, I am window object";

// Creating a function
function thisMethod() {
    console.log(this.sms); // Hi, I am window object.
    console.log(window === this); //Its return true.
}

// Calling a function
thisMethod();

//Calling Object Methods in JavaScript
//In the below example, when we calling an object constructor or any of its methods,
//‘this’ keyword refers to the instance of the object.

window.sms = "Hi, I am window object";

function objectTestMethod() {
    this.sms = "Hi, I am a test object.";
    this.method1 = function () {
        console.log(this.sms); // Hi, I am a test object.
    };
}

objectTestMethod.prototype. method2 = function () {
    console.log(this.sms); // Hi, I am a test object.
};

// Creating object and calling methods

var v = new objectTestMethod();
v. method1();
v. method2();
```

Why "self" is needed instead of "this" in JavaScript?

Anil Singh 6:00 AM

In this post, I am share the "Why **self** is needed instead of **this** in JavaScript?" and also what is the advantage of using "**var self = this**;"

```
var self = this;
```

In the JavaScript, "self" is a pattern to maintaining a reference to the original "this" keyword and also we can say that this is a technique to handle the events.

Right now, "**self**" should not be used because modern browsers provide a "**self**" as global variable (**window.self**).

Example ["self" keyword is needed instead of "this"],

```
var employee = function (name) {
   var self = this;
   self.username = name;
};
```

Stayed Informed - How do you define a class and its constructor?

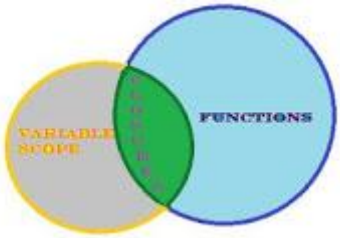
Reference,
http://stackoverflow.com/questions/17163248/whats-the-advantage-of-using-var-self-this-in-knockout-js-view-models

Closure in JavaScript

Anil Singh 5:08 AM

What is closure in JavaScript?

While you create the JavaScript function within another function and the inner function freely access all the variable of outer function. i.e.



How to create closure in JavaScript?

```
function ourterFun(i) {

    var var1 = 3;

    function innerFun(j) {
        console.log(i + j + (++var1)); // It will return the 16.
    }
    innerFun(10);
}

ourterFun(2); // Pass an argument 2
```

The output will get 16 because *innerFun()* function can access to the argument "i" & variable "*var1*" but both are define in the *outerFun()* function that is closure.

That means simply accessing variable outside of your scope create a closure.

```
// OR Other WAYS

function ourterFun(i) {
    var var1 = 3;

    return function (j) {
        console.log(i + j + (++var1)); // It will return the 16.
    }
}

var innerFun = ourterFun(2); // innerFun() function is now a closure.

innerFun(10);
```

How do you define a class and its constructor?

Anil Singh 10:47 PM

A JavaScript **Constructor** is a function which is used as a constructor that allows you to create an object from a class. A constructor access is one of public, protected and private.

Three ways to define a **JavaScript class**,

1. Using as a function
2. Using as Object Literals
3. Singleton using as a function

Stayed Informed - Why "**self**" is needed instead of "**this**" in JavaScript?

The Syntax

```
access NameOfAClass(parameters) {
   initialization code;
}
//Access as public
public customer(parameters) {
   initialization code;
}
//Access as protected
protected customer(parameters) {
   initialization code;
}
//Access as private
private customer(parameters) {
   initialization code;
}
```

Examples as,

```
//class declaration inJavaScript
//constructor
var customer = function (id, name, age) { }
customer.prototype = {}

//Instance variables members.
var customer = function (id, name, age) {
    this.id = id;
    this.name = name;
    this.age = age;
}
customer.prototype = {}


//Static variables.
var customer = function (id, name, age) {
    this.id = id;
    this.name = name;
    this.age = age;
}

customer.prototype = {
    staticVar1: 20,
    staticVar2: 'Anil'
}
```

```
//Instance functions methods.
var customer = function (id, name, age) {
    this.id = id;
    this.name = name;
    this.age = age;
}

customer.prototype = {
    getId: function () {
        return this.id;
    },

    setId: function (id) {
        this.id = id;
    }
}

//Static functions
var customer = function (Id, name, age) {
    this.id = id;
    this.name = name;
    this.age = age;
}

customer.create = function (Id, name, age) {
    return new customer(Id, name, age);
}
customer.prototype = {}
```

Stayed Informed - 39 Best Object Oriented JavaScript QA

What happens when a constructor is called?

1. It creates a new object.
2. It sets the constructor property of the object to Vehicle.
3. It sets up the object to delegate to customer.prototype.
4. It calls customer() in the context of the new object.

difference between == and === in JavaScript

Anil Singh 9:00 AM

In this post, I am going to share the very interesting double equals "==" and triple equals "===". It is very confusing topic and most of the time peoples are confused.

The details example as given below.

The double equals (==) are used for check only value of its variables but triple equals (===) are used for check value and type as well of its variables.

1. The double equal "==" is an auto-type conversion and it checks only value not type.

2. The triple equal "===" is not auto-type conversion and it check value and type both.

For example,

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" />
  <title>Difference between == and === in JavaScript</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.0.0/jquery.js"></script>
  <script>
    // alert(0 == false); // return true, because both are same type.
    // alert(0 === false); // return false, because both are of a different type.
    // alert(1 == "1"); // return true, automatic type conversion for value only.
    // alert(1 === "1"); // return false, because both are of a different type.
    // alert(null == undefined); // return true.
    // alert(null === undefined); // return false.
    //alert('0' == false); // return true.
    // alert('0' === false); // return false.
    // alert(1=== parseInt("1")); // return true.


    console.log(0 == false); // return true, because both are same type.
    console.log(0 === false); // return false, because both are of a different type.
    console.log(1 == "1"); // return true, automatic type conversion for value only.
    console.log(1 === "1"); // return false, because both are of a different type.
    console.log(null == undefined); // return true.
    console.log(null === undefined); // return false.
    console.log('0' == false); // return true.
    console.log('0' === false); // return false.
    console.log(1 === parseInt("1")); // return true.
  </script>
</head>

<body>
  <h3>Difference between == and === in JavaScript</h3>
</body>

</html>
```

difference between call and apply in JavaScript

Anil Singh 2:54 AM

What is the difference between call and apply?

CALL -

Call a function with the specified arguments. You can use call, if you know how many argument are going to pass to the functions.

APPLY -

Call a function with argument provided as an array. You can use apply if you don't know how many argument are going to pass to the functions.

Both (call and apply) are using to call a functions.

Here is an advantage over apply and call. The .call() method is little bit faster than .apply() method.

Go to live demo example http://embed.plnkr.co/4Rb8Ur/preview

Example code as given below.

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8" />
   <title>The Difference Between Call and Apply in JavaScript</title>
   <link rel="stylesheet" href="style.css" />
   <script>
      var var1 = {my: "Obj1" };

      var var2 = {my: "Obj2"};

      function preview(par1, par2) {
         alert(this.my + ' ' + par1 + ' ' + par2);
         console.log(this.my, par1, par2);
      }

      //using call as given below
      preview.call(var1, "First", "Second"); //output: var1 First Second

      //using apply as given below
      preview.apply(var2, ["First", "Second"]); //output: var2 First Second

      //using basic as given below
      preview("First", "Second");//output: undefined "First", "Second"
   </script>
</head>
<body>
   <div>
```

```
    <h3>Refresh the page and see the alert result or go to console window and see the result.</h3>
  </div>
</body>
</html>
```

The output: go to link http://embed.plnkr.co/4Rb8Ur/preview

Thank you!

Why asynchronous code is important in JavaScript?

Anil Singh 1:35 AM
Explained In detail as given below

- Using Asynchronous code, we are able to execute multiple things/JavaScript call at the same-time without locking the DOM or main thread.
- You can define a global variable for call-back method.
- You can use call-back method because JavaScript is lexical scoped and can't define a variable in the call-back methods.
- The response and data come from the parent's call-back method.
- etc..

Thank you!
For more detail you can go
http://stackoverflow.com/questions/11233633/understanding-asynchronous-code-in-laymans-terms

What is function overloading in JavaScript?

Anil Singh 5:16 AM
There is no real function overloading in JavaScript and it allows to pass any number of parameters of any type.

You have to check inside the function how many arguments have been passed and what is the type arguments using *typeof*.

How to create function overloading in JavaScript?

The example for function overloadingnot supporting in JavaScript as give below.

```
function sum(a, b) {
   alert(a + b);
}

function sum(c) {
   alert(c);
}

sum(3);//The output is 3.
sum(2, 4);//The output is 2.
```

In the JavaScript, when we write more than one functions with same name that time JavaScript consider the last define function and override the previous functions. You can see the above example output for the same.

We can achieve using the several different techniques as give below

1. You can check the declared argument name value is undefined.
2. We can check the total arguments with *arguments.length*.
3. Checking the type of passing arguments.
4. Using number of arguments
5. Using optional arguments like x=x | | 'default'
6. Using different name in the first place
7. We can use the arguments array to access any given argument by using *arguments[i]*

Some random ones

http://www.code-sample.com/2015/04/javascript-interview-questions-answers.html