JAVASCRIPT OBJECT AND PROTOTYPE

# Object

Object is the main data type in JS. Everything in JS is an object and you can create your custom object.

A JS object is a complex variable with properties and methods. In JS, all values except primitive values are objects. Primitive values: string, number, true, false, null and undefined.

Object properties: each property has a name and a value.

Object methods: method is the action of object.

There are many ways to create a JS object:

* Using an object literal

var person = {firstName: “John”, lastName: “Terry”};

* Using JS keyword new

var person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";

* Using an object constructor

function person(first, last) {  
 this.firstName = first;  
 this.lastName = last;  
}  
var myMother = new function(“Scott”, “Emily”);

In this case, keyword this is the object that owns the function.

* Using JS function Object.create(): we will have new item for this case.

*Note: JS objects are mutable (changeable)*

**Mutable objects example**

var john = {firstName: “John”, lastName: “Earn”, age: 10};  
var kent = john; // kent is not a copy of john. john and kent point to same object.  
kent.age = 15; // Both john.age and kent.age is 15

# Properties and Methods

JS object has property and method.

**Properties**

Sample, you have JS object as:

var person = {firstName: “Jack”, lastName: “Kent”, job: “Electric engineer”};

You can access to property job of person by some ways:

person.job = “Information technology engineer”;  
alert(person[“job”]);  
var property = “job”;  
console.log(person[property]);

You can use properties in for … each statement as:

for (property in person) {  
 console.log(person[property]);  
}

You can add new property birthday for person as:

person.birthday = “1/1/1980”;

And you can delete property job of person as:

delete person.job // OR delete person[“job”];

You cannot use job property after delete it without re-creating. The delete keyword does not delete inherited properties, but if you delete a prototype property, it will affect all objects inherited from the prototype.

Property has some attributes: value, enumerable, configurable, and writable.

JavaScript objects inherit the properties of their prototype.

**Methods**

Sample, you have constructor to create object Person with method changeFirstName as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.changeFirstName = changeFirstName;  
  
 function changeFirstName(name) {  
 this.firstName = name;  
 }  
}

To add new method to Person, you need to declare it in the constructor.

You can call method as:

var john = new Person(“John”, “Ricky”);  
john.changeFirstName(“Johnny”);

You can add new method, example changeLastName, into john object as:

john.changeLastName = function(lastName) {  
 this.lastName = lastName;  
}  
john.changeLastName(“Harry”);  
alert(john.lastName);

# Object prototype

A prototype is an object from which other objects inherit properties. Every object has a prototype by default. Prototypes are themselves objects. All JavaScript objects inherit their properties and methods from their prototype.

**Creating a prototype**

The standard way to create an object prototype is to use an object constructor function as

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
}

**Modifying a prototype**

You can add new property, nationality, to a prototype as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.nationality = “”;  
}

To add new method, you need modify constructor as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 …  
 this.changeFirstName = function(firstName) {  
 this.firstName = firstName;  
 }  
}

Or using another way as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.changeFirstName = changeFirstName;  
 …  
 function changeFirstName(firstName) {  
 this.firstName = firstName;  
 }  
}

# Using keyword prototype

Prototype property allows you to add properties and methods to an object.

Sample, you have a constructor as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
  
 this.changeFirstName = function(firstName) {  
 this.firstName = firstName;  
 }  
}

You can use Person.prototype to refer to the prototype of Person. *Note: you cannot use Person.prototype to access to properties and methods in Person object.*

~~document.write(Person.prototype.firstName);~~ // Undefined  
~~Person.prototype.changeFirstName(“John”);~~ // Error

**Adding prototype property and method**

You can add new property, nationality, to prototype as:

Person.prototype.nationality = “”;

You can also add new method, changeLastName, to the prototype as:

Person.prototype.changeLastName = function(lastName) {  
 this.lastName = lastName;  
}

**Deleting prototype property and method**

You can delete property from prototype as:

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
}  
  
Person.prototype.yearAge = 1970;  
var john = new Person(“John”, “Kerry”);  
document.write(john.yearAge); // Output 1970  
delete john.yearAge;  
document.write(john.yearAge); // Output 1970, inherited from Person.prototype  
delete Person.prototype.yearAge;  
document.write(john.yearAge); // Output undefined

In above example, when you delete property (yearAge) from an instance (john) of prototype (Person) then you still can access to the property of instance via prototype. Because the yearAge property of Person prototype still exists.

Another example

function Person(firstName, lastName) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
}  
  
var john = new Person(“John”, “Kerry”);  
delete Person.prototype.firstName;  
document.write(john.firstName); // Output John

Deleting property prototype after creating new instance will not effect to new instance.

You can delete methods from prototype as well as deleting properties.

# Inheritance with prototype

**Inheriting from one object**

Sample, you have Shape and Rectangle objects. Rectangle can use properties/methods from Shape. You can you prototype for inheritance in this case as below code:

function Shape() {  
 this.x = 0;  
 this.y = 0;  
}  
Shape.prototype.move = function(x, y) {  
 this.x += x;  
 this.y += y;  
};  
function Rectangle() {  
 Shape.call(this);  
}

Rectangle.prototype = Object.create(Shape.prototype);  
var rect = new Rectangle();  
rect.move(5, 5);  
document.write(rect.x);

With this style, Rectangle object can use all properties/methods from Shape object.

The Object.create() method creates a new object with the specified prototype object and properties.

**Inheriting from multiple objects**

Sample, you have one class that inherits from SuperClass and OtherSuperClass as:

function MyClass() {  
 SuperClass.call(this);  
 OtherSuperClass.call(this);  
}

MyClass.prototype = Object.create(SuperClass.prototype); // Inherit  
MyClass.prototype.constructor = MyClass; // Hand back the constructor  
mixin(MyClass.prototype, OtherSuperClass.prototype); // Mixin

MyClass.prototype.myMethod = function() {  
 // Do a thing  
};

Need to clarify/check this example

# Inheritance example

I have a basic object – Person. I have two children objects – Student and Worker.

Code to create Person object as:

/\* Creating example about Inheritance in JS \*/  
function Person(fullName) {  
 this.fullName = fullName;  
};

Person.prototype.sayHello = function() {  
 alert("Hello, my name is " + this.fullName + ".");  
};

Code to create Student object as:

function Student(fullName, subject) {  
 Person.call(this, fullName);  
 this.subject = subject;  
};

Student.prototype = Object.create(Person.prototype); // Inheritance from Person  
Student.prototype.constructor = Student; // Separating Student from Person

Student.prototype.sayHello = function() { // Overriding methods  
 alert("Hello, my name is " + this.fullName + ". I'm studying " + this.subject + ".");  
};

Student.prototype.study = function() { // Extending methods  
 alert(this.fullName + " is studying.");  
};

*Student* can use all properties (*fullName*) and methods (*sayHello*) from *Person*. It can override methods (*sayHello*). It also can extend new methods (*study*). *Person* does not have *study* method.

If *Student* does not override *sayHello* method then one instance of *Student* can call *sayHello* from *Person* object.

An instance of *Student* can call method *study*. An instance of *Person* object cannot call *study* method.

Code to create *Worker* object as:

function Worker(fullName, companyName) {  
 Person.call(this, fullName);  
 this.companyName = companyName;  
};

Worker.prototype = Object.create(Person.prototype);  
Worker.prototype.constructor = Worker;

Worker.prototype.sayHello = function() {  
 alert("Hello, my name is " + this.fullName + ". I'm working for " + this.companyName + ".");  
};

Certainly, an instance of *Worker* object cannot call method *study* from *Student*.

To test these objects, we create code as:

var janet = new Person("Janet Picky");  
var peter = new Student("Peter Doe", "Applied Physics");  
var jimmy = new Worker("Jimmy Jack", "Vinh Phat Constructor");

var people = [];  
people.push(janet);  
people.push(peter);  
people.push(jimmy);

for (var i = 0; i < people.length; i ++) {  
 people[i].sayHello();  
}

*people[0]* calls method from *Person*. *people[1]* calls method from *Student* and *people[2]* calls method from *Worker*.

*Note: janet instanceof Person, peter instanceof Person and Student, jimmy instanceof Person and Worker.*

# Inheritance from multiple objects example

I have two objects *Button* and *Rectangle*. I create new object *RectangleButton* that inherits from both objects.

The code to create *Button* object as:

var Button = function(value) {  
 this.value = value;  
}

var methodsButton = function() {  
 this.onClick = function() {  
 alert("You click on button " + this.value);  
 }  
}  
methodsButton.call(Button.prototype);  
var btnNew = new Button(“New”);

*btnNew* instanceof *Button* and has methods *onClick*.

The code to create *Rectangle* object as:

var Rectangle = function(height, width) {  
 this.height = height;  
 this.width = width;  
}

var methodsRectangle = function() {  
 this.describe = function() {  
 alert("Rectangle height " + this.height + " width " + this.width);  
 }  
}  
methodsRectangle.call(Rectangle.prototype);  
var rec = new Rectangle(5, 15);

*rec* instanceof *Rectangle* and has methods *describe*.

The code to build RectangleButton as:

var RectangleButton = function (height, width, value) {  
 this.height = height;  
 this.width = width;  
 this.value = value;  
}

methodsButton.call(RectangleButton.prototype);  
methodsRectangle.call(RectangleButton.prototype);

To test RectangleButton, run code as:

var btnSave = new RectangleButton(10, 50, "Save");  
btnSave.onClick();  
btnSave.describe();

*Note: btnSave instanceof RectangleButton but not instanceof Button and Rectangle.*