Introduction to Object-Oriented JavaScript

Perspx, 5 Aug 2008

4.92 (36 votes) Rate this:

An introduction to writing object-oriented structures in JavaScript

Download examples - 7.04 KB

Contents

- Introduction
  - Prerequisites
  - Key Terms
- A Simple Class in JavaScript
  - Defining a Class
  - Creating Class Properties
  - Creating Class Methods
- Encapsulation
  - Public, Protected and Private Members
  - Encapsulation in Practice
  - Conclusion to Encapsulation
- Inheritance
  - Inheriting Properties
  - Inheriting Methods
  - Creating an Instance of an Inherited Class
  - Conclusion to Inheritance
- Polymorphism
  - Conclusion to Polymorphism
- Conclusion
- History

Introduction

Many JavaScript programmers overlook or do not know of the ability to write object-oriented JavaScript. Whilst not conventionally an object-oriented language, JavaScript is a prototype-based language, which means that inherited classes are not derived directly from a base class, but rather are created by cloning the base class which serves as a prototype. This can be used to one's advantage in implementing encapsulation, inheritance and polymorphism in JavaScript, therefore creating a sense of object-orientation.

Object-oriented JavaScript also has several advantages. As it is an interpreted language, it means that methods and properties can be added to the class at any time, and do not have to be declared in the class constructor, like other object-oriented languages such as C++. As JavaScript supports variable data types, class properties do not have to take a fixed data type (such as `boolean` or `string`) but rather can be changed at any time. Furthermore, object-oriented JavaScript is more flexible and efficient than procedural JavaScript, as objects fully support encapsulation and inheritance and polymorphism can be implemented using the `prototype` property.

Prerequisites

Although this is an introductory article to object-oriented JavaScript, it would be beneficial to have an understanding of object-oriented programming, as this article does not go into this aspect in too much detail. However, a list of key object-oriented programming terms are listed and defined below for some guidance in this respect.

Key Terms

Several key terms will be used in this article which are summarized below:

- Class: Definition of an object including it's methods and properties.
- Encapsulation: This is where the data passed around inside an instance of an object is kept contained within the instance of that object. When a new instance of the object is created, a new set of data for that instance is created.
- Inheritance: Where an object becomes a "child" object or subclass of another class, and the "parent" class's properties and methods are applied to the subclass.
- Polymorphism: Where a subclass of a class can call the same generic inherited function in its own context.
- Property: A variable associated with a class.
- Method: A function associated with a class.

A Simple Class in JavaScript

Defining a Class

A basic class can be implemented very easily in JavaScript. All that must be done in order to define a class is for a `function` to be declared:

Hide Copy Code

```
<script language="Javascript">
..

function MyClass()
{
}

..
</script>
```

These three lines of code create a new object called `MyClass`, instances of which can be created with the `new` operator, such as:

Hide Copy Code

```
var c = new MyClass();
```

The function `MyClass` also acts as the class constructor, and when a new instance of that class is called with the `new` operator, this function is called.

Creating Class Properties

This code so far is just a simple class with only its constructor declared. To add properties to the class, we use the `this` operator, followed by the name of the property. As previously stated, methods and properties can be created anywhere in JavaScript, and not just in the class constructor. Here is an example of adding properties to `MyClass`.

Hide Copy Code

```
..

function MyClass()
{
  this.MyData = "Some Text";
  this.MoreData = "Some More Text";
}

..
```

These properties can be accessed by:

Hide Copy Code

```
var c = new MyClass();
alert(c.MyData);
```

This piece of code adds the property `MyData` and `MoreData` to the class. This can be accessed anywhere within the class constructor and the class methods using the `this` operator, so `MyData` can be accessed by using `this.MyData`. Also note that unlike some object-oriented languages, class properties are accessed with a `.` and not a `->`. This is because JavaScript does not differentiate between pointers and variables. If the class reference is stored in a variable when created, then the class properties can be accessed by the variable name followed by a `.` then the name of the class property, in this example `myData`, which is accessed with `c.MyData`.

Creating Class Methods

As said earlier in the article, class methods are created using the `prototype` property. When a method is created in a class in JavaScript, the method is added to the class object using the `prototype` property, as shown in the following piece of code:

Hide Copy Code

```
..

function MyClass()
{
  //Any properties created here
}

MyClass.prototype.MyFunction = function()
{
  //Function code here
}

..
```

For clarity, the method here is created by using `= function()`. Equally, the method can be created by declaring `function MyClass.prototype.MyFunction()`. What this code does is make `MyFunction` a method of `MyClass` using the `prototype` property. This then gives `MyFunction` access to any other methods or properties created in `MyClass` using the `this` operator. For example:

Hide Copy Code

```
..

function MyClass()
{
  this.MyData = "Some Text";
}

MyClass.prototype.MyFunction = function(newtext)
{
  this.MyData = newtext;

  alert("New text:\n"+this.MyData);
}

..
```

This piece of code creates the `MyClass` class, then creates a property called `MyData` in the class constructor. A method, called `MyFunction` is then added to the `MyClass` object, using the `prototype` operator, so that it can access the `MyClass` properties and methods. In this method, `MyData` is changed to `newtext`, which is the only argument of the method. This new value is then displayed using an alert box.

Encapsulation

Encapsulation is a useful part of object-oriented programming that "encapsulates" or contains data in an instance of a class from the data in another instance of the same class. This is why the `this` operator is used within a class, so that it retrieves the data for that variable within that instance of the class.

Public, Protected and Private Members

Encapsulation is implemented in JavaScript by separating instance data within a class. However, there is no varying scale of encapsulation through the `public`, `protected` and `private` operators. This means that access to data cannot be restricted, as seen in other object-oriented programming languages. The reason for this is that in JavaScript it is simply not necessary to do so, even for fairly large projects. As of this, class properties and methods can be accessed from anywhere, either inside the class or outside of it.

Encapsulation in Practice

An example of encapsulation can be shown below:

Hide Copy Code

```
..

function MyClass()
{
  this.MyData = "Some Text";
}

MyClass.prototype.MyFunction = function(newtext)
{
  this.MyData = newtext;

  alert("New text:\n"+this.MyData);
}

..

var c = new MyClass();
```

```
c.MyFunction("Some More Text");

var c2 = new MyClass();
c2.MyFunction("Some Different Text");
```

If called, `c.MyData` would return `"Some More Text"` and `c2.MyData` would return `"Some Different Text"`, showing that the data is encapsulated within the class.

Conclusion to Encapsulation

Encapsulation is an important part of object-oriented programming, so that data in different instances of a class are separate from one another; this is implemented in JavaScript by using the `this` operator. However, unlike other object-oriented programming languages, JavaScript does not restrict access to data within an instance of a class.

Inheritance

Inheriting Properties

As said earlier in the article, there is no direct inheritance in JavaScript, as it is a prototype language. Therefore, for a class to inherit from another class, the `prototype` operator is used, to clone the parent class constructor, and in doing so, inheriting its methods and properties. The parent class constructor is also called in the subclass's constructor, to apply all of its methods and properties to the subclass, as shown in the code below:

Hide Copy Code

```
..

//Parent class constructor
function Animal(name)
{
  this.name = name;
}

//Inherited class constructor
function Dog(name)
{
  Animal.call(this, name);
}
Dog.prototype = new Animal();

Dog.prototype.ChangeName(newname)
{
  this.name = newname;
}

..
```

In the code example above, two classes are created — a base class called `Animal` and a subclass called `Dog` which inherits from `Animal`. In the base class constructor, a property called `name` is created, and set a value passed to it.

When an inherited class is constructed, two lines of code are needed to inherit from the base class, as demonstrated with `Dog`:

Hide Copy Code

```
Animal.call(this, name);
```

This line of code is called from within the subclass's constructor. `call()` is a JavaScript function which calls a function in the specified context (the first argument). Arguments needed by the called function are passed also, starting from the second argument of `call()`,

as seen with `name`. What this means is that the base class constructor is called from within the subclass constructor, therefore applying the methods and properties created in `Animal` to the subclass.

The second line of code needed to inherit from a base class is:

Hide Copy Code

```
Dog.prototype = new Animal();
```

What this line of code does is set the prototype for the inherited class (which will clone the parent constructor when used) to be a new instance of the parent class, therefore inheriting any methods or properties in methods in the subclass.

Notice also that once a subclass has inherited from a parent class, any data that needs to be accessed from the parent class can be accessed using the `this` operator, from within the subclass, as the methods and properties are now part of the subclass object.

Inheriting Methods

Like properties, methods can also be inherited from a parent class in JavaScript, similar to the inheritance of properties, as shown below:

Hide Shrink Copy Code

```
..

//Parent class constructor
function Animal(name)
{
  this.name = name;
}

//Parent class method
Animal.prototype.alertName = function()
{
  alert(this.name);
}

//Inherited class constructor
function Dog(name)
{
  Animal.call(this, name);

  this.collarText;
}
Dog.prototype = new Animal();

Dog.prototype.setCollarText = function(text)
{
  this.collarText = text;
}

..
```

An inherited method can be called as so:

Hide Copy Code

```
var d = new Dog("Fido");
d.alertName();
```

This would call `alertName()`, which is an inherited method of `Animal`.

Creating an Instance of an Inherited Class

Classes that inherit from another class can be called in JavaScript as a base class would be called, and methods and properties can be called similarly.

Hide Copy Code

```
var d = new Dog("Fido");      //Creates an instance of the subclass
alert(d.name);                //Retrieves data inherited from the parent class
d.setCollarText("FIDO");      //Calls a subclass method
```

Methods that are inherited in a subclass can also be called similarly with the variable name in place of the `this` operator:

Hide Copy Code

```
var d = new Dog("Fido");      //Creates an instance of the subclass
d.alertName();                //Calls the inherited method
```

Conclusion to Inheritance

Inheritance is the one of three important concepts of object-oriented programming. It is implemented in JavaScript using the `prototype` and `call()` functions.

Polymorphism

Polymorphism is an extension on the principle of inheritance in object-oriented programming and can also be implemented in JavaScript using the `prototype` operator. Polymorphism is where a subclass of a class can call the same generic inherited function in its own context. For example:

Hide Shrink Copy Code

```
..

//Parent class constructor
function Animal(name)
{
  this.name = name;
}

Animal.prototype.speak = function()
{
  alert(this.name + " says:");
}

//Inherited class "Dog" constructor
function Dog(name)
{
  Animal.call(this, name);
}

Dog.prototype.speak = function()
{
  Animal.prototype.speak.call(this);

  alert("woof");
}

//Inherited class "Cat" constructor
function Cat(name)
{
  Animal.call(this, name);
}
```

```
Cat.prototype.speak = function()
{
  Animal.prototype.speak.call(this);

  alert("miaow");
}

..
```

This code means that if an instance of `Dog` is called and then Dog's `speak()` function is called, it will override the parent class's `speak()` function. However, although we want to do something different with each subclass's version of `speak()`, we want to call the parent class's generic `speak()` function, through `Animal.prototype.speak.call(this);`, which calls the inherited function in the context of the subclass. Then after that we do something else with it, which for `Dog` is `alert("woof");` and for `Cat` is `alert("miaow");`

If called, this would look like:

Hide Copy Code

```
var d = new Dog("Fido");      //Creates instance of Dog
d.speak();                    //Calls Dog's speak() function

var c = new Cat("Lucy");      //Creates instance of Cat
c.speak();                    //Calls Cat's speak() function
```

The first two lines would alert "Fido says:" (the parent class's `speak()` function) followed by "woof" (`Dog`'s `speak()` function).

The second two lines would alert "Lucy says:" (the parent class's `speak()` function) followed by "miaow" (`Cat`'s `speak()` function).

Conclusion to Polymorphism

Polymorphism is a very useful part of object-oriented programming, and whilst in this article it is not pursued too deeply, the principles remain constant and can be applied like this in most aspects of polymorphism in JavaScript.

Conclusion

After reading this article you should be able to:

- Create classes with methods and properties
- Create inherited classes
- Create polymorphic functions

This is just an introductory article, but I hope you can use these learned skills for more complicated object-oriented structures.

History

- 20th July, 2008: Unmodified first copy
- 4th August, 2008: Edited the **Encapsulation** section

License

Share

About the Author

Perspx
United Kingdom

My Blog

Perspx, or Alex Rozanski, is a teenage programmer currently residing in London. He has been programming for approximately 8 years.

He now spends most of his time with web development - trying to maintain sanity with browser incompatibilities - and dabbling with Cocoa, writing applications for Mac OS X in Objective-C. He plans to also broaden his horizons to the iPhone platform in the near future.

Between studying and programming he plays the guitar, enjoys a good horror film, heavy metal music, and tries to plan his future career path.

You may also be interested in...

Writing Object-Oriented JavaScript Part 1

Generate C# Client API for ASP.NET Web API

Object-Oriented ASP.NET

Introduction to Object-Oriented Tiered Application Design

CorePlus: a Microsoft Bot Framework v4 template

Closures in JavaScript

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

Superb Explanation
VinayGandhi 24-Feb-15 18:41

Very good explanation with example. it helps lot to developers.

Sign In · View Thread

my vote of 5
Paulo Augusto Kunzel 21-Oct-13 2:04

Very good article,
It is always good to have this kind of article as a reference on how to do things.
Thx

Sign In · View Thread

My vote of 5
Umair Feroze 6-Jan-11 0:08

Its superb and very handy

Sign In · View Thread

Nice article!
Sandeep Mewara 23-May-10 19:51

Well written...

Sign In · View Thread

Please help
Mohammad Dayyan 17-Sep-08 2:40

Hi there Perspx.
I read your article .

But I couldn't understand this script :

Hide Copy Code

```
$('sound_demo_play').observe('click', function(event) {
       event.stop();
       Sound.play($('sound_demo_track_url').value,{replace:true});
  });
```

Reference of above script : http://github.com/madrobby/scriptaculous/wikis/sound[^]

What's the substitute of this script ?

Can you help me ?
Thanks in advance.

Sign In · View Thread

Re: Please help

Perspx 17-Sep-08 6:10

Hi Mohammad,

The script you have referenced is a bit different, as it is written with the *Scriptaculous* JavaScript framework, (which in itself is based on the *Prototype* framework) which is designed to make JavaScript coding much easier. Let me help by breaking the code down for you:

The dollar function, `$`, is used in the Prototype framework to reference an element with an ID, like the JavaScript `GetElementById()` function so `$('sound_demo_play')` references the element with the ID *sound_demo_play*.

The `observe` function assigns an event to an element, in this case, the element with the ID *sound_demo_play*. The first argument is *'click'*, which means that the event function will be called when the element is clicked on.

The second argument is the event function itself, which will be called when the element (in this case) is clicked on. Inside this function, two things happen:

- The first thing that happens is the `event.stop()` method. What this does is prevent the browser performing the default action associated with the element; in this case the event element is a link, so this stops the browser navigating away from the page.
- The second thing that happens is that a sound is played, using the Scriptaculous `Sound` object. The first argument is the URL of the sound, (which is taken from the value of the element with the ID *sound_demo_track_url*) and the second argument sets the `replace` property to `true`, which stops any sound that is currently playing in the browser.

Hope this helps!
--Perspx

Don't trust a computer you can't throw out a window

-- Steve Wozniak

Sign In · View Thread 5.00/5 (1 vote)

Re: Please help [modified]
Mohammad Dayyan 17-Sep-08 11:26

Nice answer.
Thank you very much Perspx.

modified on Sunday, September 21, 2008 9:42 AM

Sign In · View Thread

Great
Abhishek Sur 15-Aug-08 10:28

Great Article for Newbie.. I must state that ...

Great work Perspx

Abhishek

Sign In · View Thread

Re: Great
Perspx 15-Aug-08 10:52

Thank you

Regards,
-- Perspx

"The Blue Screen of Death, also known as **The Blue Screen of Doom**, the **"Blue Screen of Fun"**, **"Phatul Exception: The WRECKening"** and **"Windows Vista"**, is a multi award-winning game first developed in 1995 by Microsoft" - Uncyclopedia

Introduction to Object-Oriented JavaScript

Sign In · View Thread

Great article :)
Srinath Gopinath 6-Aug-08 21:29

Very useful....

Sign In · View Thread

Re: Great article :)

Perspx 6-Aug-08 22:20

Thanks

Regards,
--Perspx

"The Blue Screen of Death, also known as **The Blue Screen of Doom**, the **"Blue Screen of Fun"**, **"Phatul Exception: The WRECKening"** and **"Windows Vista"**, is a multi award-winning game first developed in 1995 by Microsoft" - Uncyclopedia

Sign In · View Thread

Thank you. It was helpful
Mohammad Dayyan 2-Aug-08 11:12

Its great.
Thank you Perspx.

Sign In · View Thread

Re: Thank you. It was helpful
Perspx 4-Aug-08 5:33

Glad you found it useful

Regards,
--Perspx

"The Blue Screen of Death, also known as **The Blue Screen of Doom**, the **"Blue Screen of Fun"**, **"Phatul Exception: The WRECKening"** and **"Windows Vista"**, is a multi award-winning game first developed in 1995 by Microsoft" - Uncyclopedia

Sign In · View Thread 5.00/5 (1 vote)

Encapsulation in JavaScript?
Steve_ 29-Jul-08 0:48

I disagree with your conclusion from the encapsulation section. JavaScript does not implement encapsulation in a conventional way, it doesn't implement it at all! Encapsulation is defined as "A method for protecting data from unwanted access or alteration". In the case of JavaScript, any member variable can be accessed from the outside directly, so where is the protection?

Sign In · View Thread

Re: Encapsulation in JavaScript?
Perspx 29-Jul-08 10:49

What I was saying was more to do with the fact that Javascript separates data from instance-to-instance of a class, doing this similarly to other object-oriented languages. However, you are indeed right in that Javascript does not offer data protection "from unwanted access or alteration", but my summary on private, protected and public members/properties is aimed at explaining this. I see what you are saying and you are indeed right, but I meant it to mean more about separating data. Apologies for not making this clear.

Regards,
--Perspx

"When programming in Visual Basic, you can always know whether a given program will become stuck in a loop and never halt. The answer is 'yes'." - Uncyclopedia

Sign In · View Thread

Re: Encapsulation in JavaScript?
Steve_ 30-Jul-08 1:40

I should probably have phrased my comment better. I wasn't trying to pick holes in your article, on the whole I think it is quite useful. My issue is that I know a lot of less experienced or beginner developers read this site. Encapsulation is one of the fundamental concepts of OOP and I don't want them to misunderstand what it actually is.

Sign In · View Thread

Re: Encapsulation in JavaScript?
Perspx 30-Jul-08 3:14

Oh of course, I understand. Thanks for letting me know - I'll be sure to update that section to make this more clear.

Thanks for your feedback.
--Perspx

"The Blue Screen of Death, also known as **The Blue Screen of Doom**, the **"Blue Screen of Fun"**, **"Phatul Exception: The WRECKening"** and **"Windows Vista"**, is a multi award-winning game first developed in 1995 by Microsoft" - Uncyclopedia

Sign In · View Thread

Re: Encapsulation in JavaScript?
Kamarey 3-Aug-08 20:56

You can hide class members like this

```
function Dog
{
var name;

this.GetName = function() { return name; }
this.SetName = function(newName) { name = newName; }
}
```

You can't change name without calling SetName. So I think there is an encapsulation in javascript. Not a such one as in C++ or other languages, but it exists.

Sign In · View Thread

Re: Encapsulation in JavaScript?
Perspx 4-Aug-08 0:47

Yes, you are correct, that is a way to implement encapsulation in Javascript

Thanks for your input,
--Perspx

"The Blue Screen of Death, also known as **The Blue Screen of Doom**, the **"Blue Screen of Fun"**, **"Phatul Exception: The WRECKening"** and **"Windows Vista"**, is a multi award-winning game first developed in 1995 by Microsoft" - Uncyclopedia

Sign In · View Thread

very usefull
Abhijit Jana 23-Jul-08 19:46

Thansk for sharing in codeproject .

full from me

cheers,
Abhijit

Sign In · View Thread

Re: very usefull
Perspx 23-Jul-08 22:22

No problem

--Perspx

"When programming in Visual Basic, you can always know whether a given program will become stuck in a loop and never halt. The answer is 'yes'." - Uncyclopedia

Sign In · View Thread

Nice introduction to javascript oop
Cybercockroach 22-Jul-08 15:49

hi Perspx
This is a nice and short introduction to whom are not familiar with javascript oop. Thanks

Sign In · View Thread

Re: Nice introduction to javascript oop
Perspx 22-Jul-08 20:49

I'm glad you find it useful

--Perspx

"When programming in Visual Basic, you can always know whether a given program will become stuck in a loop and never halt. The answer is 'yes'." - Uncyclopedia

Sign In · View Thread

Re: Nice introduction to javascript oop
MarkieW 22-Jan-10 5:15

I agree!

Sign In · View Thread

Better than a copy / paste.....
RichardGrimmer 22-Jul-08 12:01

But re-writing a recent MSDN article isn't really worthwhile is it?

C# has already designed away most of the tedium of C++.

Sign In · View Thread

Refresh 1 2 Next »

General News Suggestion Question Bug Answer Joke Praise Rant Admin