

OO in JavaScript.

The article by Sergey Zavadski introduces the object model of the JavaScript programming language and demonstrates common practices in the OOP (object oriented programming) with the JavaScript.

Simplicity of the JavaScript.

JavaScript is proven to be the one of the simplest programming languages to learn and use. Minor snippet of the code (in fact one string) performs various actions like creating of the windows or changing the text in the status bar. The flexibility, short learning curve and the fact that JavaScript is not "strongly typed" language explains extreme popularity of the JavaScript among the web masters community which selected this language as the language of choice to provide the static HTML pages with the dynamic attractive content.

The powerful tool.

However in spite of its visible simplicity the JavaScript, when following the DOM specifications, is quite advanced and powerful tool providing all the required facilities for the creation of the complex solutions (client side) such as menus, trees, grids. These possibilities are actually reflecting the JavaScript's power when it comes to the layers manipulations. Not the last reason for this power is OOP structure of the JavaScript programming language which was planned and implemented as an object oriented language!

Object model.

The JavaScript object is the entity with the properties that can be either other objects or variables. For example we have 'Country_Italy' object. We can assign the values to the properties of this object like this:

```
Country_Italy.Name="Italy";  
Country_Italy.Capital="Rome";  
Country_Italy.Area=301000;
```

However the 'Country_Italy' object itself can be the property of other object:

```
Europe.MostBeautifulCountry= Country_Italy;
```

JavaScript comes with the library of the built-in objects like 'Window', 'Math', 'String' and many others. But what makes the language really flexible is the ability of creating own custom objects. There are two possible methods for making custom objects in the JavaScript. First one is using of the direct initialization. The code in this case may look like this:

```
Country={Name:"France",Capital:"Paris",Government:{President  
:"Jacques Chirac"}};
```

We have created the Country object with the 'Name', 'Capital' and 'Government' properties and the 'Government' property is the object with own list of properties. We've also initialized these properties with the initial values. All is done in the one single string. The following construction creates and initializes the empty object:

```
Country={};
```

Another way to create objects in the JavaScript is defining the special function constructor for the object and initialization of the object with the 'new' operator. For example:

```
function Country(Name,Capital,Population){
    this.Name=Name;
    this.Capital=Capital;
    this.Population=Population;
}
```

The 'Country' function is the constructor function. Now to create the object using this defined constructor we use the new operator:

```
Country_Italy=new Country("Italy","Rome",301000);
```

Just like in the previous example the property can be the object itself:

```
function Government(President){
    this.President=President;
}

function Country(Name,Capital,Government){
    this.Name=Name;
    this.Capital=Capital;
    this.Government=Government;
}

Government_France=new Government("Jacques Chirac");
Country_France=new
Country("France","Paris",Government_France);
```

Another advantage of the OOP fully implemented in the JavaScript is that the object can have the methods (not only properties). The methods are the functions described within the object scope that usually operate with object's properties. The methods are described much like the properties:

```
function describeCountry(){
    var desc="Name: "+this.Name+" Capital: "+
    this.Capital+" Population: "+this.Population;
    alert(desc);
}
```

In the above definition this operator provides an access to the object the described method belongs to.

```
Country_Italy.describeCountry=describeCountry;
```

The construction below fully repeats the above one:

```
Country_Italy.describeCountry=function(){
    var desc="Name: "+this.Name+" Capital: "+
    this.Capital+" Population: "+this.Population;
    alert(desc);
}
```

The construction:

```
Country_Italy.describeCountry();
```

calls the 'describeCountry' method of the 'Country_Italy' object.

Prototypes based object oriented language.

Unlike some other popular OO languages (Java, C++) whose object model is based on the classes the object model of the JavaScript is based on the prototypes. The main difference between those two approaches is that in the prototype based language there's no difference between the class and the class instance entities - you only deal with the objects. The prototype (can be viewed as a template that defines the way initial object's properties values are assigned) is any object declared with the prototype operator. This object can be the parent of any newly created object and this is how the JavaScript supports inheritance - important OOP language characteristic. Let's see our Country object:

```
function Country(Name,Capital,Population){
    this.Name=Name;
    this.Capital=Capital;
    this.Population=Population;
}
```

Now let's declare 'Cn' - the 'Country' object prototype:

```
Cn=Country.prototype;

Cn.describeCountry=function(){
    var desc="Name: "+this.Name+" Capital: "+
    this.Capital+" Population: "+this.Population;
    alert(desc);
}
```

Now we can call the 'describeCountry' method in any 'Country' descendant with only two strings of code:

```
Country_Italy=new Country("Italy","Rome",301000);
Country_Italy.describeCountry();
```

Using this approach we can define in the prototype any number of the methods and properties. Defining methods and properties in the

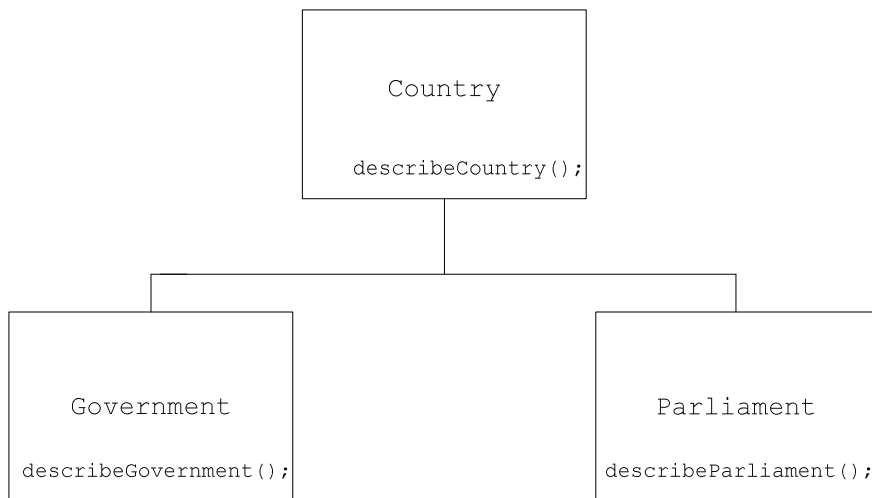
prototype rather than defining them directly for each object gives many advantages:

- You don't have to define all required methods or properties any time the object is created. You define all the necessary methods and properties in the prototype and then all you have to do is to create the object.
- Makes your code more secure since you can keep all object definitions in the separate file
- Makes your code easier to read

Inheritance.

JavaScript also supports the inheritance mechanism that is the cornerstone of the OOP.

To show the easiness of the JavaScript inheritance mechanism let's review two already defined objects of ours 'Country' and 'Government' and one new object 'Parliament'. Let's assume that these objects have the following object hierarchy:



Constructor functions for these objects will look like these:

```
function Country(Name,Capital,Population){
    this.Name=Name;
    this.Capital=Capital;
    this.Population=Population;
}

function Government(President){
    this.President=President;
}

function Parliament(Speaker){
```

```
        this.Speaker=Speaker;
    }
```

Take a look at the 'Country' object prototype.

```
Cn=Country.prototype;

Cn.describeCountry=function(){
    var desc="Name: "+this.Name+" Capital: "+
    this.Capital+" Population: "+this.Population;
    alert(desc);
}
```

The following construction defines 'Country' as 'Government' prototype ('Government' inherits the 'Country'):

```
Government.prototype=new Country("France","Paris",59330000);
```

When 'Government' object is created it inherits all the 'Country' methods and properties. In other words, although 'describeCountry()' method is not defined for 'Government' object it is available since 'Country' object is defined as 'Government' prototype.

```
Government_France=new Government("Jacques Chirac");
Government_France.describeCountry();
```

New methods can also be defined for the 'Government_France' object:

```
Government_France.describeGovernment=function(){
    alert("President: "+this.President);
}
```

Let's complete the 'Parliament' object definitions.

```
Parliament.prototype=new Country("France","Paris",59330000);
```

```
Pn=Parliament.prototype;
Pn.describeParliament=function(){
    alert("Speaker: "+this.Speaker);
}
```

```
Parliament_France=new Parliament("Jean-Louis Debre");
```

As you can see prototypes can also be modified for the inherited objects.

The above examples demonstrates elegance of the JavaScript OOP model which enables this easy to learn language perform variety of the complex tasks for the real life programming.

Real life example.

This very OOP approach was used in our company when we developed the CodeThat.Com's set of the web controls. For example the 'CodeThatCalendar' is nothing but an object declared as:

```
function CodeThatCalendar(def) {
    this.def = def;
    this.links = {};
    this.styles = {};
    this.hideifr = true;

    if (typeof(this.def.img_path)!="undefined") {
        if
        (this.def.img_path.lastIndexOf("\/")!=this.def.img_path.leng
        th-1) this.def.img_path=this.def.img_path+"\>";
    }

    if (typeof(this.def.template_path)!="undefined") {
        if
        (this.def.template_path.lastIndexOf("/")!=this.def.template
        _path.length-1)
        this.def.template_path=this.def.template_path+"/";
        if (this.def.template_path.indexOf("\/")!=0)
            if (typeof(this.def.img_path)!="undefined"
            && this.def.img_path.indexOf("\/")!=0) {
                s=this.def.template_path;
                a=s.split("/");
                a.length=a.length-1;
                t="";
                for (i=0; i<a.length; i++){
                    t=t+"../";
                }
                this.img_path=t+this.def.img_path;
            }
    }
};

var CTc = CodeThatCalendar.prototype;

CTc.hide = function()
{
    ...
}

CTc.create = function(d,ctl) {
```

```
    ...  
}
```

Various methods of the control work with the layers giving the control fancy look and feel and enabling user perform various actions the calendar supposed to do. For example, the below function sets the value of the HTML form control to the date user have selected. This is performed each time user clicks the calendar date.

```
function CodeThatSetDay(c,f,d,m,y,i,ifr) {  
var doc;  
var w = window.opener||this.parent;  
  
if(w&&!i)  
    doc = w.document;  
else  
    doc = document;  
var e = CodeThatFind(doc,c);  
if(Def(e))  
{  
    e.value=CodeThatDateFormat(f,d,m,y);  
    if(e.fireEvent) e.fireEvent("onchange");  
    else {  
        if(e.onchange) e.onchange();  
    }  
}  
  
if(w&&!i)  
{  
  
    if(Def(w) && Def(ifr))  
    {  
        var iframe = CodeThatFind(doc,ifr);  
        if(Def(iframe))  
            iframe.style.visibility = 'hidden';  
        if(ua.opera6)  
        {  
            var d = CodeThatFind(doc,"calendar_div");  
            if(Def(d))  
                d.style.visibility='hidden';  
        }  
    }  
    else  
    {  
        window.close();  
    }  
}  
};
```

(The examples of CodeThatCalendar functions are not fully functional. They are listed here for illustrative purposes).

As you can see JavaScript is indeed powerful and user friendly programming language with many abilities which help to create really complex and interesting solutions. Since the DOM model is supported by majority of modern browsers you can be sure your solution will be accessible for the majority of users. Of course there's much more to learn in the JavaScript: events, DOM, layers, and the cross-browser portability is also not always that transparent but this is for another story or... several stories.

Author.

About the author: Sergey Zavadski is one of the core JavaScript developers at CodeThat.Com (<http://www.codethat.com/>) He has developed number of the scripts offered by the company and leads the customers support department.