JavaScript Best Practices

Sang Shin
http://www.JPassion.com
"Learn with JPassion!"

Topics

- Use Object-oriented JavaScript
- Use Object hierarchy
- Use the prototype property
- Write reusable JavaScript code
- Use Object literals as function parameters
- Load JavaScript on demand
- Clean separation of content, CSS, and JavaScript
- Reduce the size of JavaScript file

Use Object Oriented JavaScript

Use Object Oriented JavaScript

- Provides better reusability of the code
- Enables your objects to be better organized
- Allow for dynamic loading of objects

Example: Cart Object in JavaScript

```
//  The Cart object provides basic support for maintaining
// an internal array of Item objects

function Cart() {
    this.items = [];
}

function Item (id,name,desc,price) {
        this.id = id;
        this.name = name;
        this.desc = desc;
        this.price = price;
}

// Create an instance of the Cart and add items

var cart = new Cart();
cart.items.push(new Item("id-1","Paper","something you write on",5));
cart.items.push(new Item("id-2","Pen", "Something you write with",3));
var total = 0;

for (var I = 0; I < cart.items.length; I++) {
     total = total + cart.items[I].price;
}
```

Example: Comparable Java Code

```
public class Cart {
    private ArrayList items = new ArrayList();
    public ArrayList getItems() {
        return items;
    }
}

public class Item {
    private String id; private String name; private String desc; private double price;

    public Item(String id, String name, String desc, double price) {
        this.id = id;
        this.name = name;
        this.desc = desc;
        this.price = price;
    }

    public String getId() {return id;}
    public String getName() {return name;}
    public String getDesc() {return desc;}
    public float getPrice() {return price;}
}
```

Use Object Hierarchy

Use Object Hierarchies to Organize JavaScript Objects to Avoid Name Collision

- In JavaScript, there is the potential for object names to collide
  - In Java language, package names are used to prevent naming collisions
- JavaScript does not provide package names like Java. However, you can do it yourself as following
  - When writing components, use objects and object hierarchies to organize related objects and prevent naming collision

Example: Create a top level object BLUEPRINTS which acts in a sense like a namespace for related objects

```
// create the base BLUEPRINTS object if it does not exist.
if (!BLUEPRINTS) {
    var BLUEPRINTS = new Object();
}

// define Cart under BLUEPRINTS
BLUEPRINTS.Cart = function () {
   this.items = [];

   this.addItem = function(id, qty) {
       this.items.push(new Item(id, qty));
   }

   function Item (id, qty) {
       this.id = id;
       this.qty = qty;
   }
}

//  create an instance of the cart and add an item
var cart = new BLUEPRINTS.Cart();
cart.addItem("id-1", 5);
cart.addItem("id-2", 10);
```

Use the prototype property

Use the prototype property

- Use it to define shared behavior and to extend objects
- The prototype property is a language feature of JavaScript
  - The property is available on all JavaScript objects

Example: Usage of “prototype” property

```
function Cart() {
        this.items = [ ];
}

function Item (id,name,desc,price)) {
        this.id = id;
        this.name = name;
        this.desc = desc;
        this.price = price;
}

// SmartCart extends the Cart object inheriting its properties and adds
// a total property

function SmartCart() {
        this.total = 0;
}
SmartCart.prototype = new Cart();
```

Example: Usage of “prototype” property

```
// Declare a shared addItem and calcualteTotal function and adds them
// as a property of the SmartCart prototype member.

Cart.prototype.addItem = function(id,name,desc,price) {
        this.items.push(new Item(id,name,desc,price));
}

Cart.prototype.calculateTotal = function() {
        for (var I=0; I < this.items.length; I++) {
            this.total = this.total + this.items[I].price;
        }
        return this.total;
}

// Create an instance of a Cart and add an item

var cart = new SmartCart();
cart.addItem("id-1","Paper", "Something you write on", 5);
cart.addItem("id-1","Pen", "Soemthing you write with", 3);
alert("total is: " + cart.calculateTotal());
```

Write reusable JavaScript

Write reusable JavaScript

- JavaScript should not be tied to a specific component unless absolutely necessary
- Consider not hard coding data in your functions that can be parameterized

Example: Reusable JavaScript function

```
// The doSearch() function in this example can be reused because it is
// parameterized with the String id of the element, service URL, and the <div>
// to update. This script could later be used in another page or application

<script type="text/javascript">
        doSearch(serviceURL, srcElement, targetDiv) {
                var targetElement = document.getElementById(srcElement);
                var targetDivElement = document.getElementById(targetDiv);
                // get completions based on serviceURL and srcElement.value
                // update the contents of the targetDiv element
        }
</script>

<form onsubmit="return false;">
        Name: <input type="input" id="ac_1" autocomplete="false"
                        onkeyup="doSearch('nameSearch','ac_1','div_1')">
        City: <input type="input" id="ac_2" autocomplete="false"
                        onkeyup="doSearch('citySearch','ac_2','div_2')">
        <input type="button" value="Submit">
</form>

<div class="complete" id="div_1"></div>
<div class="complete" id="div_2"></div>
```

Use object literals as flexible function parameters

Object Literals

- Object literals are objects defined using braces ({}) that contain a set of comma separated key value pairs much like a map in Java
- Example
  - > {key1: "stringValue", key2: 2, key3: ['blue','green','yellow']}
- Object literals are very handy in that they can be used as arguments to a function
- Object literals should not be confused with JSON which has similar syntax

Example: Usage of Object Literals as parameters

```
function doSearch(serviceURL, srcElement, targetDiv) {

        // Create object literal
          var params = {service: serviceURL,
                        method: "get",
                        type: "text/xml"};

        // Pass it as a parameter
          makeAJAXCall(params);
}

// This function does not need to change even when params changes
function makeAJAXCall(params) {
          var serviceURL = params.service;
          ...
}
```

Example: Usage of Object Literals as parameters - Anonymous Object Literals

```
function doSearch() {
        makeAJAXCall(
                        {
                          serviceURL: "foo",
                            method: "get",
                            type: "text/xml",
                            callback: function(){alert('call done');}
                          }
                    );
}
function makeAJAXCall(params) {
        var req = // getAJAX request;
        req.open(params.serviceURL, params.method, true);
        req.onreadystatechange = params.callback;
        ...
}
```

Load JavaScript on Demand

Load JavaScript On Demand

- If you have a large library or set of libraries you don't need to load everything when a page is loaded
- JavaScript may be loaded dynamically at runtime using a library such as JSAN or done manually by using AJAX to load JavaScript code and calling eval() on the JavaScript

Example: Load JavaScript On Demand

```
// Suppose the following is captured as cart.js file
function Cart () {
      this.items = [ ];
      this.checkout = function() {
                              // checkout logic
                        }
 }

-----------------------------------------------------------------

 // Det cart.js using an AJAX request and
 // eval on the javascript
 eval(javascriptText);
 var cart = new Cart();

 // add items to the cart
 cart.checkout();
```

Clean separation of Content, CSS, and JavaScript

Separation of content, CSS, and JavaScript

- A rich web application user interface is made up of
  - content (HTML/XHTML)
  - styles (CSS)
  - JavaScript
- Separating the CSS styles from the JavaScript is a practice which will make your code more manageable, easier to read, and easier to customize
- Place CSS and JavaScript code in separate files
- Optimize the bandwidth usage by having CSS and JavaScript file loaded only once

Example: Bad Practice

```
<style>
#Styles
</style>

<script type="text/javascript">
// JavaScript logic
<script>

<body>The quick brown fox...</body>
```

Example: Good Practice

```
<link rel="stylesheet" type="text/css" href="cart.css">
<script type="text/javascript" src="cart.js">

<body>The quick brown fox...</body>
```

Reduce the size of JavaScript file

Reduce the size of JavaScript file

- Remove the white spaces and shortening the names of variables and functions in a file
- While in development mode keep your scripts readable so that they may be debugged easier
- Consider compressing your JavaScript resources when you deploy your application
- If you use a 3rd party JavaScript library use the compressed version if one is provided.
  - Example compression tool: ShrinkSafe

Thank you!

Sang Shin
Michèle Garoche
http://www.javapassion.com
“Learn with Passion!”