JavaScript Refresher

Doris Chen Ph.D.
Senior Technology Evangelist
doris.chen@microsoft.com

Microsoft

Who am I?

- Developer Evangelist at Microsoft based in **Silicon Valley, CA**
  - Blog: http://blogs.msdn.com/b/dorischen/
  - Twitter **@doristchen**
  - Email: **doris.chen@microsoft.com**
- Has over 15 years of experience in the software industry focusing on web technologies
- Spoke and published widely at JavaOne, O'Reilly, Silicon Valley Code Camp, SD West, SD Forum and worldwide User Groups meetings
- Doris received her Ph.D. from the University of California at Los Angeles (UCLA)

Agenda

- Common Gotchas in JavaScript
- How to write good JavaScript: Best Practices and tips

Common Gotchas in JavaScript

Variable Scope: global versus local scope

- **Global**: variables **not** declared with *var*
- Local: variables declared with *var*

```
anonymousFuntion1 = function(){
        globalvar = 'global scope'; // globally declared because "var" is missing.
        return localvar;
}();

alert(globalvar);     // alerts 'global scope' because variable within the function is declared globally

anonymousFuntion2 = function(){
        var localvar = 'local scope';   //locally declared with "var"
        return localvar;
}();

alert(localvar);        // error "localvar is not defined". there is no globally defined localvar
```

Functions in JavaScript are objects

- can be invoked at will and can be passed around to other functions

```
function blah() { console.log("blah"); }
  blah()              //prints blah
  blah.call()         //prints blah
  blah.apply()        //prints blah
```

- Functions as objects in JavaScript

```
  blah.foo()          //TypeError: Object function blah() {
                      console.log("blah"); } has no method 'foo'
```

Classes in JavaScript:
Functions as classes in JavaScript

- an function object with a name and a few properties

```
function Message(to, from, msg){
 this.to = to;
 this.from = from;
 this.msg = msg;

 this.asJSON = function(){
  return "{'to':'" + this.to + "', 'from':'" + this.from + "', 'message':'" +
    this.msg + "'}";
 }
}
```

Classes in JavaScript:
Using classes in JavaScript

```
var message = new Message('Andy', 'Joe', 'Party tonight!');
message.asJSON();
```

Result JSON message

"{'to':'Andy', 'from':'Joe', 'message':'Party tonight!'}"

Primitives and objects

- Number, String, and Boolean
  - var string = "test“ //typeof string
  - var aNumber = 10
  - var anotherNumber = 0.99
  - var aBool = true
  - var notABoolean = "false“
- Object: Array
  - var myArray = ["Hello", 1, true] //typeof myArray is **object**
  - myArray instanceof Array //true

typeof

- Returns type of an instance of a primitive type
  - Array is actually not a primitive type, so the typeof an Array object is Object
- typeof {} // "object"
- typeof "" // "string"
- typeof [] // "array"
- typeof null // “object”
- typeof new Number(123); // "object"
- typeof Number(123); // "number"
- typeof 123; // "number"

instanceof

- If you want to test type of a Boolean, String, Number, or Function, use *typeof*. For anything else, use *instanceof*
- "hello" instanceof String; //false
- new String("hello") instanceof String; //true
- ["item1", "item2"] instanceof Array; //true
- new Array("item1", "item2") instanceof Array; //true

Constructing Built-in Types with the 'new' Keyword

- JavaScript has the types Object, Array, Boolean, Number, String, and Function.
- Each has its own literal syntax and so the **explicit constructor is never required**.

Explicit (bad)	**Literal (good)**
var a = new Object(); a.greet = "hello";	var a = { greet: "hello" };
var b = new Boolean(true);	var b = true;
var c = new Array("one", "two");	var c = ["one", "two"];
var d = new String("hello");	var d = "hello"
var e = new Function("greeting", "alert(greeting);");	var e = function(greeting) { alert(greeting); };

Use === Instead of ==

- two different kinds of equality operators
- == | != performs a comparison with type coercion
  - "" == 0 //true - empty string is coerced to Number 0.
  - 0 == "0" //true - Number 0 is coerced to String "0"
  - "" == "0" //false - operands are both String so no coercion is done.
- ===|!== compare type and value
  - === true
  - !== false

string replace

- string replace function only replaces the **first match**, not all matches as you may expect.

```
"bob".replace("b", "x");    // "xob"
```

- To replace all matches, must use a Regular Expression, and add the global modifier to it

```
"bob".replace(/b/g, "x"); // "xox"
"bob".replace(new RegExp("b", "g"), "x"); // "xox" (alternate explicit RegExp)
```

Undefined ≠ null

- Null is for an object, undefined is for a property, method or variable.
- To be null, object has to be defined.
  - If your object is not defined, and you test to see whether it's null, since it's not defined, it will throw an error.

```
if(myObject !== null  && typeof(myObject) !== 'undefined') {
    //if myObject is undefined, it can't test for null, and will throw an error
}

if(typeof(myObject) !== 'undefined' && myObject !== null) {
    //code handling myObject
}
```

NaN

- type of NaN (Not a Number) is… Number.

```
typeof NaN === "number" //true
NaN === NaN;            // false -- NaN compare to anything is false
```

- only way to test whether a number is equal to NaN is with the helper function isNaN.

Don’t Use Short-Hand

- Technically, get away with omitting most curly braces and semi-colons

```
if(someVariableExists)
   x = false
   anotherfunctionCall();
```

Bad

Interpreted by some browsers

```
if(someVariableExists) {
   x = false;}
   anotherFunctionCall();
```

```
if(someVariableExists) {
  x = false;
  anotherFunctionCall();
}
```

Good

Line Breaks

- Include a hard line break in between quotes will get a parsing error

```
var bad  = '<ul id="myId">
              <li>some text</li>
              <li>more text</li>
            </ul>';                          // unterminated string error

var good = '<ul id="myId">' +
              '<li>some text</li>' +
              '<li>more text</li>' +
           '</ul>'; //correct
```

How to write good JavaScript: Best Practices and tips

Efficiently Structure Markup

Link Style Sheets at Top of Page

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
      <link rel="stylesheet" type="text/css" href="class.css" />
   </head>
   <body>
      …
      …
      …
   </body>
</html>
```

Never Link Style Sheets at Bottom of Page

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
   </head>
   <body>
      …
      …
      …
   </body>
      <link rel="stylesheet" type="text/css" href="styles.css"/>
</html>
```

Only Include Necessary Styles

Efficiently Structure Markup

```
/* These styles are for the home page. */
HomePage
{
  color: red;
}

/* These styles are for the content page. */
ContentPage
{
  color: green;
}
```

Always Link JavaScript at End of File

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
   </head>
   <body>
      …
      …
      …
      <script src="myscript.js" … ></script>
   </body>
</html>
```

Avoid Linking JavaScript In Head

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
      <script src="myscript.js" … ></script>
   </head>
   <body>
      …
      …
      …
   </body>
</html>
```

Avoid Inline JavaScript

Efficiently Structure Markup

```
<html>
  <head>
      <script type="text/javascript">
           function helloWorld() {
                alert('Hello World!') ;
           }
      </script>
  </head>
  <body>
      …
   </body>
</html>
```

Asynchronously Download Script

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
   </head>
   <body>
      …
      …
      …
      <script async src="myscript.js"></script>
   </body>
</html>
```

Remove Duplicate Code

Efficiently Structure Markup

```
<html>
   <head>
      <title>Test</title>
   </head>
   <body>
      …
      <script src="jquery.js" … ></script>
      <script src="myscript.js" … ></script>
      <script src="navigation.js" … ></script>
      <script src="jquery.js" … ></script>
   </body>
</html>
```

Standardize on a Single Framework

Efficiently Structure Markup

```
<script src="jquery.js" … ></script>
<script src="prototype.js" … ></script>
<script src="dojo.js" … ></script>
<script src="animater.js" … ></script>
<script src="extjs.js" … ></script>
<script src="yahooui.js" … ></script>
<script src="mochikit.js" … ></script>
<script src="lightbox.js" … ></script>
<script src="jslibs.js" … ></script>
<script src=“gsel.js" … ></script>
…
```

Don’t Include Script To Be Cool

Efficiently Structure Markup

```
<script src="facebookconnect.js" … ></script>
<script src="facebooklike.js" … ></script>
<script src="facebookstats.js" … ></script>
<script src="tweetmeme.js" … ></script>
<script src="tweeter.js" … ></script>
<script src="tweetingly.js" … ></script>
<script src="googleanalytics.js" … ></script>
<script src="doubleclick.js" … ></script>
<script src="monitor.js" … ></script>
<script src="digg.js" … ></script>
…
```

Write Good and Fast JavaScript

Eval = Bad

`eval``(`*string*`)`

- **`eval`** function compiles and executes a string and returns the result
  - gives us access to JavaScript's compiler
- what the browser uses to convert strings into actions
- most misused feature of the language
  - decreases script's performance substantially
  - also poses a huge security risk because it grants far too much power to the passed in text
- Avoid it if you can!

Use Native JSON Methods

Write Fast JavaScript

Native JSON Methods

```
var jsObjStringParsed = JSON.parse(jsObjString);
var jsObjStringBack = JSON.stringify(jsObjStringParsed);
```

JSON Always Faster than XML for Data

XML Representation

```
<!DOCTYPE glossary PUBLIC "DocBook V3.1">
   <glossary><title>example glossary</title>
      <GlossDiv><title>S</title>
         <GlossList>
          <GlossEntry ID="SGML" SortAs="SGML">
             <GlossTerm>Markup Language</GlossTerm>
             <Acronym>SGML</Acronym>
             <Abbrev>ISO 8879:1986</Abbrev>
             <GlossDef>
                <para>meta-markup language</para>
                <GlossSeeAlso OtherTerm="GML">
                <GlossSeeAlso OtherTerm="XML">
             </GlossDef>
          <GlossSee OtherTerm="markup">
       </GlossEntry>
    </GlossList>
   </GlossDiv>
</glossary>
```

JSON Representation

```
"glossary":{
   "title": "example glossary", "GlossDiv":{
      "title": "S", "GlossList": {
         "GlossEntry": {
            "ID": "SGML",
            "SortAs": "SGML",
            "GlossTerm": "Markup Language",
            "Acronym": "SGML",
            "Abbrev": "ISO 8879:1986",
            "GlossDef": {
               "para": "meta-markup language",
               "GlossSeeAlso": ["GML", "XML"] },
            "GlossSee": "markup" }
         }
      }
   }
```

Declare Variables Outside of the For Statement

Bad

```
for(var i = 0; i < someArray.length; i++) {
    var container = document.getElementById('container');
    container.innerHtml += 'my number: ' + i;
    console.log(i);
}
```

Good

```
var container =
document.getElementById('container');
for(var i = 0, len = someArray.length; i < len; i++) {
    container.innerHtml += 'my number: ' + i;
    console.log(i);
}
```

Avoid automatic conversions of DOM values

Values from DOM are strings by default

```
this.boardSize = document.getElementById("benchmarkBox").value;

for (var i = 0; i < this.boardSize; i++) {          //this.boardSize is “25”
  for (var j = 0; j < this.boardSize; j++) {        //this.boardSize is “25”
      ...
  }
}
```

SLOW

```
this.boardSize = parseInt(document.getElementById("benchmarkBox").value);

for (var i = 0; i < this.boardSize; i++) {          //this.boardSize is 25
  for (var j = 0; j < this.boardSize; j++) {        //this.boardSize is 25
      ...
  }
}
```

FAST

(25% marshalling cost reduction in init function)

Don't Pass a String to "SetInterval" or "SetTimeOut"

- Never pass a string to SetInterval and SetTimeOut

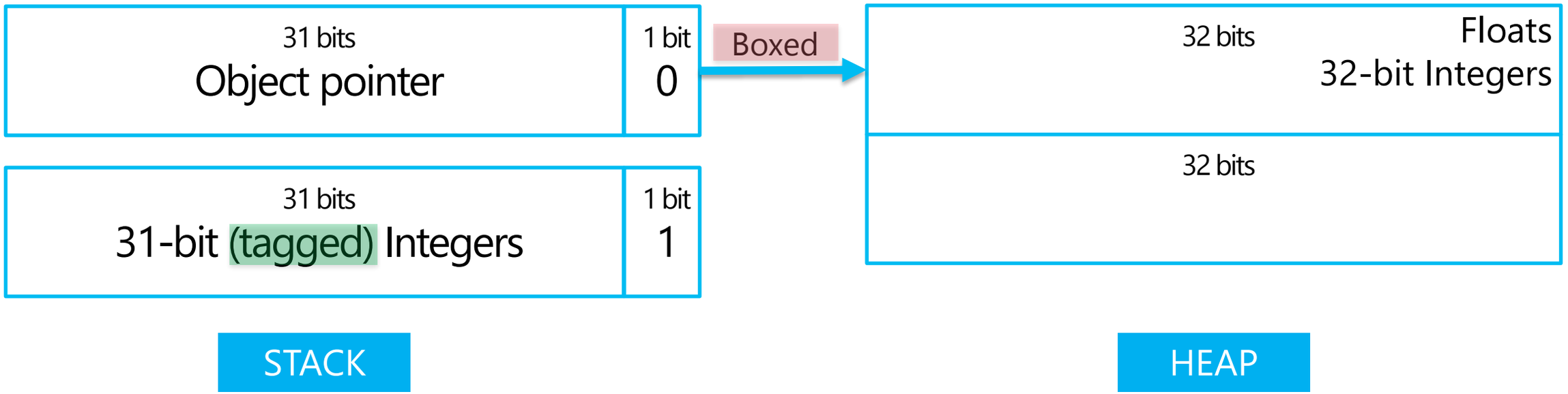
```
setInterval(
    "document.getElementById('container').innerHTML += 'My new
    number: ' + i", 3000
);
```

- Pass function name

```
setInterval(someFunction, 3000);
```

Numbers in JavaScript

- All numbers are IEEE 64-bit floating point numbers
  - Great for flexibility
  - Performance and optimization challenge



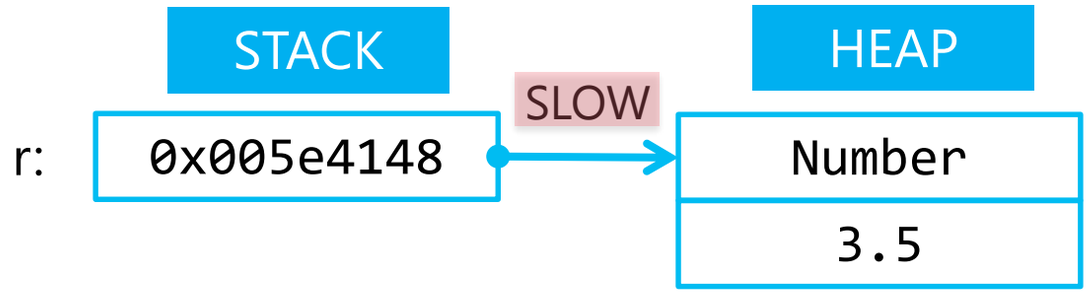
Avoid creating floats if they are not needed

Fastest way to indicate integer math is |0

```
var r = 0;

function doMath(){
  var a = 5;
  var b = 2;
  r = ((a + b) / 2);                    // r = 3.5
}
...
var intR = Math.floor(r);
```

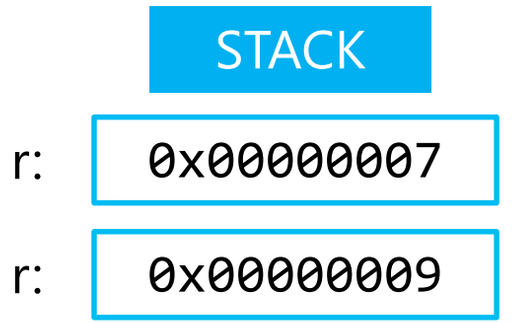
SLOW




```
var r = 0;

function doMath(){
  var a = 5;
  var b = 2;
  r = ((a + b) / 2) |0 ;                // r = 3
  r = Math.round((a + b) / 2);          // r = 4
}
```

FAST



Take advantage of type-specialization for arithmetic

Create separate functions for ints and floats; use consistent argument types

```
function Distance(p1, p2) {
  var dx = p1.x - p2.x;
  var dy = p1.y - p2.y;
  var d2 = dx * dx + dy * dy;
  return Math.sqrt(d2);
}



var point1 = {x:10, y:10};
var point2 = {x:20, y:20};
var point3 = {x:1.5, y:1.5};
var point4 = {x:0x0AB, y:0xBC};



Distance(point1, point3);
Distance(point2, point4);
```

SLOW

```
function DistanceFloat(p1, p2) {
  var dx = p1.x - p2.x;
  var dy = p1.y - p2.y;
  var d2 = dx * dx + dy * dy;
  return Math.sqrt(d2);
}
function DistanceInt(p1,p2) {
  var dx = p1.x - p2.x;
  var dy = p1.y - p2.y;
  var d2 = dx * dx + dy * dy;
  return (Math.sqrt(d2) | 0);
}
var point1 = {x:10, y:10};
var point2 = {x:20, y:20};
var point3 = {x:1.5, y:1.5};
var point4 = {x:0x0AB, y:0xBC};

DistanceInt(point1, point2);
DistanceFloat(point3, point4);
```

FAST

User .innerHTML to Construct Your Page

Use DOM Efficiently

```
function InsertUsername()
{
      document.getElementById('user').innerHTML = userName;
}
```

More JavaScript Books

- JavaScript the Good Parts Douglas Crockford, May 2008
  - http://www.amazon.com/JavaScript-Good-Parts-Douglas-Crockford/dp/0596517742/ref=sr_1_1?s=books&ie=UTF8&qid=1362696373&sr=1-1&keywords=javascript+good+parts
- JavaScript Patterns Stoyan Stefanov, September 2010
  - http://www.amazon.com/JavaScript-Patterns-Stoyan-Stefanov/dp/0596806752/ref=sr_1_1?s=books&ie=UTF8&qid=1362696287&sr=1-1&keywords=stoyan+stefanov
- JavaScript Cookbook Shelley Powers, July 2010
  - http://shop.oreilly.com/product/9780596806149.do

Code Quality

- Code Conventions for the JavaScript Programming Language
  - http://javascript.crockford.com/code.html
- Use JSLint.com. Pass with no warnings
- JSLint can help improve the robustness and portability of your programs
  - http://www.jslint.com/
  - http://www.javascriptlint.com/download.htm