JavaScript and its suitability as an application development language

Johan Wikström (jwikstrom) - 645714
Anaël Bonneton (abonneton) - 646275

October 22, 2013

Contents

1 Introduction

`JavaScript` was created by Brendan Eich in 1995 for the company Netscape Communications Corporation. Nowdays, it is the most used programming language for client-side programming. It has also become common in sever-side programming. However, `JavaScript` is still quite unpopular among professional developers. The main reason is that it was initially created to make programming easier than with traditional programming languanges such as `Java`. `JavaScript` was created for the 'dummies'.

We will focus on the following question *"Is JavaScript suitable for application programming?"* with different points of view : the application design, the type system, and finally the security.

2 Application design

The `JavaScript` syntax is easy, without a lot of contraints like in `Java`. For example, semi-colons are not compulsary at the end of each statement.

2.1 Scope of variables

`Javascript` allows to the programmer to define *local* and *global* variables (listing 1).

```
1     var i = 10; // a local variable
2     j = 20;     // a global variable
```

Listing 1: Defining variables

In JavaScript local variables are declared with the keyword var. If the programmer forgets to add this keyword before the declaration of a variable it is a global variable. Global variables are accessible from anywhere, even in different files loaded on the same page. In the listing 2, even if the variable j is declared inside the function f, this variable is global : it will be accessible everywhere.

```
1 function f(x, y) {
2     var i = 10; // a local variable
3     j = 20;     // a global variable
4 }
```

Listing 2: Global and local variables

With JavaScript it is very easy to use a lot of global variables, and to create them by accident (forgetting the keyword var). However, this is not a good programming practice. The scope of the global variables is not limited, they can be read and modified everywhere : it may be a source of unreadability. Code is easier to understand and to maintain when the scope of its variables is limited. Besides, it is very easy to overwrite gobal variables unintentionally and create bugs. Finally, it is very hard to reuse your code in another project if it contains a lot of global variables : the risk of collision is too important.

With JavaScript there is no scope like in Java : even if a variable is defined inside a function, its scope is not always only the function. Programmers have to be very careful with the scope of variables in JavaScript.

2.2 Creating an object

JavaScript offers several ways to create an object with its fields and its methods. The listings 3, 4 and 5 show the different options.

```
1  function coordinates(x, y) {
2      // Definition of the fields
3      this.x = x;
4      this.y = y;
5      // Definition of the methods
6      this.setX = setX;
7      function setX(x) {
8          this.x = x;
9      }
10     this.setY = setY;
11     function setY(y) {
12         this.y = y;
```

```
13      }
14  }
15  var point = new coordinates(1, 2);
```

Listing 3: Creating an object with a constructor

```
1  // We suppose the functions setX and setY are defined before
2  var point = {x:1, y:2, setX:setX, setY:setY};
```

Listing 4: Creating an object with the "C-structure" syntax

```
1  var point = new Object();
2  point.x = 1;
3  point.y = 2;
4  // We suppose the functions setX and setY are defined before
5  point.setX = setX;
6  point.setY = setY;
```

Listing 5: Creating an object with the operator "new Object()"

Whatever the method used to create the object, it is always possible to add another field or another method dynamically (listing 6).

```
1  point.z = 3;
2  function setZ(z) {
3          this.z = z;
4      }
5  point.setZ = setZ;
```

Listing 6: Adding a new field or a new method dynamically"

With the "C-structure" syntax and the operator `new Object()` it is very easy to develop a `JavaScript` application without thinking about the design of the application. Non-experienced developers may write code very quickly without thinking about readability for future developers and maintainability. However, the `JavaScript` constructors can be used as the classes in `Java` to define all the fields and the methods of an object.

3 Type System

JavaScript has a loose dynamic type system. This means that when you create a variable, you do not need to specify a type and the type of the variable can change during execution. Internally there is a type however and there are six internal types:

1. Undefined
2. Null
3. Boolean

4. String
5. Number
6. Object

```
int x = 1;
x = "hi";   // invalid, causes compiler error
```

Listing 7: Changing type of variable in Java

```
var x = 1;   // now x has Number type
x = "hi";    // now x has String type
x = {};      // now x has Object type
```

Listing 8: Changing type of variable in JavaScript

If a variables type does not fit the operation that is applied on it, JavaScript will attempt to *coerce* the variable into that type. This can be a source of great confusion and many bugs since it effectively masks errors. But it can make programming easier for beginners since they do not have to worry about bitness, converting input strings into numbers and some sorts of zero inputs.

```
2 + '10'   == '210' // Number coerced into String
2 * '10'   == 20    // String coerced into Number
true - 10  == -9    // Boolean coerced into Number
```

Listing 9: Automatic type coercion

In general, JavaScript tries to recover from errors made by the programmer, by avoiding throwing exceptions when encountering type errors. For example, when an uninitialized variable is used, there is no error but the variable is assigned the type Undefined and value undefined. This usually causes errors later in the program when the value is used. The errors are amplified by the fact that several normal operations are defined for the undefined value which further delays detection of the error.

```
1+undefined        == NaN
''+undefined       == 'undefined'
undefined && true  == undefined
undefined || true  == true
```

Listing 10: Normal operations on undefined value

This loose typing may be a disadvantage of the language when comparing the language to strongly typed languages such as Java. In Java the type system allows for static analysis of the code and many bugs are found at compile time. The true advantage of the loose type system only becomes apparent when the programmer uses the ability to augment types with new methods and properties at runtime.

3.1 JavaScript Object and prototyping

In Java, classes are static. There is no way to add a new method or property to an object after it is created. In JavaScript, there are no classes so all methods and properties are dynamically. This is a very powerful feature that may also be the cause of many bugs since properties and methods can be added and removed at any time during the programs execution.

```
1 a = {}  // empty object
2 a.property = "a"
3 a.method = function(){print("hello")}
```

Listing 11: JavaScript

```
Java
class A {
public String property;
public void method(){
print("hello");
}
}
A a = new A();
a.property = "a";
a.property2 = "b"; // Invalid, was not specified in class
```

Another difference between Java and JavaScript is the way inheritance is implemented. In Java, a class can statically inherit methods and variables from a parent class. In JavaScript, there are no classes, only dynamic objects, so JavaScript uses another form of inheritance. All objects in a JavaScript execution are linked to a prototype object. The prototype describes the methods and properties assigned to the object and this is used by the dynamic type system to determine whether One feature - inheritance - loose typing - dynamic

4 Security

4.1 Cross site scripting

Cross site scripting is one of the most commond security breaches today. It is a generic term describing the inclusion of malicious code into a webpage. Cross site scripting is not limited to JavaScript but the language has several features that facilitate cross site scripting.

One way to include malicious code into a web page is to trick the server into inserting the correct HTML tags into a dynamic web page. This is facilitated by the fact that JavaScript can be inserted and run anywhere on a webpage by using the ¡script¿ tag. One example of this can be shown in listing 12 and the

problem is amplified by browser makers that are trying to make sense out of broken script tags and run them. So even if the web server has some sort of protection of script inclusion, it is often insufficient.

```
<!-- the user has previously submitted his name to the application
    -->
<html>
<body>
<!-- Here Mr Anderson is fetched from a database-->
<p>Hello Mr Anderson</p>
</body>
</html>
<!--malicious case, the user has entered code instead of a name -->
<html>
<body>
<p>Hello <script> evilCode();</script></p>
</body>
</html>
```

Listing 12: Insertion of malicious script on the server side

Another form of cross site scripting is the result of one unfortunate language feature. JavaScript has the ability to evaluate and run JavaScript using the eval() function. As demonstrated in listing 13, the eval function takes a string and runs the JavaScript code contained in it. In combination with the many global variables in JavaScript, this is disastrous from a security perspective

```
eval('var a = 5');              // creates variable a
var userInput = getUserInput();
eval(userInput);                // evaluates possibly harmful input
```

Listing 13: Insertion of malicious script on the client side, eval().

5 Conclusion