Una veloce introduzione alla programmazione procedurale in JavaScript

Matteo Pradella
matteo.pradella@polimi.it
Settembre 2013

Il linguaggio JavaScript

- Introdotto da Brendan Eich (Netscape) nel 1995 come linguaggio di scripting per WWW
- supportato da tutti i browser
- *Sintassi stile C*: molti costrutti sono identici (if, while, switch, …)

Il linguaggio JavaScript (2)

- *Tipizzazione dinamica*: non si definiscono i tipi delle variabili; in controllo avviene a *runtime*
- *Gestione automatica della memoria*: non è necessario allocare/deallocare esplicitamente i dati (*Garbage Collector*)
- Si può usare direttamente a linea di comando in quasi tutti i browser. Per es. in Chrome: *Tools* > *Developer Tools*

Variabili: definizione ed assegnamento

```
var x;    // x assume il valore undefined
var y = “ciao”;    // stringa
var z = 23;
```

Le variabili sono *riferimenti* (puntatori) al dato, non “contenitori” come in C.

L’assegnamento è sintatticamente come in C:

```
x = 12;
```

Se serve, c’è `null`

Operazioni sui tipi predefiniti

Ci sono le solite ++, --, +=, …

Nel caso delle stringhe, + concatena:

```
> “ciao” + “ ” + “mondo”
“ciao mondo”
```

Accesso con le solite quadre:

```
> “ciao”[2]
“a”
```

Array

- Per esempio sia `var A = [1,2,3];`
- Il solito accesso: `A[2]` è 3, si parte da 0
- Lunghezza: `A.length`
- Iterazione: c'è una comoda variante di *for*:

  `for(var i in A) A[i] += 1;`
- Accesso a sotto-array con *slice*:

```
> A.slice(1,3) // secondo estremo escluso
[2,3]
```

Funzioni e passaggio dei parametri

- Esempio di definizione di funzione:

```
function f(x) { return x*x; }
```

- Il passaggio di una variabile a una funzione avviene *copiando* il riferimento, in una copia *locale* alla funzione

Passaggio dei parametri: esempi

```
> function f(x) { x++; return x; }
> var x = 12; // globale
> f(x);
13
> x // è stata cambiata solo la x locale a f
12
> function g(a) {a[0] = 0;}
> var A = [1,2,3];
> g(A) // in questo caso A cambia
```

Un esempio classico

```
function insertion_sort(A) {
    for(var j = 1; j < A.length; j++) {
        var key = A[j];
        var i = j-1;
        while (i >= 0 && A[i] > key) {
            A[i+1] = A[i];
            i--;
        }
        A[i+1] = key;
    }
}
```

Confrontiamolo con lo pseudocodice

```
function insertion_sort(A) {
  for(var j=1; j < A.length; j++) {
      var key = A[j];
      var i = j-1;
      while (i >= 0 && A[i] > key) {
          A[i+1] = A[i];
          i--;
      }
      A[i+1] = key;
  }
}
```

```
INSERTION-SORT(A)
1 for j := 2 to A.length
2   key := A[j]
3   i := j−1
4   while i > 0 and A[i] > key
5     A[i+1] := A[i]
6     i := i − 1

7   A[i+1] := key
```

Definizione di nuovi tipi di dati

In JS non c'è *struct*, si usa una *funzione costruttore*:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}
```

Definizione di nuovi tipi di dati (2)

Per creare una nuova persona (analogo di *malloc*):

```
var p = new Person("Giovanni", 21);
```

Esempio:

```
> p
Person {name: "Giovanni", age: 21}
> p instanceof Person // controllo il tipo
true
```