Node Callback Passing Style

NODE CALLBACK PASSING STYLE

Per la gestione di task asincrone in JavaScript storicamente viene utilizzata la prassi di passare funzioni, sotto forma di parametri, che vengono eseguite dopo il completamento del task (callback).

```
function asyncFunction(a, b, callback) {
  // Fa cose
  callback(data);
}
```

NODE CALLBACK PASSING STYLE

Nel mondo Node.js c'è la prassi di utilizzare due argomenti per le callback. Il primo rappresenta l'eventuale errore (una forma di `throw` asincrono), il secondo l'eventuale risultato (un `return` asincrono).

```
fs.readFile('./my-data.txt', function (err, data) {
  if (err) {
    // Abbiamo un errore. Questa sezione è come un "catch"
  } else {
    // Tutto OK. Possiamo usare data
  }
});
```

NODE CALLBACK PASSING STYLE

In alcuni contesti invece si usano due diverse callback, una per il caso "positivo" una per il caso "negativo":

```
function asyncFunction(a, b, onResult, onError) {
  // Fa cose
  if (error) {
    onError(error);
  } else {
    onResult(data);
  }
}
```

> NOTE
>
> Nomi alternativi per il caso positivo: `onResult`, `onSuccess`, anche solo `callback`
>
> Nomi alternativi per il caso negativo: `onError`, `onFailure`.

NODE CALLBACK PASSING STYLE

Un esempio di funzione asincrona è, nel browser, `setTimeout`:

```
setTimeout(
  // espressione che ritorna la funzione da eseguire "in ritardo"
  sayHello,
  // tempo del ritardo (delay) in millisecondi
  1000
);

function sayHello() {
  alert('Ciao!');
}

alert('Che bella giornata!');
```

> NOTA
>
> Anche se l'alert per "Che bella giornata" compare più tardi nel codice, anche dopo aver chiamato `setTimeout`, verrà mostrato per primo.
>
> L'esecuzione della funzione `sayHello` verrà rimandata nel tempo, mentre l'ultimo alert dello snippet qui sopra fa parte del percorso "sincrono" del codice.