Typescript: dichiarare variabili, parametri, classi e interfacce

#### di [Andrea Boschin](http://mvp.microsoft.com/en-us/mvp/Andrea%20Boschin-4000289) – Microsoft MVP

* 1. 

*Maggio, 2013*

Nel [precedente articolo](http://msdn.microsoft.com/it-it/library/dn133041.aspx) abbiamo visto rapidamante l'espressività che ci è consentita da Typescript nella dichiarazione di nostri tipi ed interfacce. Abbiamo anche evidenziato che il compilatore applica lo structured typing consentendoci di lavorare minimizzando le dichiarazioni, proprio perchè esso è in grado di identificare ricorrenze nella struttura dei tipi e verificare che siano soddisfatte. E' chiaro tuttavia che conoscere bene la semantica del linguaggio ci permette di sfruttarene al meglio l'espressività e adottare una migliore tecnica object oriented. L'obbiettivo di questo articolo è di approfondire l'utilizzo dei tipi, a partire dai semplici parametri fino alla definizione di classi e interfacce e nella loro ereditarietà.

### I tipi e il loro utilizzo

Il caso più semplice e ricorrente nelle dichiarazioni è rappresentato da variabili e parametri di un metodo e da qui partiremo analizzando le peculiarità del linguaggi. Typescript ammette pochi e semplici tipi di base:

**number**: numero reale. Esso è trattato sempre come se si trattasse di un "double"

**string**: stringa alfanumerica

**bool**: valore booleano (true o false)

**any**: è l'equivalente di una dichiarazione di tipo "variant" in quanto annulla il controllo del compilatore sui membri. Su un membro "any" può essere invocato qualunque metodo ma è onere del programmatore assicurarsi che esso esista.

**function**: così come avviene per javascript, il compilatore ammette anche la dichiarazione di metodi che ammettano una funzione, consentendo ad esempio l'uso di callback.

Tralasciano di tipi ovvi quali number, string e bool che dovrebbero essere di uso comune a chiunque e che obbiettivamente non presentano sorprese, vale la pena approfondire l'utilizzo di any e delle dichiarazioni di funzione:

* 1. TypeScript
  2. 1: var theWindow : any = <any>window;
  3. 2: theWindow.execute();

La precedente dichiarazione rende la variabile myObject generica a tal punto che è possibile chiamare su di essa un metodo di cui typescript non conosce l'esistenza. Il caso illustrato è abbastanza comune quando in una pagina convive il codice javascript di una libreria di terze parti e il nostro typescript. Se una ipotetica libreria avesse dichiarato una funzione nel documento, Typescript non potrebbe conoscerla e di conseguenza il compilatore si interromperà con un errore. Ecco quindi che si effettua il cast di "window" ad "any" (il cast si indica con le parentesi angolari <any>) e a questo punto è possibile chiamare la funzione senza che il compilatore si impunti.

Un altro caso particolare sono le variabili che ammettono come valore una funzione:

* 1. TypeScript
  2. 1: var callback : (number) => string;
  3. 2:
  4. 3: callback = function(n)
  5. 4: {
  6. 5: return "The answer is " + n;
  7. 6: };
  8. 7:
  9. 8: var result = callback(42);

La variabile callback in questione viene dichiarata con una sintassi che mima quella delle lambda expression. La sintassi "(int) => string" identifica una funzione che ha un parametro intero e un valore di ritorno di tipo stringa.

Inutile dire che è ammissibile avere funzioni senza parametri

var callback : () => string

con un valore di ritorno void

var callback : (mynum: number) => void

oppure con parametri multipli

var callback : (mynum: number, mystring: string, mybool: bool) => string

Interessante comunque notare che il compilatore non richiede che la funzione passata abbia esplicitato il tipo dei parametri ma, per mezzo dell'inferenza, applica automaticamente i tipi attesi. E' comunque sempre ammesso esplicitarli. Per passare una funzione alla variabile callback è possibile operare come nell'esempio, oppure utilizzare la sintassi lambda-expression come segue:

* 1. TypeScript
  2. 1: var callback : (number) => string;
  3. 2:
  4. 3: callback = (n) => { return "The answer is " + n };
  5. 4:
  6. 5: var result = callback(42);

Fin'ora per semplicità abbiamo visto le dichiarazioni applicate alle variabile, ma i medesimi tipi possono essere utilizzati anche per dichiare i parametri di un metodo piuttosto che le proprietà di una interfaccia o di una classe. Vediamo alcuni esempi:

* 1. TypeScript
  2. 1: // metodo
  3. 2:
  4. 3: function getRemoteData(callback: (result) => void) : void
  5. 4: {
  6. 5: // download here
  7. 6: }
  8. 7:
  9. 8: // interfaccia
  10. 9:
  11. 10: interface IHandler
  12. 11: {
  13. 12: value: number;
  14. 13: callback : (result) => void;
  15. 14: }
  16. 15:
  17. 16: // costruttore di classe
  18. 17:
  19. 18: class MyHandler implements IHandler
  20. 19: {
  21. 20: value: number;
  22. 21:
  23. 22: constructor(public callback: (result) => void)
  24. 23: {}
  25. 24: }

In ordine di apparizione, nel primo esempio viene utilizzato un parametro denominato "callback" che è richiesto sia una funzione che accetta un parametro "result" ("any" visto che è omessa la dichiarazione di tipo) e non ritorna alcun valore. Nel secondo esempio invece abbiamo la definizione di una interfaccia che accetta una proprietà numerica e una seconda proprietà "callback" che richiede una function. Infine nel terzo esempio di da implementazione alla precedente interfaccia con una classe. In questo caso il metodo callback verrà fornito nel costruttore e la keyword public sul parametro da automaticamente implementazione alla proprietà pubblica (e questo basta per soddisfare l'interfaccia).

### Parametri opzionali

Un problema ricorrente nella definizione di metodi sono i parametri opzionali. E' tipico, soprattuto di Javascript, l'avere funzioni i cui parametri non forniti sono automaticamente valorizzati a "null" e quindi con un semplice controllo è possibile simulare l'overloading che esiste nei linguaggi di più alto livello. In Typescript l'overloading dei metodi non esiste, se non nelle dichiarazioni delle interfacce, ma vige invece la possibilità di definire opzionali alcuni o tutti i parametri. Partendo da questa ultima abbiamo:

* 1. TypeScript
  2. 1: function addItem(name: string, value?: number): void
  3. 2: {
  4. 3: if (value == null)
  5. 4: value = 0;
  6. 5: }

L'indicazione di un "?" dopo il nome del parametro lo rende opzionale. E' chiaro che a questo punto abbiamo l'onere di verificare che esso sia valorizzato per evitare errori. Nell'esempio il parametro "value" viene valorizzato con un valore di default nel caso non sia stato specificato.

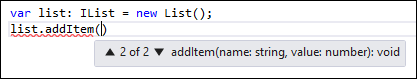
La cosa interessante è che possiamo fare uso delle interfacce per simulare l'overload del metodo, fornendo una descrizione nell'intellisense che è di grande aiuto per lo sviluppatore. Prima di tutto dichiariamo una classe con il precedente metodo:

* 1. TypeScript
  2. 1: class List
  3. 2: implements IList
  4. 3: {
  5. 4: addItem(name: string, value?: number): void
  6. 5: {
  7. 6: if (value == null)
  8. 7: value = 0;
  9. 8: }
  10. 9: }

A questo punto, nell'interfaccia IList possiamo dare la definizione di due metodi con il medesimo nome. Attenzione che questa operazione nella classe darà adito ad un errore di compilazione mentre è perfettamente ammessa nell'interfaccia:

* 1. TypeScript
  2. 1: interface IList
  3. 2: {
  4. 3: addItem(name: string): void;
  5. 4: addItem(name: string, value: number): void;
  6. 5: }

In seguito a questa dichiarazione potremmo sfruttare l'interfaccia e l'intellisense di Visual Studio 2012 ci fornirà una descrizione del tutto simile a quella cui siamo abituati con C#:



### Conclusioni

Il ridotto numero di tipi disponibile in Typescript non deve trarre in inganno. Così come in Javascript siamo in grado di trattare qualunque tipo abbastanza semplicemente, tanto più il Typescript saremo in grado di gestire agilmente qualunque informazione, con il supporto di una libreria di classi che sopperisce alla ristrettezza di tipi primitivi. E’ il caso ad esempio del tipo Date che, come accade in Javascript, non è nativo del linguaggio. Nella libreria automaticamente linkata dal compilatore troviamo le definizioni del tipo in termini di interfaccia e possiamo gestire correttamente il tipo in questione.

#### di [Andrea Boschin](http://mvp.microsoft.com/en-us/mvp/Andrea%20Boschin-4000289) - Microsoft MVP

*twitter*: @aboschin

*blog italiano*: <http://blog.boschin.it>

*blog inglese*: <http://xamlplayground.org>

*facebook***:** <http://www.facebook.com/thelittlegrove>

*profilo***:** <http://slpg.org/AndreaBoschin>

Articolo pubblicato anche [sul Blog italiano](http://www.boschin.it/post/2013/05/17/Typescript-dichiarare-variabili-parametri-classi-e-interfacce.aspx)