Tecnologie e applicazioni web

TypeScript

Filippo Bergamasco (filippo.bergamasco@unive.it)
http://www.dais.unive.it/~bergamasco/
DAIS - Università Ca'Foscari di Venezia
Anno accademico: 2017/2018

What if we could strengthen JavaScript with the things that are missing for large scale application development, like static typing, classes [and] modules . . . that's what TypeScript is about.

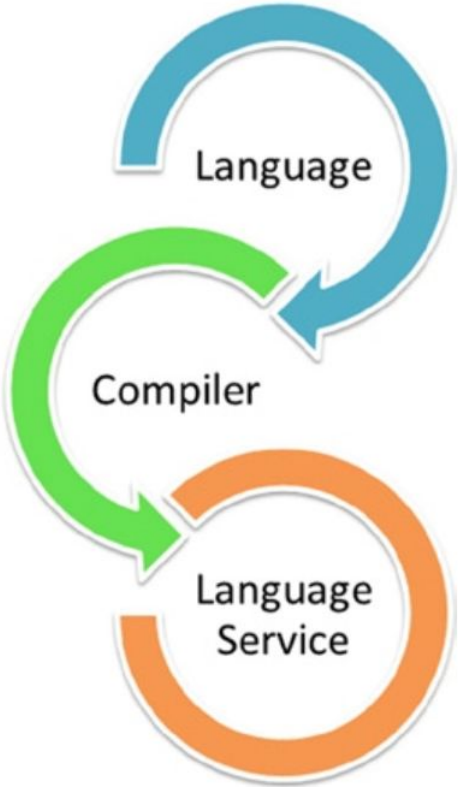
—Anders Hejlsberg

TypeScript

E’ un linguaggio creato da Microsoft, rilasciato nel 2004 con licenza open-source Apache 2.0.

E’ un superset tipato del linguaggio Javascript: contiene tutte le caratteristiche (sintattiche e funzionali) di Javascript e ne aggiunge di nuove

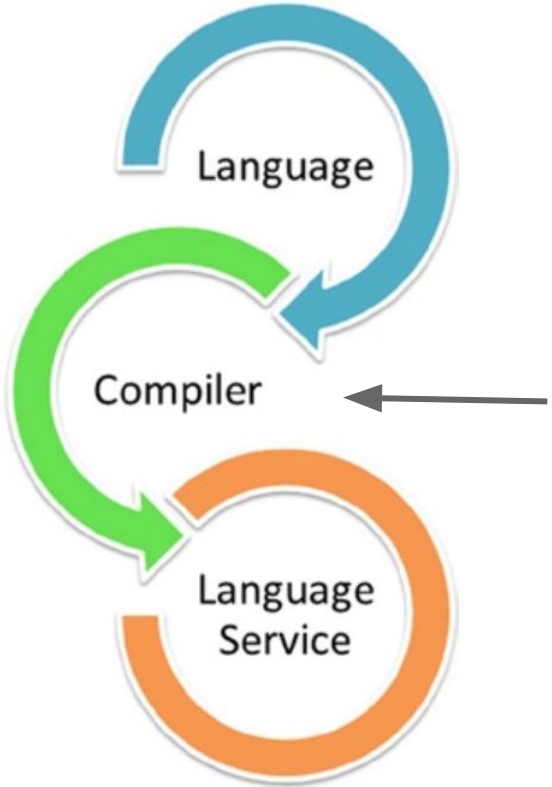
TS è composto da 3 parti



Linguaggio, superset di Javascript, che aggiunge una nuova sintassi, keywords e annotazioni.

E' importante capire le caratteristiche del linguaggio per trarre massimo profitto dal compilatore e dal language service

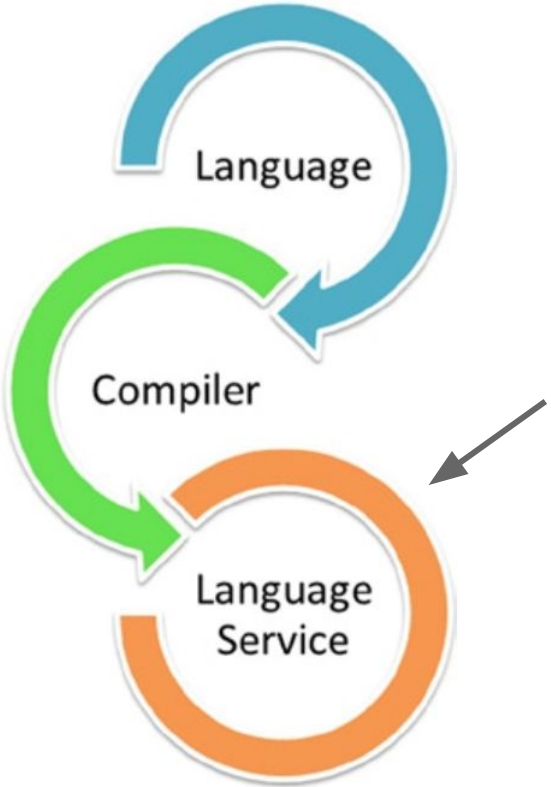
TS è composto da 3 parti



Il compilatore trasforma il codice TS in codice Javascript puro (ES5, ES6 o ES3 opzionalmente)

Può emettere errori o semplici warning e svolgere altri task come combinare più files sorgente, generare source maps, etc

TS è composto da 3 parti



Il language service offre informazioni sui tipi che possono essere utilizzati dagli ambienti di sviluppo per aiutare la programmazione: autocompletion, type hinting, refactoring, etc

Compilazione o transpilazione?

Il fatto che TS produca in output codice Javascript è interessante perché permette di utilizzare tutti i frameworks e le infrastrutture che già sono pensate per utilizzare quel linguaggio

Quando la compilazione produce codice in un linguaggio con un livello espressivo simile a quello di input è chiamata transpilazione

Installazione e utilizzo

Il compilatore è un tool scritto in Javascript, installabile attraverso il gestore di pacchetti npm:

```
$ npm install -g typescript
```

Quando si invoca su un file con estensione .ts produce un file Javascript (.js) e, opzionalmente, un file di mapping per permettere il debugging

```
$ tsc inputfile.ts
```

Codice Javascript in TS

Un codice sorgente in Javascript è automaticamente codice TypeScript valido.

Dato che il sistema di tipi TS è più severo di quello di Javascript, il compilatore può generare dei warning anche se il codice Javascript è sintatticamente valido.

NOTA: TS cerca quando possibile di generare codice Javascript anche se vengono generati degli errori (ad esempio sul type checking)

.

TS Features

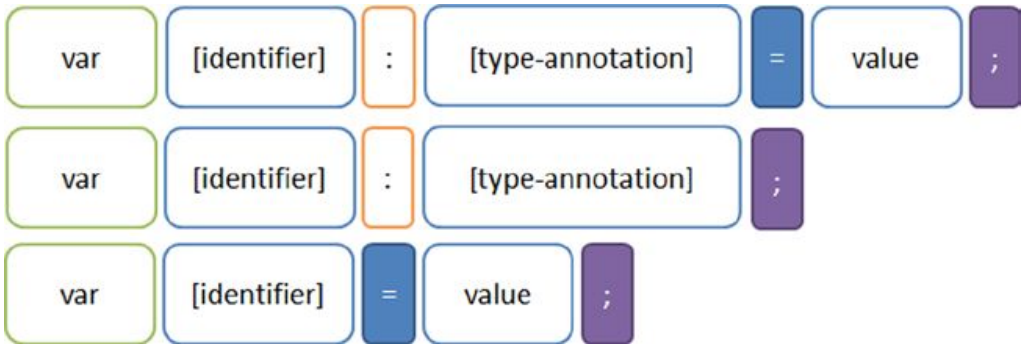
- Checking sui tipi
  - Tipi possono essere espressi staticamente
  - Type inference sugli assegnamenti
- Funzioni
  - Tipo dei parametri
  - Parametri opzionali
  - Parametri di default
  - Numero variabile di parametri
  - Overloading

TS Features

- Classi e interfacce
  - Definizione esplicita di classe ed interfacce
  - Costrutti per gestire l'ereditarietà tra classi
- Moduli
  - Concetto intrinseco di modulo per introdurre namespace
  - Moduli esterni caricabili con pattern CommonJS oppure AMD

Type annotations

Anche se TS può fare inferenza sui tipi, è molto utile poterli esprimere in modo esplicito attraverso le annotazioni:



Enumerazioni

TS supporta le enumerazioni nel type checker che permette di definire dei nuovi tipi che rappresentano liste di elementi corrispondenti a valori interi costanti

In TS le enumerazioni sono open-ended: tutte le dichiarazioni con lo stesso nome all'interno di un unico scope contribuiscono a formare uno stesso singolo tipo

.

Funzioni

Le funzioni possono essere annotate per descriverne in modo statico la firma.

Oltre al tipo, ciascun parametro può essere annotato come **opzionale** (è permesso non fornire alcun valore) o avere un valore di **default**.

- Ciascun parametro opzionale deve essere posizionato dopo quelli non opzionali
- I parametri di default possono contenere valori non costanti a compile time

.

Parametri variabili

E' possibile definire funzioni con un numero variabile di parametri (detti anche REST parameters) in aggiunta a quelli specificati nella firma.

- E' permesso un solo REST parameter
- Il REST parameter deve apparire alla fine della lista dei parametri
- Il tipo del REST parameter deve essere di tipo Array

Overload

TS supporta l'overload delle funzioni, implementato nel seguente modo:

- Si esplicitano tutte le possibili firme delle funzioni prima dell'implementazione
- Si implementa la funzione un'unica volta utilizzando **any** come tipo di dato

Interfacce

TS permette la definizione di Interfacce come i più comuni linguaggi object oriented.

Utilizzi:

- Per definire i metodi pubblici e le proprietà di una determinata classe
- Per definire la struttura di oggetti
- Per definire le operazioni disponibili in librerie di terze parti (per cui si vuole definire il sistema di tipi)

Interfacce

Una particolarità delle interfacce TS è che non producono alcun codice Javascript associato.

- Utilizzate dal type checker

In TS le interfacce possono estendere altre interfacce ma anche altre classi. In questo caso l'interfaccia eredita tutte le firme delle funzioni

.

Structural subtyping

I linguaggi tipati classici sono di tipo nominativo:

- L'equivalenza fra due tipi è determinata dal loro nome o da esplicite dichiarazioni

TS utilizza il cosiddetto structural subtyping

- Un tipo x è compatibile con un altro tipo y se y ha almeno tutti i membri di x

Questo permette di avere un type checking più flessibile e vicino allo "stile Javascript"

Type assertions

Vista la natura dinamica del linguaggio, vi possono essere casi in cui il type checker determina che un assegnamento non è corretto (e quindi restituisce errore) anche se semanticamente valido

E' possibile forzare il tipo di un determinato tipo con il costrutto `<tipo>`. Funziona in modo simile al concetto di cast ma avviene soltanto a livello del type checker senza alcun intervento a run-time

.

Classi

TS supporta il concetto di classi introducendo i seguenti costrutti:

- La keyword **class** per definire una classe
- La keyword **constructor** per specificare il costruttore della classe
- Le keyword **public**, **private** e **protected** per modificare la visibilità di metodi e proprietà
- Getters and setters

Generics

I generics sono uno strumento molto efficace per creare componenti software riutilizzabili. Permettono infatti di avere componenti in grado di lavorare su una diversa varietà di tipi anziché uno specifico

Sono più potenti di usare un tipo generico come **any** perché permettono comunque di riferirsi ad uno specifico tipo anche se non è noto in fase di definizione della funzione o classe

.

Moduli

TS fornisce il concetto di moduli all'interno del linguaggio stesso. Possono essere di due tipi:

- **Interni (namespace):** Si comportano come dichiarazioni di namespace, in cui è possibile decidere cosa esportare attraverso la keyword `export`
- **Esterni:** Ciascun file .ts rappresenta un modulo distinto che può essere importato in stile CommonJS o AMD

Approfondimenti

Documentazione ufficiale:
https://www.typescriptlang.org/docs/home.html

Definizioni di tipi per le librerie più comuni (ex. jQuery)
http://definitelytyped.org

Altri linguaggi

Coffeescript

Linguaggio ispirato a Ruby, Python e Haskell.

http://coffeescript.org/

Dart

Sviluppato da Google con lo scopo di sostituire Javascript come linguaggio unico per il web.

https://www.dartlang.org/guides/language/language-tour