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: 2021/2022

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.

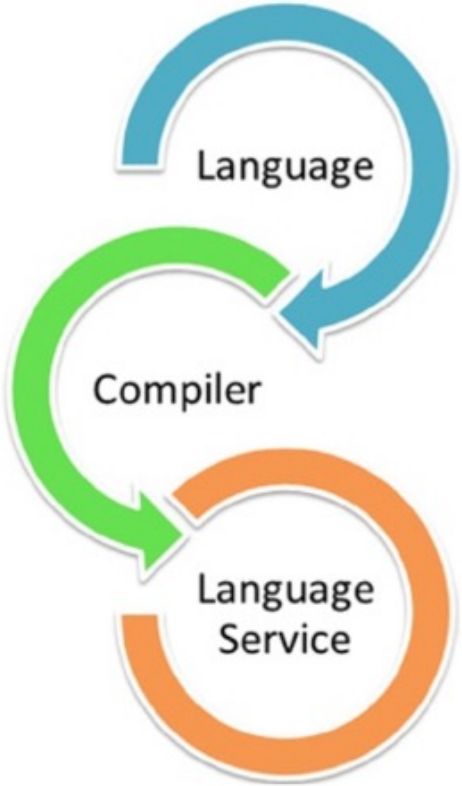
—Anders Hejlsberg

TypeScript

Language created by Microsoft, released in 2004 with the Apache 2.0 open-source license.

TypeScript is a typed superset of JavaScript: adds new features while keeping all the functional and syntactic characteristics of JavaScript

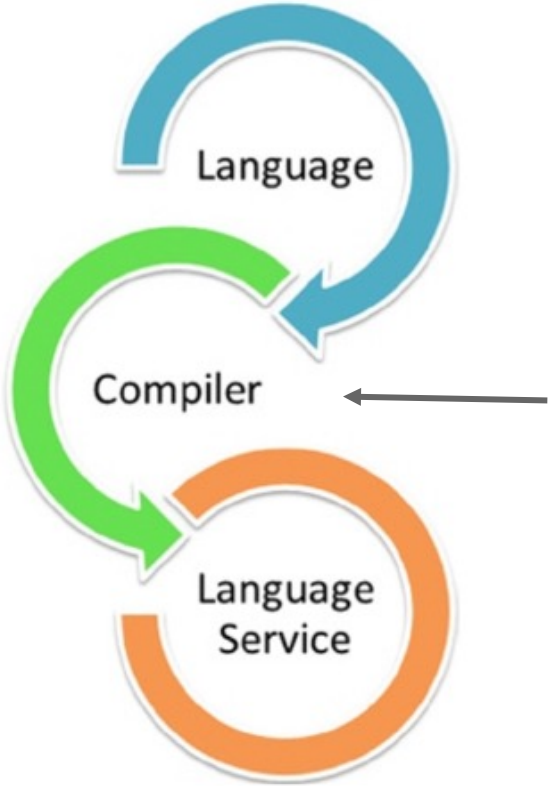
TS is composed by 3 parts



The language, adding keywords and new constructs to the JavaScript language.

It is important to understand its characteristics to take advantage of the compiler and language service.

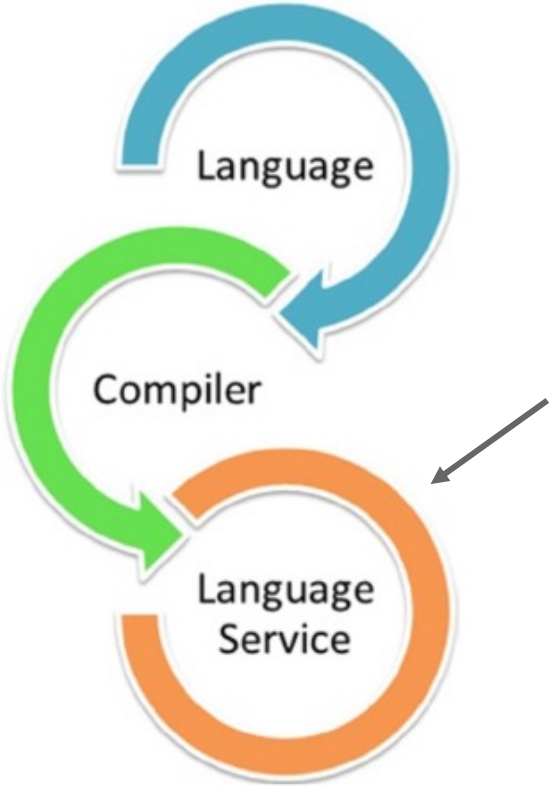
TS is composed by 3 parts



The compiler transforms TS code into pure JavaScript code (ES5, ES6 or ES3 optionally).

Can emit errors and warnings or perform simple tasks like combine multiple source files, generate source maps, etc.

TS is composed by 3 parts



The language service offers type information useful for IDEs to help with code completion, type hinting, refactoring, etc.

Compile or transpile?

When a compiler produces code to an high level language we call il **transpilation**.

The fact that TS actually produces JavaScript code is a great advantage since we can use all the frameworks already designed to use JavaScript.

Installation and usage

The compiler itself is a tool written in JavaScript. Can be installed through the npm package manager:

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

tsc can be invoked on a .ts file:

```
$ tsc inputfile.ts
```

To produce the compiled js file and a «map» file useful for debuggers

JS and TS

Any JavaScript code is also a valid TypeScript code.

Since TS type system is more strict with respect to JS, the TS compiler can generate some warnings even if the input file is a syntactically valid JS file.

NOTE: TS always tries to generate the JS file even if some errors are reported (for example due to the type checker)

.

TS Features

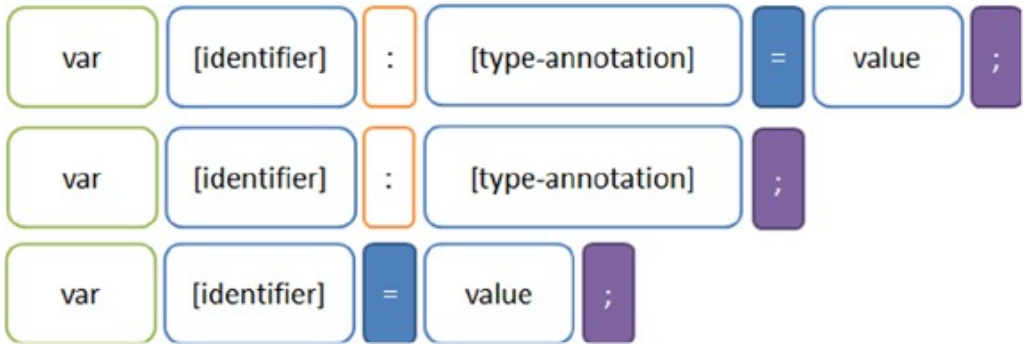
- Type checking
  - Types can be specified statically
  - Automatic type inference on variable assignments
- Functions
  - Typed parameters
  - Optional and default parameters
  - Overloading

TS Features

- Classes and Interfaces
  - Explicit class and interface definition
  - Class inheritance
- Modules
  - TS modules to define namespaces
  - External modules

Type annotations

TS can do automatic type inference. However, it is also useful to explicitly specify them via type annotations:



Base types

Nome	Valori
boolean	true / false
number	All floating point numbers
string	Strings
Array< base > o base[ ]	Every array in which elements are of a base type
any	Type representing any value (useful when the type is unknown during transpilation)
void	Type representing the absence of a value (like functions not returning any value)
Object	Everything that is not a primitive type in JavaScript

Enumerations

TS support enumerations: list of elements mapped by constant integer values.

Enumerations are open-ended: all the declarations with the same name inside a scope will concur to define the same type.

Functions

Functions can be annotated to statically describe their signature.

Together with the type definition, each parameter can be **optional** or a **default** value can be specified.

- Every optional parameter must be subsequent of the non-optional ones
- Default parameters can contain non-constant values (ie. Values unknown at compile time)

Variadic functions

A function can accept a variable number of parameters (called REST parameters) together with the ones specified in the signature.

- A single REST parameter is allowed
- The REST parameter must be the last one in the signature parameter's list
- The REST parameter must be an Array

Overload

TS support function overload, implemented as follows:

- All the possible signatures are specified before the implementation.
- Function is implemented only once using **any** as parameter type

Interfaces

Interfaces can be used in TS similar to other common object oriented languages.

Usages:

- To define public methods and attributes of a class
- To define the structure of an object
- To define the operations available on third party libraries (on which we want to implement a type system).

Interfaces

TS interfaces do not produce any associated JavaScript code!

They are meant to be used exclusively by the type checker.

Interfaces can extend other interfaces but also classes. In this case the interface will inherit all the function signatures.

Structural subtyping

Usually, in classic typed languages, the equivalence between two types is determined explicitly by their name or specific declarations.

TS uses the so called **structural subtyping:**

A type x is compatible with another type y if y has all the attributes of x

This allows a more flexible type checker closer to the JavaScript style.

Type assertions

A type definition can be forced using the construct `<type>`.

The idea is conceptually similar to type casting but is limited to the context of the type checker (no actual conversion happens at runtime)

Classes

TS support classes in the following way:

- The **class** keyword is used to define a class
- The **constructor** keyword is used to specify the class constructor
- **public**, **private** e **protected** are used to modify methods and attributes visibility
- Getters and setters are supported

Generics

Generics are a great tool to create reusable software components by let them work over a variety of types rather than a single one. This allows users to consume these components and use their own types

Better than simply using **any** because a generic type can be referenced even if unknown when a function or a class is defined (see the examples…)

More info

Official documentation:
https://www.typescriptlang.org/docs/home.html

Type definitions for popular libraries and frameworks:
http://definitelytyped.org