

Type Script

Created by :
Sangeeta Joshi

Agenda

- Why Typescript
- Typescript modules

TypeScript

Get TypeScript

- Node.js : The command-line TypeScript compiler can be installed as a Node.js package
- INSTALL : `npm install -g typescript`
- Check Version : `tsc -v`
- COMPILE : `tsc helloworld.ts`

Why Typescript

- developers who are familiar with Object Oriented Programming (OOP) feeling very comfortable with TypeScript's classes and interfaces.
- There is a lot of value here in getting running quickly and feeling secure that you have written solid code.

Why Typescript

- *TypeScript* Has Great Tools
- The biggest selling point of TypeScript is tooling. It provides advanced autocompletion, navigation, and refactoring. Having such tools is almost a requirement for large projects.
- Without them the fear changing the code puts the code base in a semi-read-only state, and makes large-scale refactorings very risky and costly.

Why Typescript

- This is because building rich dev tools has to be an explicit goal from day one, which it has been for the TypeScript team.
- That is why they built language services that can be used by editors to provide type checking and autocompletion. (Not only compiler)
- why there are so many editors with great TypeScript supports? the answer is the language services.
- intellisense and basic refactorings (e.g., rename a symbol)

Why Typescript

- *TypeScript is a Superset of JavaScript*

Since TypeScript is a superset of JavaScript, you don't need to go through a big rewrite to migrate to it. You can do it gradually, one module at a time.

TypeScript Intro

When Do I Need Classes?

Consider 3 main conditions where we consider a class, regardless of the language.

- Creating multiple new instances
- Using inheritance
- Singleton objects

Not necessary in Javascript but it's because they ***make it so much easier to do*** these things.

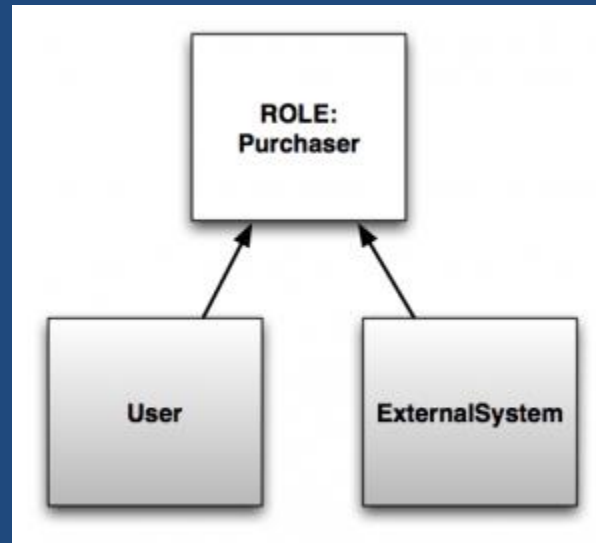
Interfaces

- TypeScript Makes Abstractions Explicit
- A good design is all about well-defined interfaces. And it is much easier to express the idea of an interface in a language that supports them.
- One of the coolest parts of TypeScript is how it allows you to define complex type definitions in the form of interfaces

Interfaces

- For instance, imagine a book-selling application where a purchase can be made by either a registered user through the UI or by an external system through some sort of an API.

Interfaces



As you can see, both classes play the role of a purchaser. Despite being extremely important for the application, the notion of a purchaser is not clearly expressed in the code. There is no file named `purchaser.js`. And as a result, it is possible for someone modifying the code to miss the fact that this role even exists.

Interfaces

- Defining an interface forces me to think about the API boundaries, helps me define the public interfaces of subsystems

Interfaces

TypeScript Makes Code Easier to Read and Understand

1. `jQuery.ajax(url, settings)`
 2. `ajax(url: string, settings?: JQueryAjaxSettings): JQueryXHR;`
- The first argument of this function is a string.
 - The settings argument is optional. We can see all the options that can be passed into the function, and not only their names, but also their types.
 - The function returns a JQueryXHR object, and we can see its properties and functions.

Typescript modules

- Starting with the ECMAScript 2015, JavaScript has a concept of modules. TypeScript shares this concept.
- Modules are executed within their own scope, not in the global scope;
- this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the export forms.
- Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the import forms.
- In TypeScript, just as in ECMAScript 2015, any file containing a top-level import or export is considered a module.