**Basic Typescript**

1. Khái niệm TypeScript

**TypeScript** được xem là một phiên bản nâng cao hơn của JavaScript vì nó được thiết kế thêm nhiều chức năng tiện lợi hơn, cải tiến hơn từ những điểm yếu của JavaScript như các lớp hướng đối tượng và Static Structural typing, bên cạnh đó TypeScript còn có thể hoạt động rộng rãi cho các ứng dụng của ngôn ngữ Angular2 và Nodejs.

Khác với sự đơn giản của JavaScript, du TypeScript cũng đồng thời kế thừa nhiều định nghĩa, khái niệm của đa dạng các ngôn ngữ C#, Java,… nhưng TypeScript lại có yêu cầu cao về trật tự rõ ràng.

## Chức năng của TypeScript

### Static Typing

Bạn có thể khai báo kiểu cho biến, và trình biên dịch sẽ giảm được tỷ lệ gán sai kiểu của các giá trị. Tuy nhiên nếu khai báo kiểu bị bỏ qua, chúng sẽ được tự động phát hiện từ code của bạn.

Các biến, tham số của hàm hoặc giá trị trả lại có thể có các kiểu được định nghĩa khi khởi tạo

Biên dịch tsc sẽ báo lỗi khi chúng ta thao tác không hợp lệ. Hệ thống cũng sẽ báo lỗi nếu thao tác truyền sai tham số tới một hàm.

Trong trường hợp TypeScript biên dịch thành JavaScript thành công, toàn bộ khai báo kiểu sẽ bị xóa

### Các kiểu dữ liệu được sử dụng phổ biến nhất:

* Any: Biến với kiểu này có thể có giá trị là một string, number hoặc bất kỳ kiểu nào.
* String: Giống chức năng của string trong JavaScript, có thể được bao quanh bởi ‘dấu nháy đơn’ hoặc “dấu nháy kép”.
* Number: Tất cả giá trị số trong hàm đều được biểu diễn bởi kiểu number, không có định nghĩa riêng cho số nguyên (integer), số thực (float) cũng như các kiểu khác.
* Boolean: true hoặc false, sử dụng 0 và 1 sẽ gây ra lỗi biên dịch.
* Arrays: Có 2 kiểu cú pháp: *my\_arr: number[];* hoặc *my\_arr: Array<number>*.
* Void: sử dụng khi **hàm** không trả lại bất kỳ giá trị nào
* Enum
* Null and undefined
* Object

### Interfaces

Chức năng chính của **Interfaces** là sử dụng để kiểm tra, xem một đối tượng có phù hợp với một cấu trúc nhất định hay không, trợ giúp trong giai đoạn giai đoạn phát triển.

Bằng cách định nghĩa một interface, ta có thể đặt tên trong trường hợp có sự kết hợp đặc biệt của các biến, đảm bảo rằng chúng luôn luôn đi cùng nhau.

Ngoài ra, thứ tự của các thuộc tính không quan trọng bằng việc chúng ta phải fill đủ số lượng các thuộc tính và đúng kiểu. Nếu một thuộc tính nào đó bị thiếu, hoặc sai kiểu, hoặc sai tên, trình biên dịch sẽ cảnh báo chúng ta.

### Classes

Trong các dự án, ứng dụng lớn, lập trình hướng đối tượng rất hay được các Dev sử dụng nhất là trong các ngôn ngữ như Java hoặc C#.

TypeScript cung cấp hệ thống class khá tương đồng với các ngôn ngữ này ví dụ như chức năng kế thừa, abstract classes, interface implementations, setter/getters, …

Từ phiên bản ECMAScript 2015 trở đi, classes được xem là một tính năng có sẵn trong JS và có thể không cần sử dụng TypeScript. Mặc dù 2 phiên bản này có nhiều chức năng tương tự nhau, nhưng chúng vẫn có điểm khác biệt, đó là TypeScript nghiêm ngặt hơn.

Sử dụng từ khóa *extends* để kế thừa các thuộc tính từ một class cha

### Modules

Tính module hóa rất quan trọng khi bạn đang thực hiện những dự án lớn. Nó phân chia code thành nhiều thành phần nhỏ và còn có khả năng tái sử dụng giúp dự án của bạn dễ tổ chức và dễ hiểu hơn so với file có hàng ngàn dòng code.

TypeScript có chức năng exporting và importing các module. Dùng [require.js](http://requirejs.org/) cho các ứng dụng client và [CommonJS](https://en.wikipedia.org/wiki/CommonJS) cho Node.js để cho phép modules TS dựa trên các thư viện của bên thứ ba.

### Generics

Chức năng của Generics là cho phép cùng một hàm có thể chấp nhận các tham số với nhiều kiểu khác nhau. Việc tạo ra các thành phần có thể tái sử dụng với generics tốt hơn sử dụng kiểu any, vì generics bảo tồn kiểu của các biến vào và ra của chúng.

## Cài đặt TypeScript

Cài đặt TypeScript package toàn cục:

npm install -g typescript

Kiểm tra phiên bản typescript:

tsc -v

TypeScript được viết trong các file **.ts** (hoặc **.tsx** cho **JSX**), nó không thể sử dụng trực tiếp trong trình duyệt và cần biên dịch thành JavaScript.

Câu lệnh sau sẽ nhận một file TypeScript là *main.ts* và chuyển nó thành JavaScripts *main.js*. Nếu *main.js* đã tồn tại nó sẽ bị ghi đè:

tsc main.ts

Biên dịch nhiều file:

tsc main.ts worker.ts

Hoặc tsc \*.ts

có thể sử dụng tùy chọn –watch để tự động biên dịch một file TypeScript khi có thay đổi:

tsc main.ts --watch

Cũng có thể sử dụng file *tsconfig.json* để có thể hỗ trợ quá trình tự động hóa.

vd:

{

"[compilerOptions](https://www.typescriptlang.org/tsconfig#compilerOptions)": {

"[module](https://www.typescriptlang.org/tsconfig#module)": "commonjs",

"[noImplicitAny](https://www.typescriptlang.org/tsconfig#noImplicitAny)": true,

"[removeComments](https://www.typescriptlang.org/tsconfig#removeComments)": true,

"[preserveConstEnums](https://www.typescriptlang.org/tsconfig#preserveConstEnums)": true,

"[sourceMap](https://www.typescriptlang.org/tsconfig#sourceMap)": true

},

"[files](https://www.typescriptlang.org/tsconfig#files)": [

"core.ts",

"sys.ts",

"types.ts",

"scanner.ts",

"parser.ts",

"utilities.ts",

"binder.ts",

"checker.ts",

"emitter.ts",

"program.ts",

"commandLineParser.ts",

"tsc.ts",

"diagnosticInformationMap.generated.ts"

]

}

## Ưu điểm của Typescript

### TypeScript thuận tiện và hoàn toàn miễn phí

### Có khả năng tái cấu trúc

### Giảm tỷ lệ mắc lỗi trong hệ thống

### Hạn chế thử nghiệm Boilerplate

### Quá trình hợp nhất mã đơn giản

### Hỗ trợ tối ưu hóa quy trình làm việc

1. Nhược điểm của TypeScript

### Bắt buộc dùng biên dịch

### Bước thiết lập cồng kềnh

### Chỉ là phần ngôn ngữ mở rộng hỗ trợs