# LOẠI CƠ BẢN

## Typescript là gì?

Là ngôn ngữ lập trình mở rộng của JavaScript, bổ xung các tính năng kiểu tĩnh và nhiều tính năng nâng cao giúp kiểm soát lỗi và phát triển ứng dụng dễ dàng hơn.

Lợi ích:

* Phát hiện lỗi ngay từ khi viết code thay vì đợi đến lúc chạy.
* Cải thiện đọc hiểu và bảo trì code.

## Kiểu chú thích

Chỉ định kiểu dữ liệu của biến, hàm, đối tượng.

Cú pháp:

: kiểu dữ liệu

VD1:

let counter: number;

counter = 1; // đúng

counter = "Hello"; // sai, biên dịch lỗi

VD2: Mảng.

let names: string[] = ['John', 'Jane', 'Peter', 'David', 'Mary'];

VD3: Đối tượng.

let person: {

name: string;

age: number;

};

person = {

name: 'John',

age: 25,

}; // valid

VD4: Hàm.

const sayHello = (name: string): string => {

return `Hello, ${name}!`

}

Có thể quy định 1 biến có 2 hoặc nhiều kiểu dữ liệu ngăn cách nhau bởi |

const userId = string | number.

## Kiểu suy luận

VD:

let counter = 0;

Tương đương với:

let counter: number = 0;

Đối với hàm nếu không có return sẽ mặc định quy định kiểu hàm là void, còn nếu có trả về thì nó sẽ dựa trên kiểu dữ liệu của return mà quy định kiểu trả về.

Đối với kiểu suy luận về cơ bản giống javascript, nó vẫn chạy được trong hầu hết các trường hợp, tuy nhiên đối với dự án lớn, đòi hỏi mức độ chính xác cao cần nên chỉ định kiểu.

## Kiểu số

Dùng kiểu : number cho tất cả các kiểu số, ngoài ra đối với số lớn hơn 253 – 1 thì dùng : bigint.

let price: number;

let big: bigint = 9007199254740991n;

## Kiểu chuỗi

Dùng kiểu : string, cho kiểu dữ liệu chuỗi.

let firstName: string = 'John';

let title: string = "Web Developer";

let profile: string = `I'm ${firstName}`;

## Kiểu đúng sai

Dùng kiểu : Boolean.

let completed: boolean = true;

## Kiểu đối tượng

Là kiểu đại diện cho tất cả giá trị không thuộc loại kiểu nguyên thủy.

Kiểu nguyên thủy: number, bigint, string, boolean, null, undefined, symbol

VD:

let employee: object;

employee = {

firstName: 'John',

lastName: 'Doe',

age: 25,

jobTitle: 'Web Developer'

};

## Kiểu mảng

Cú pháp:

let arrayName: type[];

VD:

let skills: string[] = [];

let scores : (string | number)[];

scores = ['Programming', 5, 'Software Design', 4];

## Kiểu Tuple

Hoạt động giống như mảng nhưng sẽ có thêm vài điều kiện.

VD:

let skill: [string, number];

skill = ['Programming', 5];

Nếu thay đổi thứ tự, chương trình sẽ lỗi.

Khai báo biến tùy chọn, đôi khi bạn không chắc chắn hoặc tham số đó có thể có hoặc không bạn có thể dùng ? để quy định nó:

let skill: [string, number?];

skill = ['Programming', 5]; // đúng

skill = ['Programming']; // đúng

// Tham số thứ 2 bắt buộc là kiểu number và có giá trị hoặc không đều không lỗi.

## Kiểu enum

Là đặt tên của một nhóm các giá trị hằng số (viết tắt của enumerated).

Cú pháp:

enum name {constant1, constant2, ...};

VD:

enum Month {

Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec,

}

console.log(Month.Jan);

// 0

enum Status {

Pending = 1, // Bắt đầu từ 1

InProgress = 3,

Completed = 5

}

console.log(Status.Pending); // 1

console.log(Status.Completed); // 5

Tuy nhiên enum thường không được sử dụng mấy trong dự án thực tế vì nó có khá nhiều nhược điểm:

* Một tập hợp nhỏ có các giá trị cố định.
* Giá trị này không được thay đổi

Nên sử dụng type thay vì enum.

## Kiểu dữ liệu any

Là kiểu dữ liệu bất kỳ, typescript sẽ bỏ qua bắt lỗi đối với biến này, tức là biến này có thể gán bất kì giá trị nào, chỉ nên sử dụng trong trường hợp không biết trước hoặc chấp nhận mọi giá trị chấp nhận rủi ro.

VD:

let variable: any;

let variable; // mặc định không gán kiểu hoặc giá trị sẽ là any

## Kiểu unknown

Cũng giống như kiểu any, bạn có thể gắn bất kì giá trị gì cho nó, ví dụ bạn không thể gọi một hàm hoặc toán tử trên giá trị unknown.

VD:

let result: unknown;

result = [1,2,3];

const total = result.reduce((a: number, b:number ) => a + b, 0);

console.log(total);

Vì nó không phải mảng number nên bạn không thể sử dụng hàm reduce hay bất kì hàm nào bạn có thể ép kiểu.

VD:

let result: unknown;

result = [1, 2, 3];

const total = (result as number[]).reduce((a: number, b: number) => a + b, 0);

console.log(total); // 6

So sánh sự khác biệt giữa any và unknown.

|  |  |  |
| --- | --- | --- |
| Tính năng | any | unknown |
| Mức độ an toàn | Không an toàn | An toàn |
| Thực thi | Thực thi không cần kiểm tra | Cần phải xác nhận loại trước khi thực thi |
| Trường hợp sử dụng | Sử dụng cho các giá trị động nhưng không an toàn | Sử dụng cho các giá trị động và an toàn nhưng cần xác thực trước khi sử dụng |
| Kiểm soát lỗi | Không | Có |
| Trường hợp sử dụng | Khi chuyển basecode từ JS sang TS | Khi lấy dữ liệu từ nguồn ngoài (gọi API, …) |

## Kiểu void

Dùng để khai báo cho hàm việc không trả về bất kì giá trị gì, hàm đơn giản chỉ thực thi và kết thúc.

function log(message): void {

console.log(messsage);

}

Tuy nhiên nó gán cho nó return cũng sẽ không gây lỗi nhưng có thể tiềm ẩn rủi ro không kiểm soát lỗi được.

## Kiểu never

Kiểu never là chỉ những giá trị không bao giờ xảy ra, một hàm có kiểu trả về là nerver sẽ không kết thúc bình thường.

Trường hợp sử dụng:

* Một hàm xác định luôn throw lỗi.

function throwError(message: string): never {

throw new Error(message);

}

* Hàm là vòng lặp vô hạn.

function infiniteLoop(): never {

while (true) {

console.log("Running forever...");

}

}

## Kiểu union

Là kiểu có thể nhận nhiều kiểu dữ liệu khác nhau bằng việc sử dụng dấu gạch đứng | .

let id: number | string;

id = 101; // Hợp lệ

id = "USER101"; // Hợp lệ

id = true; // Lỗi: boolean không nằm trong union type

## Kiểu dữ liệu chuỗi kí tự

Là kiểu chỉ gán duy nhất cho 1 chuỗi kí tự.

VD:

let click: 'click';

click = 'click'; // Hợp lệ

click = 'dblclick'; // Không hợp lệ

## Kiểu biệt danh

Cho phép bạn tạo một tên mới dựa kiểu dữ liệu đã tồn tại. Thường dùng trong những hệ thống phức tạp nhiều loại trùng tên nhau nhưng khác nội dung nên có thể đặt tên mới để phân biệt chúng giúp dễ đọc hơn và tránh xung đột.

* Kiểu nguyên thuỷ.

type Name: string;

let firstName: Name;

let lastName: Name;

* Kiểu đối tượng.

type Person = {

name: string;

age: number;

};

let person: Person = {

name: 'John',

age: 25

};

* Kiểu union.

type alphanumeric = string | number;

let input: alphanumeric;

input = 100; // valid

input = 'Hi'; // valid

input = false; // Compiler error

* Kiểu giao nhau.

type Personal = {

name: string;

age: number;

};

type Contact = {

email: string;

phone: string;

};

type Candidate = Personal & Contact;

let candidate: Candidate = {

name: "Joe",

age: 25,

email: "joe@example.com",

phone: "(408)-123-4567"

};

# KIỂM SOÁT LUỒNG TRẠNG THÁI

Phần này tương tự như trong Javascript.

* if else
* switch case
* for
* while
* do while
* break
* continue

# HÀM

## Hàm

Hàm là khối code có thể đọc được, bảo trì và tái sử dụng, cũng giống như javascript, tuy nhiên typescript sẽ yêu cầu kiểu dữ liệu.

VD: Trong Javascript:

function sum(a, b) {

return a + b;

}

Còn trong Typescript

function sum(a: number, b: number): number {

return a + b;

}

## Kiểu dữ liệu hàm

Là quy định kiểu trả về của hàm.

VD: : number

function sum(a: number, b: number): number {

return a + b;

}

Hàm sẽ có kiểu trả về là number.

## Tham số tuỳ chọn

Đối với tham số có thể có hoặc không, ta dùng dấu ? để xác định nó có thể có hoặc không.

VD:

function multiply(a: number, b: number, c?: number): number {

if (typeof c !== 'undefined') {

return a \* b \* c;

}

return a \* b;

}

## Tham số mặc định

Khi bạn muốn khi có tham số thì lấy giá trị đó, còn khi không có thì tham số đó mặc định là 1 giá trị nào đó:

VD:

function sum(a: number, b: number = 2): number {

return a + b;

}

console.log(sum(2)); // 4

console.log(sum(2,3)); // 5

## Tham số rest

Typescript cho phép một hàm chấp nhận không hoặc nhiều đối số của kiểu dữ liệu cụ thể. Yêu cầu:

* Một hàm chỉ nhận một tham số rest.
* Tham số res xuất hiện cuối cùng trong danh sách tham số.
* Kiểu dữ liệu của rest là mảng kiểu dữ liệu.

VD:

function getTotal(...numbers: number[]): number {

let total = 0;

numbers.forEach((num) => total += num);

return total;

}

getTotal(2, 3) // 5

getTotal(2, 3, 5) // 10

rest có thể là mảng chứa nhiều kiểu dữ liệu

## Hàm quá tải

Nó tương tự như tính đa hình trong lập trình hướng đối tượng (nhưng khác chỗ typescript/javascript chỉ cho tạo 1 hàm có 1 tên duy nhất).

VD:

// Các chữ ký hàm (Overload Signatures)

function double(value: number): number;

function double(value: string): string;

// Hàm thực tế (Implementation)

function double(value: any): any {

if (typeof value === "number") {

return value \* 2;

} else if (typeof value === "string") {

return value + value;

}

}

console.log(double(10)); // Output: 20

console.log(double("Hi")); // Output: "HiHi"

console.log(double(true)); // Lỗi: Không có overload nào cho kiểu Boolean

# LỚP

## Lớp

Dùng để tạo đối tượng theo mô hình hướng đối tượng (OOP). Sử dụng từ khoá class để định nghĩa một lớp.

VD:

class Person {

ssn: string;

firstName: string;

lastName: string;

constructor(ssn: string, firstName: string, lastName: string) {

this.ssn = ssn;

this.firstName = firstName;

this.lastName = lastName;

}

getFullName(): string {

return `${this.firstName} ${this.lastName}`;

}

}

## Trình sửa đổi truy cập

Typescript cung cấp 3 kiểu truy cập sửa đổi.

* public: thuộc tính hoặc phương thức có thể truy cập vào đối tượng để lấy giá trị (mặc định).
* private: thuộc tính hoặc phương thức chỉ có thể được sử dụng trong class, ngoài phạm vi không thể truy cập được.
* protect: thuộc tính hoặc phương thức chỉ có thể sử dụng trong class hoặc lớp kế thừa

## Công cụ sửa đổi readonly

## Getters & Setters

## Kế thừa

## Phương thức tĩnh và thuộc tính

## Lớp trừu tượng

# INTERFACE

## Interface

## Interface mở rộng

## Interface và lớp trừu tượng

# LOẠI NÂNG CAO

## Kiểu giao điểm

## Kiểu bảo vệ

## Kiểu xác nhận

# GENERIC

## Giới thiệu về Typescript Generics

## Hằng Generics

## Lớp Generics

## Giao diện Generics

# MODULES

## Typescript Modules