FPT Education

FPT POLYTECHNIC

cdio™

Conceive Design Implement Operate

THỰC HỌC – THỰC NGHIỆP

LẬP TRÌNH TYPESCRIPT

ADVANCED TYPES

GENERICS

http://www.poly.edu.vn

Mục tiêu

- Intersection type
- More on type guards
- Discriminated Unions
- Type Casting

Mục tiêu

- First generic method
- Another generic function
- Keyof Constraint
- Generic class

Phần 1
Advanced types

INTERSECTION TYPE

- ***Intersection type***: giao (intersection) các kiểu dữ liệu lại với nhau.
- Kiểu intersection sẽ có tất cả các thuộc tính từ hai kiểu giao nhau
- Ví dụ

```
type Combinable = string | number;
type Numeric = number | boolean;

type Universal = Combinable & Numeric;
```

TYPE GUARD

- Type guard: cho phép thu hẹp type(loại) của đối tượng trong khối điều kiện
  - Typeof
  - Instance of
  - in

TYPE GUARD

- typeof

```
function add(a: Combinable, b: Combinable) {
  if (typeof a === 'string' || typeof b === 'string') {
    return a.toString() + b.toString();
  }
  return a + b;
}
```

TYPE GUARD

- In: kiểm tra sự tồn tại của một thuộc tính trên một đối tượng.

```
type UnknownEmployee = Employee | Admin;

function printEmployeeInformation(emp: UnknownEmployee) {
  console.log('Name: ' + emp.name);
  if ('privileges' in emp) {
    console.log('Privileges: ' + emp.privileges);
  }
  if ('startDate' in emp) {
    console.log('Start Date: ' + emp.startDate);
  }
}
```

TYPE GUARD

Instanceof và class

```
class Car {
  drive() {
    console.log('Driving...');
  }
}

class Truck {
  drive() {
    console.log('Driving a truck...');
  }

  loadCargo(amount: number) {
    console.log('Loading cargo ...' + amount);
  }
}
```

```
type Vehicle = Car | Truck;

const v1 = new Car();
const v2 = new Truck();

function useVehicle(vehicle: Vehicle) {
  vehicle.drive();
  if (vehicle instanceof Truck) {
    vehicle.loadCargo(1000);
  }
}
```

DISCRIMINATED UNIONS

- Discriminated union: được sử dụng khi trong class, interface có thành phần dữ liệu là literal
- Giúp phân biệt với các thành phần dữ liệu là union

```
interface Bird {
  type: 'bird';
  flyingSpeed: number;
}

interface Horse {
  type: 'horse';
  runningSpeed: number;
}

type Animal = Bird | Horse;
```

TYPE CASTING

- Type casting: cho phép chuyển đổi một biến từ kiểu này sang kiểu khác
- Sử dụng từ khoá **as** hoặc toán tử **< >**

- Ví dụ

```
let input = document.querySelector('input[type="text"]') as HTMLInputElement;
```


demo

Phần 2
Generics

VẤN ĐỀ

- Xét function

```
function identity(arg: number): number {
  return arg;
}
```

- Vấn đề
  - Đầu vào: number
  - Trả về: number

  => Không thể mở rộng / tái sử dụng
  - Dùng any: ????

  => Không thể phán đoán được kiểu trả về là gì từ đó thực hiện các xử lý tiếp theo.

GIẢI QUYẾT VẤN ĐỀ

- Dùng Generic
  - Tạo ***type variable*** (type parameters, generic parameter) bằng cách tạo biến T đặt trong <>
  - Biến T bây giờ sẽ trở thành một placeholder cho 1 kiểu giá trị mà chúng ta muốn truyền vào hàm
  - Biến T được viết tắt của từ type. Thực tế có thể dung bất cứ tên gì

```
function identity<T> (arg: T): T {
  return arg;
}
```

GENERICS

- Generics: là công cụ giúp tạo ra các components có thể tái sử dụng. Có thể tạo ra component có thể hoạt động trên nhiều loại dữ liệu khác nhau thay vì 1 loại dữ liệu duy nhất
- Generic type là việc cho phép truyền ***type*** vào components (function, class, interface) như là 1 tham số

- Từ khoá extends: giới hạn phạm vi của type variable

```
function identity<T extends object> (arg: T): T {
  return arg;
}
```

- Default value

```
function identity<T = string> (arg: T): T {
  return arg;
}
```

GENERIC FUNCTION

- Single type variable

```
function identity<T> (arg: T): T {
  return arg;
}
```

- Multiple type variables

```
function merge<T,U>(objA: T, objB: U) {
  return Object.assign(objA, objB);
}
```

GENERIC FUNCTION

- Array method

```
function displayNames<T>(names:T[]): void {
  console.log(names.join(", "));
}
```

- **keyof** Constraits (ràng buộc)

```
function getProperty<T, K extends keyof T>(obj: T, key: K) {
  return obj[key];
}
```

Generic Interface

- Generic interface

```
interface IProcessor<T>
{
    result:T;
    process(a: T, b: T) => T;
}
```

- Generic interface như là 1 function

```
interface KeyValueProcessor<T, U>
{
    (key: T, val: U): void;
};
```

Generic Class

- Tham số đặt trong <> sau tên class

```
class KeyValuePair<T,U>
{
    private key: T;
    private val: U;

    setKeyValue(key: T, val: U): void {
        this.key = key;
        this.val = val;
    }

    display():void {
        console.log(`Key = ${this.key}, val = ${this.val}`);
    }
}
```

GENERIC CLASS IMPLEMENTS GENERIC INTERFACE

- Ví dụ

```
interface IKeyValueProcessor<T, U>
{
    process(key: T, val: U): void;
};

class kvProcessor<T, U> implements IKeyValueProcessor<T, U>
{
    process(key:T, val:U):void {
        console.log(`Key = ${key}, val = ${val}`);
    }
}
```


demo

SUMARRY

- ☑ Intersection type
- ☑ More on type guards
- ☑ Discriminated Unions
- ☑ Type Casting

- ☑ First generic method
- ☑ Another generic function
- ☑ Keyof Constraint
- ☑ Generic class


thank
you!