Chapter 2: Arrays, Tuples, Enums và Objects

Mô tả tổng quát

Chương này giới thiệu các cấu trúc dữ liệu quan trọng trong TypeScript để quản lý tập hợp dữ liệu một cách hiệu quả. Chúng ta sẽ học về:

- **Arrays**: Lưu trữ danh sách các giá trị cùng kiểu.
- **Tuples**: Mảng có số lượng phần tử cố định với kiểu cụ thể cho từng phần tử.
- **Enums**: Tạo nhóm hằng số có tên để code dễ đọc và bảo trì.
- **Objects**: Định nghĩa kiểu cho các đối tượng với các thuộc tính cụ thể.

Mục tiêu là giúp bạn hiểu cách sử dụng các cấu trúc này và áp dụng chúng trong các tình huống thực tế.

Tiêu đề chương

Làm việc với Arrays, Tuples, Enums và Kiểu Đối Tượng.

Tóm tắt lý thuyết chính

1. Arrays (Mảng)

Arrays trong TypeScript cho phép lưu trữ nhiều giá trị cùng kiểu, giúp quản lý danh sách dữ liệu như danh sách sản phẩm, điểm số, hoặc tên người dùng. TypeScript cung cấp hai cách khai báo kiểu cho mảng:

Cách 1: Sử dụng `[]`

- Thêm `[]` sau kiểu dữ liệu của phần tử.
- Ví dụ: `number[]` là mảng chỉ chứa số, `string[]` là mảng chỉ chứa chuỗi.

```
let numbers: number[] = [1, 2, 3, 4, 5]; // Mảng số
let names: string[] = ["Alice", "Bob", "Charlie"]; // Mảng chuỗi
```

```
// numbers.push("six"); // Lỗi: TypeScript phát hiện kiểu sai (chuỗi thay vì
```

Cách 2: Sử dụng Generic `Array<elementType>`

- Tương đương với cách trên, nhưng cú pháp rõ ràng hơn trong các trường hợp phức tạp.

```
let scores: Array<number> = [100, 90, 85]; // Mảng số
let flags: Array<boolean> = [true, false, true]; // Mảng boolean
```

Ưu điểm:

- Linh hoạt về kích thước (có thể thêm/xóa phần tử).
- Hỗ trợ các phương thức JavaScript như `push`, `pop`, `map`, `filter`, v.v.
- TypeScript kiểm tra kiểu để tránh lỗi khi thao tác.

Hạn chế:

- Chỉ nên chứa các phần tử cùng kiểu để đảm bảo tính nhất quán.

Ví dụ thực tế: Quản lý danh sách tên sản phẩm trong cửa hàng:

```
let products: string[] = ["Laptop", "Phone", "Tablet"];
products.push("Headphones"); // Thêm sản phẩm mới
console.log(products); // ["Laptop", "Phone", "Tablet", "Headphones"]
```

2. Tuples

Tuples là một dạng mảng đặc biệt, có số lượng phần tử cố định và kiểu dữ liệu của từng phần tử được xác định trước. Tuples rất hữu ích khi bạn muốn biểu diễn dữ liệu có cấu trúc rõ ràng, như tọa độ (`[x, y]`) hoặc thông tin người dùng (`[name, age]`).

Khai báo Tuple

- Xác định kiểu cho từng phần tử theo thứ tự.

```
let userProfile: [string, number, boolean]; // Tuple chứa tên (string), tuổi
userProfile = ["John Doe", 30, true]; // OK
// userProfile = [30, "John Doe", true]; // Lỗi: Sai kiểu
// userProfile = ["Jane Doe", 25]; // Lỗi: Thiếu phần tử
```

Truy cập phần tử

- Sử dụng chỉ số (index) như mảng thông thường.

```
console.log(userProfile[0]); // "John Doe" (tên)
console.log(userProfile[1]); // 30 (tuổi)
// userProfile[3] = "test"; // Lỗi: Vượt quá số phần tử
```

Ưu điểm:

- Đảm bảo cấu trúc dữ liệu cố định (số lượng và kiểu).
- Hữu ích khi trả về nhiều giá trị từ hàm.

Hạn chế:

- Không linh hoạt như Arrays (không thể thêm/xóa phần tử).
- Phải nhớ thứ tự kiểu của từng phần tử.

Ví dụ thực tế: Biểu diễn tọa độ 2D:

```
type Point2D = [number, number]; // Tuple cho tọa độ [x, y]
let point: Point2D = [10, 20];
```

```
console.log(`Tọa độ: x=${point[0]}, y=${point[1]}`); // Tọa độ: x=10, y=20
```

3. Enums (Liệt kê)

Enums cho phép định nghĩa một nhóm hằng số có tên, giúp code dễ đọc và dễ bảo trì hơn. Thay vì sử dụng số hoặc chuỗi "magic" (ví dụ: `0` hoặc `"ERROR"`), bạn có thể dùng tên hằng số như `Direction.Up` hoặc `LogLevel.ERROR`.

Numeric Enums

- Mặc định, giá trị bắt đầu từ `0` và tăng dần.

```
enum Direction {
    Up,     // 0
    Down,   // 1
    Left,   // 2
    Right   // 3
}
let currentDirection: Direction = Direction.Up;
console.log(currentDirection); // 0
if (currentDirection === Direction.Up) {
    console.log("Đang di chuyển lên!"); // Đang di chuyển lên!
}
```

- Có thể gán giá trị cụ thể:

```
enum HttpStatus {
    OK = 200,
    CREATED = 201,
    BAD_REQUEST = 400,
    NOT_FOUND = 404
}
console.log(HttpStatus.OK); // 200
```

String Enums

- Mỗi giá trị là một chuỗi hằng, dễ debug hơn.

```
enum LogLevel {
    INFO = "INFO",
    WARNING = "WARNING",
    ERROR = "ERROR",
    DEBUG = "DEBUG"
}
function logMessage(level: LogLevel, message: string): void {
    console.log(`[${level}] - ${message}`);
}
logMessage(LogLevel.INFO, "Người dùng đăng nhập."); // [INFO] - Người dùng đ
```

Const Enums (Bổ sung)

- Sử dụng từ khóa `const` để tối ưu hóa mã biên dịch (TypeScript sẽ thay thế enum bằng giá trị thực tại thời điểm biên dịch).

```
const enum Theme {
    Light = "LIGHT",
    Dark = "DARK"
}
let currentTheme: Theme = Theme.Dark;
console.log(currentTheme); // "DARK"
// Mã JavaScript sau biên dịch sẽ không chứa định nghĩa enum, giảm kích thướ
```

Ưu điểm:

- Tăng tính dễ đọc và bảo trì.
- String enums dễ debug, numeric enums tự động tăng giá trị.
- Const enums tiết kiệm bộ nhớ và tối ưu hiệu suất.

Hạn chế:

- String enums yêu cầu gán giá trị rõ ràng.
- Const enums không thể truy cập ngược (reverse mapping) như numeric enums.

Ví dụ thực tế: Quản lý trạng thái giao diện:

```
enum UIState {
    LOADING = "LOADING",
    SUCCESS = "SUCCESS",
    ERROR = "ERROR"
}
let appState: UIState = UIState.LOADING;
console.log(`Trạng thái ứng dụng: ${appState}`); // Trạng thái ứng dụng: LOA
```

4. Objects (Kiểu đối tượng ẩn danh)

TypeScript cho phép định nghĩa "hình dạng" của đối tượng bằng cách khai báo các thuộc tính và kiểu tương ứng. Điều này rất hữu ích khi làm việc với dữ liệu phức tạp như thông tin người dùng, sản phẩm, hoặc bài viết.

Khai báo kiểu đối tượng

```
let product: { name: string; price: number; inStock: boolean };
product = { name: "Laptop Pro", price: 30000000, inStock: true };
// product = { name: "Mouse", price: "500000" }; // Lỗi: price phải là số
// product = { name: "Keyboard" }; // Lỗi: Thiếu price và inStock
```

Thuộc tính tùy chọn (`?`)

- Đánh dấu thuộc tính không bắt buộc bằng `?`.

```
let employee: {
    id: number;
```

```
    name: string;
    department?: string; // Tùy chọn
};
employee = { id: 1, name: "Alice" }; // OK
employee = { id: 2, name: "Bob", department: "Kỹ thuật" }; // OK
console.log(employee.department?.toUpperCase()); // KỸ THUẬT (hoặc undefined
```

Thuộc tính chỉ đọc (`readonly`)

- Ngăn thay đổi giá trị sau khi khởi tạo.

```
let book: {
    readonly isbn: string;
    title: string;
    author: string;
};
book = { isbn: "978-3-16-148410-0", title: "TypeScript Deep Dive", author: "
// book.isbn = "123"; // Lỗi: isbn là readonly
book.title = "Học TypeScript"; // OK
```

Excess Property Checks (Bổ sung)

- TypeScript kiểm tra các thuộc tính dư thừa khi gán đối tượng trực tiếp.

```
let user: { name: string; age: number };
user = { name: "Alice", age: 25, email: "alice@example.com" }; // Lỗi: email
// Cách khắc phục: Lưu đối tượng vào biến trước
let temp = { name: "Alice", age: 25, email: "alice@example.com" };
user = temp; // OK
```

Ưu điểm:

- Định nghĩa cấu trúc dữ liệu rõ ràng.

- Hỗ trợ thuộc tính tùy chọn và chỉ đọc.
- TypeScript kiểm tra kiểu để tránh lỗi.

Hạn chế:

- Cú pháp dài dòng khi định nghĩa nhiều đối tượng phức tạp (sẽ khắc phục bằng `interface` hoặc `type` sau).

Ví dụ thực tế: Mô tả thông tin người dùng:

```
let user: {
    readonly id: number;
    name: string;
    email?: string;
    greet: () => void;
};
user = {
    id: 1,
    name: "Alice",
    greet: function() {
        console.log(`Xin chào, tôi là ${this.name}!`);
    }
};
user.greet(); // Xin chào, tôi là Alice!
```

Code ví dụ tổng hợp

```
// Arrays: Quản lý danh sách sản phẩm
let products: string[] = ["Laptop", "Phone", "Tablet"];
products.push("Headphones");
console.log("Danh sách sản phẩm:", products); // ["Laptop", "Phone", "Tablet

// Tuples: Biểu diễn màu RGB
type RGBColor = [number, number, number];
let primaryRed: RGBColor = [255, 0, 0];
console.log("Màu đỏ (RGB):", primaryRed); // [255, 0, 0]
```

```typescript
// Numeric Enums: Quản lý trạng thái đơn hàng
enum OrderStatus {
    PENDING, // 0
    PROCESSING, // 1
    SHIPPED, // 2
    DELIVERED // 3
}
let myOrder: OrderStatus = OrderStatus.PROCESSING;
console.log("Trạng thái đơn hàng:", OrderStatus[myOrder]); // PROCESSING

// String Enums: Quản lý quyền truy cập
enum FileAccess {
    Read = "READ",
    Write = "WRITE",
    Execute = "EXECUTE"
}
let userAccess: FileAccess = FileAccess.Read;
console.log("Quyền truy cập:", userAccess); // READ

// Objects: Mô tả thông tin xe hơi
let car: {
    readonly make: string;
    model: string;
    year: number;
    color?: string;
    startEngine: () => void;
};
car = {
    make: "Toyota",
    model: "Camry",
    year: 2023,
    startEngine: function() {
        console.log(`${this.make} ${this.model} khởi động động cơ!`);
    }
};
car.startEngine(); // Toyota Camry khởi động động cơ!
car.color = "Bạc";
console.log("Xe:", car);
```

Danh sách bài tập

Bài 1: Trắc nghiệm - So sánh Array và Tuple

Tiêu đề: So sánh Array và Tuple
Mô tả: Kiểm tra hiểu biết về sự khác biệt giữa Array và Tuple.
Câu hỏi: Phát biểu nào đúng nhất về sự khác biệt giữa Array và Tuple?
A. Array chứa các phần tử khác kiểu, Tuple chỉ chứa cùng kiểu.
B. Tuple có số lượng phần tử cố định và kiểu xác định trước, Array linh hoạt hơn.
C. Array không thể thay đổi kích thước, Tuple có thể.
D. Tuple là bí danh của Array.
Đáp án: B
Giải thích: Tuple yêu cầu số lượng phần tử cố định và kiểu cụ thể cho từng vị trí, trong khi Array có thể thay đổi kích thước và thường chứa cùng kiểu.

Bài 2: Code - Thao tác với Array

Tiêu đề: Thực hành với Mảng
Mô tả: Khai báo mảng tên sinh viên, thêm phần tử mới và in ra console.
Yêu cầu:

- Khai báo mảng `studentNames: string[]` với ít nhất 2 tên.
- Thêm một tên mới bằng `push()`.
- In mảng ra console.

```
let studentNames: string[] = ["Lan Anh", "Minh Khôi"];
console.log("Ban đầu:", studentNames);
studentNames.push("Hoàng Nam");
console.log("Sau khi thêm:", studentNames);
```

Bài 3: Code - Tạo Enum cho trạng thái thanh toán

Tiêu đề: Thực hành với Enums
Mô tả: Định nghĩa enum `PaymentStatus` với các trạng thái `UNPAID`, `PAID`, `REFUNDED`, `FAILED`.
Yêu cầu:

- Định nghĩa enum với giá trị số cụ thể.
- Khai báo biến `currentPaymentStatus` và in giá trị.

```
enum PaymentStatus {
    UNPAID = 10,
    PAID = 20,
    REFUNDED = 30,
    FAILED = 40
}
let currentPaymentStatus: PaymentStatus = PaymentStatus.PAID;
console.log("Trạng thái:", PaymentStatus[currentPaymentStatus]); // PAID
```

Bài 4: Code - Tạo Tuple cho sách

Tiêu đề: Thực hành với Tuples
Mô tả: Khai báo tuple `bookInfo` chứa tiêu đề, tác giả, năm xuất bản.
Yêu cầu:

- Khai báo tuple `[string, string, number]`.
- Tạo biến `myBook` và in từng phần tử.

```
type BookInfoTuple = [string, string, number];
let myBook: BookInfoTuple = ["Lập Trình TypeScript", "John Doe", 2024];
console.log("Tiêu đề:", myBook[0]);
console.log("Tác giả:", myBook[1]);
console.log("Năm:", myBook[2]);
```

Bài 5: Code - Định nghĩa kiểu đối tượng bài viết

Tiêu đề: Thực hành với Objects
Mô tả: Định nghĩa kiểu cho đối tượng `article` với `id` (readonly), `title`, `content`, `author`, và `tags` (tùy chọn).
Yêu cầu:

- Tạo đối tượng `myArticle` và thử thay đổi `id`.
- Tạo đối tượng khác không có `tags`.

```
let myArticle: {
    readonly id: number;
    title: string;
    content: string;
    author: string;
    tags?: string[];
};
myArticle = {
    id: 101,
    title: "TypeScript là gì?",
    content: "TypeScript là superset của JavaScript...",
    author: "Dev A",
    tags: ["typescript", "javascript"]
};
console.log("Bài viết:", myArticle);
// myArticle.id = 102; // Lỗi: readonly
let anotherArticle: typeof myArticle = {
    id: 102,
    title: "Ưu điểm TypeScript",
    content: "Kiểm tra kiểu tĩnh...",
    author: "Dev B"
};
console.log("Bài viết khác:", anotherArticle);
```