TypeScript: Interface vs Type - সম্পূর্ণ গাইড

মূল পার্থক্য

TypeScript-এ `interface` এবং `type` দুটোই object এর structure define করার জন্য ব্যবহার হয়, কিন্তু তাদের মধ্যে কিছু গুরুত্বপূর্ণ পার্থক্য রয়েছে।

Interface

Interface হলো object এর shape বা structure define করার একটি উপায়। এটি মূলত contract এর মতো কাজ করে।

Interface এর বৈশিষ্ট্য:

- **Extensible**: Interface extend করা যায়
- **Mergeable**: একই নামের multiple interface merge হয়ে যায়
- **Declaration Merging**: Same name এর interface গুলো automatically merge হয়

Interface এর উদাহরণ:

```typescript
```

```typescript
// Basic Interface
interface User {
  name: string;
  age: number;
  email: string;
}

// Interface Extension
interface Admin extends User {
  role: string;
  permissions: string[];
}

// Declaration Merging
interface User {
  phone?: string; // এটি আগের User interface এর সাথে merge হবে
}

// ব্যবহার
const user: User = {
  name: "রহিম",
  age: 25,
  email: "rahim@example.com",
  phone: "01712345678" // optional property
};

const admin: Admin = {
  name: "করিম",
  age: 30,
  email: "karim@example.com",
  role: "super-admin",
  permissions: ["read", "write", "delete"]
};
```

Type

Type হলো আরো flexible একটি উপায় যা শুধু object নয়, যেকোনো type define করতে পারে।

Type এর বৈশিষ্ট্য:

- **More Flexible**: Union, Intersection, Primitive types সব handle করতে পারে
- **Cannot be extended**: Direct extension সম্ভব নয়

- **Cannot be merged**: Same name এর type দুটো থাকতে পারে না

Type এর উদাহরণ:

typescript

```typescript
// Basic Type
type User = {
  name: string;
  age: number;
  email: string;
};

// Union Type
type Status = "loading" | "success" | "error";

// Intersection Type
type AdminUser = User & {
  role: string;
  permissions: string[];
};

// Function Type
type EventHandler = (event: string) => void;

// Generic Type
type ApiResponse<T> = {
  data: T;
  status: number;
  message: string;
};

// ব্যবহার
const user: User = {
  name: "সালিম",
  age: 28,
  email: "salim@example.com"
};

const currentStatus: Status = "loading";

const adminUser: AdminUser = {
  name: "নাসির",
  age: 35,
  email: "nasir@example.com",
  role: "admin",
  permissions: ["read", "write"]
};
```

```typescript
const handleClick: EventHandler = (event) => {
  console.log(`Event: ${event}`);
};

const apiResponse: ApiResponse<User[]> = {
  data: [user],
  status: 200,
  message: "Success"
};
```

কখন কোনটা ব্যবহার করবেন?

Interface ব্যবহার করুন যখন:

1. **Object Structure Define করার জন্য**

```typescript
interface Product {
  id: number;
  name: string;
  price: number;
  category: string;
}
```

2. **Class Implementation এর জন্য**

```typescript
interface Drawable {
  draw(): void;
}

class Circle implements Drawable {
  draw() {
    console.log("Drawing a circle");
  }
}
```

3. **Library/API Design এর জন্য**

```typescript
```

```typescript
interface DatabaseConnection {
  connect(): Promise<void>;
  disconnect(): Promise<void>;
  query(sql: string): Promise<any>;
}
```

4. **Extension প্রয়োজন হলে**

```typescript
interface BaseEntity {
  id: string;
  createdAt: Date;
}

interface User extends BaseEntity {
  name: string;
  email: string;
}
```

Type ব্যবহার করুন যখন:

1. **Union Types এর জন্য**

```typescript
type Theme = "light" | "dark" | "auto";
type Size = "small" | "medium" | "large";
```

2. **Complex Type Compositions**

```typescript
type UserAction =
  | { type: "LOGIN"; payload: { username: string; password: string } }
  | { type: "LOGOUT" }
  | { type: "UPDATE_PROFILE"; payload: { name: string; email: string } };
```

3. **Utility Types**

```typescript
```

```typescript
type PartialUser = Partial<User>;
type RequiredUser = Required<User>;
type UserEmail = Pick<User, "email">;
```

4. **Function Types**

```typescript
type ValidateFunction = (value: string) => boolean;
type AsyncHandler<T> = (data: T) => Promise<void>;
```

Performance এবং Best Practices

Performance:

- Interface সামান্য দ্রুত কারণ TypeScript compiler এটি optimize করতে পারে
- Type alias গুলো inline হয়ে যায়

Best Practices:

1. **Object shapes এর জন্য Interface প্রাধান্য দিন**

```typescript
// ভাল
interface UserConfig {
  apiUrl: string;
  timeout: number;
}

// এড়িয়ে চলুন
type UserConfig = {
  apiUrl: string;
  timeout: number;
};
```

2. **Union/Intersection এর জন্য Type ব্যবহার করুন**

```typescript
```

```typescript
// ভাল
type Status = "idle" | "loading" | "success" | "error";

// সম্ভব নয়
interface Status extends "idle" | "loading" {} // Error!
```

3. **Consistent naming convention অনুসরণ করুন**

```typescript
// Interface এর জন্য PascalCase
interface UserInterface {
  name: string;
}

// Type এর জন্য PascalCase + descriptive suffix
type UserType = {
  name: string;
};
```

সারসংক্ষেপ

বৈশিষ্ট্য	Interface	Type
Object Definition	✅	✅
Union Types	❌	✅
Intersection	❌ (extends ব্যবহার করুন)	✅
Declaration Merging	✅	❌
Computed Properties	❌	✅
Class Implementation	✅	❌
Performance	সামান্য ভাল	ভাল

সাধারণ নিয়ম: Object structures এর জন্য `interface`, অন্য সব complex types এর জন্য `type` ব্যবহার করুন।