Item 13.

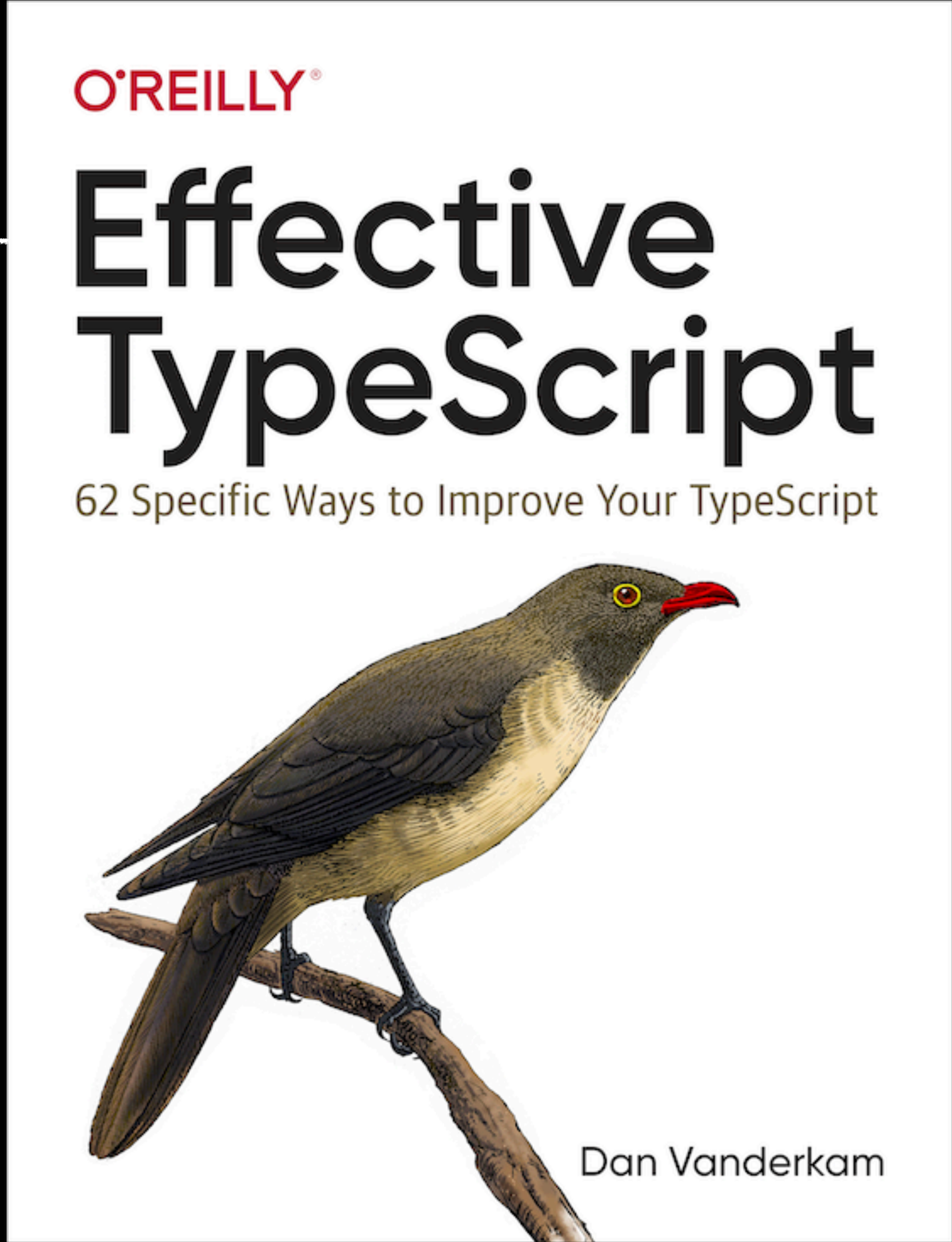
타입과 인터페이스의 차이점 알기

Effective TypeScript

@Bori-github(이보리)

목 차

1. 명명된 타입 정의하기
2. 인터페이스 선언과 타입 별칭 선언
   - 공통점
   - 차이점
3. 결론



1. 명명된 타입 정의하기

명명된 타입(named type)을 정의하는 방법

1. 인터페이스(interface)

- 상호 간에 정의한 약속 혹은 규칙
- interface 키워드를 사용하여 타입 정의

```typescript
interface Todo {
  id: number;
  content: string;
  completed: boolean;
}

// 변수 todo의 타입으로 Todo 인터페이스를 선언
let todo: Todo;

// 변수 todo는 Todo 인터페이스를 준수해야 함
todo = { id: 1, content: 'typescript', completed: false };
```

객체의 필드와 메서드를 선언하고,

이를 구현한 클래스나 객체가 인터페이스의 규칙을 따르도록 강제하여 일관성 유지

1. 명명된 타입 정의하기

명명된 타입(named type)을 정의하는 방법

2. 타입 별칭(type alias)

- 원시 타입이나 유니언 타입, 튜플 등등 모든 변수에 자유롭게 활용할 수 있는 커스텀 타입
- type 키워드를 사용하여 타입 정의

```typescript
type Todo = {
  id: number;
  content: string;
  completed: boolean;
}

const todo: Todo = { id: 1, content: 'typescript', completed: false };
```

기존에 존재하는 타입에 새로운 이름을 부여하여 복잡한 타입을 단순하게 줄여서 사용 가능

2. 인터페이스 선언과 타입 별칭 선언

공통점 1. 잉여 속성 체크

type Person 또는 interface Person를 추가 속성과 함께 할당

⇒ 잉여 속성 체크가 수행되며 오류 발생

2. 인터페이스 선언과 타입 별칭 선언

공통점 2. 인덱스 시그니처

```ts
type Dict = { [key: string]: string };

interface Dict {
  [key: string]: string;
}
```

공통점 3. 제너릭

```ts
type Pair<T> = {
  first: T;
  second: T;
}

interface Pair<T> {
  first: T;
  second: T;
}
```

⇒ 인터페이스와 타입 별칭 모두 인덱스 시그니처와 제너릭 사용 가능

2. 인터페이스 선언과 타입 별칭 선언

공통점 4. 함수 타입 정의

```ts
type FnType = (x: number) => string;

interface FnInterface {
  (x: number): string;
}

const toStrT: FnType = x => '' + x;  // OK
const toStrI: FnInterface = x => '' + x;  // OK
```

```ts
type FnWithProperties = {
  (x: number): number;
  prop: string;
}

interface FnWithProperties {
  (x: number): number;
  prop: string;
}
```

⇒ 인터페이스와 타입 별칭 모두 함수 타입 정의 가능

함수 타입에 추가적인 속성이 있다면 오른쪽 코드와 같이 작성 가능

2. 인터페이스 선언과 타입 별칭 선언

공통점 5. 타입 확장

```ts
type State = {
  name: string;
  capital: string;
}

interface StateWithPop extends State {
  population: number;
}
```

- 인터페이스는 타입 별칭을 확장 가능

```ts
interface State {
  name: string;
  capital: string;
}

type StateWithPop = State & { population: number; };
```

- 타입 별칭은 인터페이스를 확장 가능

⇒ 인터페이스와 타입 별칭은 서로 확장 가능

2. 인터페이스 선언과 타입 별칭 선언

공통점 5. 클래스 구현(implements)

```ts
type StateType = {
  name: string;
  capital: string;
}

class State implements StateType {
  name: string = '';
  capital: string = '';
}
```

```ts
interface StateInterface {
  name: string;
  capital: string;
}

class State implements StateInterface {
  name: string = '';
  capital: string = '';
}
```

⇒ 클래스를 구현할 때 타입 별칭과 인터페이스 모두 사용 가능

2. 인터페이스 선언과 타입 별칭 선언

차이점 1. 타입 확장 - 인터페이스

```ts
interface Person {
  name: string;
  age: number;
}

interface Employee extends Person {
  salary: number;
}
```

인터페이스는 extends 키워드 사용하여 상속을 통한 타입 확장 가능

2. 인터페이스 선언과 타입 별칭 선언

차이점 1. 타입 확장 - 인터페이스

```ts
type Input = string;
type Output = string;

// 유니온 타입
type InputOrOutput = Input | Output;

// 인터페이스는 유니온 타입을 확장할 수 없다.
interface ExtendsInterface extends InputOrOutput {
  more: string;
}

// 유니온 타입을 하나의 변수명으로 매핑은 가능하다.
interface ExtendsInterface {
  more: InputOrOutput;
}
```

인터페이스는 개체 형식 또는 정적으로 알려진 멤버가 포함된 개체 형식의 교집합만 확장할 수 있습니다. ts(2312)

```ts
type InputOrOutput = string
```

유니온 또는 인터섹션 활용 불가

⇒ 타입을 확장할 수 있으나, **복잡한 타입을 표현하거나 확장이 불가**

2. 인터페이스 선언과 타입 별칭 선언

차이점 1. 타입 확장 - 타입 별칭

```ts
type Person = {
  name: string;
  age: number;
};

type Employee = Person & {
  salary: number;
};
```

타입 별칭은 extends 키워드를 통한 타입 확장 불가

2.7 버전부터 인터섹션 타입의 등장으로 & 기호를 이용하여 타입 결합 가능

2. 인터페이스 선언과 타입 별칭 선언

차이점 1. 타입 확장 - 타입 별칭

```ts
type Input = string;
type Output = string;

// 유니온 타입
type InputOrOutput = Input | Output;
// 유니온 타입에 인터섹션을 이용한 name 속성을 붙인 타입
type NamedVariable = (Input | Output) & { name: string };

```

유니온 타입, 매핑된 타입(mapped type), 조건부 타입(conditional type)을 활용

⇒ 복잡한 타입 표현 가능

2. 인터페이스 선언과 타입 별칭 선언

차이점 2. 튜플과 배열 타입

- 타입 별칭 ✅

```ts
type Pair = [number, number];
type StringList = string[];
type NamedNums = [string, ...number[]];

```

- 인터페이스

```ts
interface Tuple {
  0: number;
  1: number;
  length: 2;
}
const t: Tuple = [10, 20];  // OK
```

⇒ concat 같은 튜플의 메서드를 사용 불가

2. 인터페이스 선언과 타입 별칭 선언

차이점 3. 선언 병합

선언 병합 : 컴파일러가 같은 이름으로 선언된 개별적인 선언 두 개를 하나의 정의로 합치는 것

- 인터페이스 : 선언 병합 가능

⇒ 선언 병합을 이용한 보강(augment)이 가능

- 타입 별칭 : 선업 병합 불가능

⇒ 타입 별칭은 기존 타입에 추가적인 보강이 필요 없는 경우에만 사용

```ts
interface Person {
  name: string;
  gender: string;
}

interface Person {
  job: boolean;
}

const bori: Person = {
  name: 'Bori',
  gender: 'Female',
  job: false
};
```

4. 결론

타입을 정의는 **타입 별칭**과 **인터페이스** 두 가지 방법으로 할 수 있습니다.
두 문법은 비슷한 역할을 하지만 명확한 차이점이 있습니다.

인터페이스는 선언 병합을 통한 확장성이 좋습니다. 따라서 일반적으로는 인터페이스를 사용하고, 유니온이나 튜플과 같은 타입이 필요한 경우에만 타입 별칭을 사용하는 것이 좋습니다.
TypeScript 공식 문서에서도 특별한 상황이 아니라면 인터페이스 사용을 권장하고 있습니다.

프로젝트에서 어떤 문법을 사용할 지 결정해야할 때 한 가지 일관된 스타일을 확립하고, 확장이 필요한지 고려해야 합니다.

감사합니다 :)

Effective TypeScript

@Bori-github(이보리)