인터페이스의 유니온을 사용하기

Item 32

ryan-dia(강철원)

유니온의 인터페이스

```ts
type FillPaint = unknown;
type LinePaint = unknown;
type PointPaint = unknown;
type FillLayout = unknown;
type LineLayout = unknown;
type PointLayout = unknown;
interface Layer {
  layout: FillLayout | LineLayout | PointLayout;
  paint: FillPaint | LinePaint | PointPaint;
}
```

인터페이스의 유니온

```ts
interface FillLayer {
  layout: FillLayout;
  paint: FillPaint;
}
interface LineLayer {
  layout: LineLayout;
  paint: LinePaint;
}
interface PointLayer {
  layout: PointLayout;
  paint: PointPaint;
}
type Layer = FillLayer | LineLayer | PointLayer;
```

속성 간의 관계가 더 명확해짐

📌 유니온의 인터페이스

```ts
interface MyInterface {
  id: number | string;
  name: string;
}
```

```ts
type Status = 'success' | 'error' | 'loading';

interface MyInterface {
  status: Status;
  message: string;
}
```

태그된 유니온(Tagged Union)

유니온에 타입에 태그라는 추가적인 정보를 사용하여 다양한 타입을 구분하는 방법이다.
각각의 타입은 고유한 태그 값을 가지며, 이를 사용하여 해당 값이 어떤 타입인지 식별할 수 있다.

```ts
type Shape = Square | Circle | Triangle;

interface Square {
  kind: "square";
  size: number;
}

interface Circle {
  kind: "circle";
  radius: number;
}

interface Triangle {
  kind: "triangle";
  base: number;
  height: number;
}
```

위의 예시에서는 각각의 인터페이스는 'kind'라는 태그를 가진다.

```ts
function calculateArea(shape: Shape): number {
  switch (shape.kind) {
    case "square":
      return shape.size * shape.size;
    case "circle":
      return Math.PI * shape.radius * shape.radius;
    case "triangle":
      return (shape.base * shape.height) / 2;
  }
}
```

이렇게 상태 기반의 로직을 처리하는 데 유용하다.

📌 인터페이스의 유니온

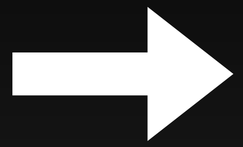
태그된 유니온(구분된 유니온)

```
interface FillLayer {
  type: 'fill';
  layout: FillLayout;
  paint: FillPaint;
}
interface LineLayer {
  type: 'line';
  layout: LineLayout;
  paint: LinePaint;
}
interface PointLayer {
  type: 'paint';
  layout: PointLayout;
  paint: PointPaint;
}
type Layer = FillLayer | LineLayer | PointLayer;
```

```
function drawLayer(layer: Layer) {
  if (layer.type === 'fill') {
    const {paint} = layer;  // Type is FillPaint
    const {layout} = layer;  // Type is FillLayout
  } else if (layer.type === 'line') {
    const {paint} = layer;  // Type is LinePaint
    const {layout} = layer;  // Type is LineLayout
  } else {
    const {paint} = layer;  // Type is PointPaint
    const {layout} = layer;  // Type is PointLayout
  }
}
```

📌 인터페이스의 유니온

```ts
interface Person {
  name: string;
  // 다음은 둘 다 동시에 있거나 동시에 없다.
  placeOfBirth?: string;
  dateOfBirth?: Date;
}
```



```ts
interface Person {
  name: string;
  birth?: {
    place: string;
    date: Date;
  }
}
```

🚀 활용

```ts
function eulogize(p: Person) {
  console.log(p.name);
  const {birth} = p;
  if (birth) {
    console.log(`was born on ${birth.date} in ${birth.place}.`);
  }
}
```

```ts
const alanT: Person = {
  name: 'Alan Turing',
  birth: {
// ~~~~ Property 'date' is missing in type
//      '{ place: string; }' but required in type
//      '{ place: string; date: Date; }'
    place: 'London'
  }
}
```

📌 인터페이스의 유니온

타입의 구조를 손 댈 수 없는 상황(ex API의 결과)

```ts
interface Name {
  name: string;
}

interface PersonWithBirth extends Name {
  placeOfBirth: string;
  dateOfBirth: Date;
}

type Person = Name | PersonWithBirth;
```

```ts
function eulogize(p: Person) {
  if ('placeOfBirth' in p) {
    p // Type is PersonWithBirth
    const {dateOfBirth} = p  // OK, type is Date
  }
}
```

📚 정리

- 유니온 타입의 속성을 여러 개 가지는 인터페이스에서는 속성 간의 관계가 분명하지 않기 때문에 실수가 자주 발생하므로 주의!
- 유니온의 인터페이스보다 인터페이스의 유니온이 더 정확하고 타입스크립트가 이해하기도 좋다
- 태그된 유니온은 타입스크립트와 매우 잘 맞기 때문에 자주 사용하도록 해보자