Nullable

This lesson explains the nullable mapped type.

We'll cover the following

Nullable mapped type#

Another possibility with the mapped type is that it can handle `null`. Similar to the optional `Partial<T>`, you can create your own `Nullable<T>`. To create your own type, the first step is to build it with no generic (without `<T>`, a plain interface or type) and then adjust it by adapting with a generic parameter (e.g. `<T>`).

For example, if we have the entity `Cat`, we might want to create a `NullableCat` type as a first step. The second step then would be to make it possible to null a `Dog` or any other object.

Intermediary step: Two interfaces into a specific nullable type#

If the second step seems challenging, we can add an intermediary step to create the entity with the null type (e.g. having a union that accepts the type

create the entity with the null type (e.g. having a union that accepts the type or null) which might help to visualize the problem.

Lines 11, 12, and 13 add `null` to all fields from the former interface defined on lines 1 to 5.

```
interface Cat {
  age: number;
  weight: number;
  numberOfKitty: number;
}

const cat1: Cat = { age:1, weight: 2, numberOfKitty: 0 };

// NullableCat1 have union with the null type. It allows to visualize the dual type.
interface NullableCat1 {
  age: number | null;
  weight: number | null;
  numberOfKitty: number | null;
}
```

Intermediary step with the union of null

From the long-winded version of the `NullableCat` interface, it is highly visible that every property is of the main type or null. The next step is to use the mapped type. The following code on line 10 shows a mapped type where all fields of Cat (`P`) are unioned with `null` .

```
interface Cat {
  age: number;
  weight: number;
  numberOfKitty: number;
}

const cat1: Cat = { age: 1, weight: 2, numberOfKitty: 0 };

// The nullable cat is now a mapped type.
type NullableCat = { [P in keyof Cat]: Cat[P] | null };

const cat2: NullableCat = { age: 1, weight: null, numberOfKitty: null };
```

Mapped type strongly associated with the interface, Cat

Generic nullable mapped type#

Generic nullable mapped type#

The second step is good: you want to avoid duplicating every field. In case of changes in the main type, `Cat`, you do not have to synchronize. However, if you need to have a similar nullable logic for an entity, `Dog`, the proliferation of the mapped type would be cumbersome.

```
interface Cat {
  age: number;
  weight: number;
  numberOfKitty: number;
}

const cat1: Cat = { age: 1, weight: 2, numberOfKitty: 0 };

// The mapped type that goes beyond Cat with generic <T>
type Nullable<T> = { [P in keyof T]: T[P] | null };

const cat2: Nullable<T> = { age: 1, weight: null, numberOfKitty: null };
```

Mapped type with generic

The transformation is to remove all mentions of a specific type by a generic identifier, e.g. `T`. Next, remove from the name, the mention of the entity, and add the generic bracket with the generic identifier.

Back

Partial

Next

Pick

Mark as Completed

Report an Issue