

Debugging Errors: Unintuitive Error Message

This lesson walks you through debugging a complex and unintuitive error message.


WE'LL COVER THE FOLLOWING ^

- Overview
- Digging deeper
- Fixing the error

Overview

In this example, we'll look at a complex and unintuitive error message.

Below you can find a short piece of code using the [RxJS](#) library. In short, RxJS is a library for creating and manipulating streams of data. The piece of code below is supposed to create a stream of five numbers, map it into a stream of those numbers multiplied by two, and finally return a stream that emits only the last number.

```
const source = of(1, 2, 3, 4, 5).pipe(
  map(x => x * 2),
  last //  Error!
);
```

When compiled, this code will yield a large and complex error message.

```
Argument of type '{ <T, D = T>(predicate?: null | undefined, defaultValu
e?: D | undefined): OperatorFunction<T, T | D>; <T, S extends T>(predicat
e: (value: T, index: number, source: Observable<T>) => value is S, default
Value?: S | undefined): OperatorFunction<...>; <T, D = T>(predicate: (valu
e: T, index: number, source: Observable<.....' is not assignable to parame
ter of type 'OperatorFunction<unknown, unknown>'.
```

```
Types of parameters 'predicate' and 'source' are incompatible.
```

```
Type 'Observable<unknown>' is not assignable to type 'null | undefine
d'
```

Type 'Observable<unknown>' is not assignable to type 'null'.ts(2345)

Digging deeper

Let's break down this error message.

```
Argument of type '{ <T, D = T>(predicate?: null, defaultValue?: D): OperatorFunction<T, T | D>; <T, S extends T>(predicate: (value: T, index: number, source: Observable<T>) => value is S, defaultValue?: S): OperatorFunction<T, S>; <T, D = T>(predicate: (value: T, index: number, source: Observable<...>) => boolean, defaultValue?: D): ...}' is not assignable to parameter of type 'OperatorFunction<unknown, unknown>'.
```

Looks like we're trying to pass something of a very complex type `{ <T, D = T>(predicate?: null, defaultValue?: D): OperatorFunction<T, T | D>;... }` where `OperatorFunction<Response, unknown>` is expected. Where does this huge type come from? When we [inspect the source code](#) of the `last` operators, we'll see three function definitions, none of them matching this lengthy type definition.

```
export declare function last<T, D = T>(predicate?: null, defaultValue?: D): OperatorFunction<T, T | D>;
export declare function last<T, S extends T>(predicate: (value: T, index: number, source: Observable<T>) => value is S, defaultValue?: S): OperatorFunction<T, S>;
export declare function last<T, D = T>(predicate: (value: T, index: number, source: Observable<T>) => boolean, defaultValue?: D): OperatorFunction<T, T | D>;
```

Let's look closer at this type.


```
{
  <T, D = T>(predicate?: null, defaultValue?: D): OperatorFunction<T, T | D>;
  <T, S extends T>(predicate: (value: T, index: number, source: Observable<T>) => value is S, defaultValue?: S): OperatorFunction<T, S>;
  <T, D = T>(predicate: (value: T, index: number, source: Observable<...>) => boolean, defaultValue?: D): ...
}
```

Although the compiler truncated the definition, we can still see that it's a **callable interface**. A value of this type can be called in at least three ways. If you look closely, you'll notice that these three call signatures match the three

overloads of `last!`. Therefore, we can see that this complex type is TypeScript's convoluted way of representing function overloads.

Fixing the error `#`

Great, now we finally know what the error message is trying to tell us. The type we're providing, a function that returns `OperatorFunction`, doesn't match the expected type, an `OperatorFunction`. We simply forgot to call `last`.

```
const source = of(1, 2, 3, 4, 5).pipe(  
  map(x => x * 2),  
  last() //  No errors!  
);
```

With this example, I'd like to show that despite sometimes imperfect error messages coming from TypeScript, it's always possible to debug the error. Sometimes it takes a bit more time and thought to dig deeper into the error message and break it down into smaller, understandable parts.

The next lesson looks into an example of using TypeScript with the React library.