Strict null checks in Typescript

Matěj Chalk
Front-end developer @FlowUp

matejchalk
matejchalk

The Billion Dollar Mistake

Tony Hoare

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. […] This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

NPE - from Java to Kotlin

- all Java objects may be null
  - no compiler checks
- NullPointerException is the most common production bug in Java apps

- Kotlin is a less verbose and safer alternative to Java
  - also compiles to JVM bytecode
  - null checking is performed at compile-time
- Google now recommends all new Android apps be developed in Kotlin

Javascript - `null` vs `undefined`

```
> const foo = {
    x: null,
    y: undefined,
  };
<· undefined
> foo.x
<· null
> foo.y
<· undefined
> foo.z
<· undefined
> 'y' in foo
<· true
> 'z' in foo
<· false
```

```
> const arr = [42];
<· undefined
> arr[1]
<· undefined
> arr.find(x => x > 100)
<· undefined
```

```
> document.getElementById('foo')
<· null
> document.querySelector('foo')
<· null
```

```
> typeof undefined
<· "undefined"
> typeof null
<· "object"
```

Top 10 Javascript errors

- Rollbar (error tracking company) aggregated production errors from thousands of projects
- 7 out of top 10 could be prevented by proper null handling

Rollbar

Top 10 Javascript errors (1-3)

➔ read a property or call a method on an undefined/null object

1. Uncaught TypeError: Cannot read property
2. TypeError: ‘undefined’ is not an object (evaluating)
3. TypeError: null is not an object (evaluating)

```
> let foo;
<· undefined
> foo.bar
✕ ▸Uncaught TypeError: Cannot read property 'bar' of undefined        VM906:1
    at <anonymous>:1:5
```

Top 10 Javascript errors (5-6)

➔ call an undefined method

5. TypeError: Object doesn't support property
6. TypeError: 'undefined' is not a function

```
> this.foo()
✕ ▸Uncaught TypeError: this.foo is not a function                VM1061:1
    at <anonymous>:1:6
```

Top 10 Javascript errors (8 & 9)

- reading length property for an undefined variable
  8. TypeError: Cannot read property ‘length’

```
> let testArray;
<· undefined
> testArray.length
✕ ▸Uncaught TypeError: Cannot read property 'length' of undefined      VM1121:1
      at <anonymous>:1:11
```

- try to set property of undefined
  9. Uncaught TypeError: Cannot set property

```
> let test = undefined;
<· undefined
> test.value = 0;
✕ ▸Uncaught TypeError: Cannot set property 'value' of undefined        VM1168:1
      at <anonymous>:1:12
```

Typescript strict mode

- Typescript has an opt-in philosophy to type checking
- to enable full type-safety, enable `strict` flag in `tsconfig.json`

Typescript strict mode - null checks

- null checking capabilities are also opt-in
- by default any type can have `null` or `undefined` assigned to it
- `strictNullChecks` flag must be set in order to account for `null` and `undefined`

Types used in running example

```
interface CommentModel {
  text: string;
  author?: AuthorModel;
}

interface AuthorModel {
  firstName: string;
  lastName: string;
}
```

Example of Typescript null checks

- with **`strictNullChecks`** (compile-time error):

```
function getAuthorName(comment: CommentModel): string {
  return `${comment.author.firstName} ${comment.author.lastName}`;
}
```

TS2532: Object is possibly 'undefined'.

Suppress with @ts-ignore Alt+Shift+Enter More actions... Alt+Enter

- without **`strictNullChecks`** (runtime error):

```
Uncaught TypeError: Cannot read property 'firstName' of undefined    VM1200:2
    at getAuthorName (<anonymous>:2:28)
    at <anonymous>:1:1
```

Handling null with non-null assertion

- ➔ ! operator casts nullable type to non-nullable

```
function getAuthorName(comment: CommentModel): string {
  return `${comment.author!.firstName} ${comment.author!.lastName}`;
}
```

```
Uncaught TypeError: Cannot read property 'firstName' of undefined    VM1200:2
    at getAuthorName (<anonymous>:2:28)
    at <anonymous>:1:1
```

- ➔ recommendation: enable **`no-non-null-assertion`** rule in TSLint

```
function getAuthorName(comment: CommentModel): string {
  return `${comment.author!.firstName} ${comment.author!.lastName}`;
}
```

TSLint: Forbidden non null assertion(no-non-null-assertion)

Suppress 'no-non-null-assertion' for current line Alt+Shift+Enter More actions... Alt+Enter

Handling nulls with type guards

```
function getAuthorName(comment: CommentModel): string | undefined {
  if (comment.author == null) {
    return undefined;
  }
  return `${comment.author.firstName} ${comment.author.lastName}`;
}
```

Narrowed to 'AuthorModel'

flow

Handling null with type guards - continued

```
function getAuthorName(comment: CommentModel): string | undefined {
  return comment.author == null
    ? undefined
    : `${comment.author.firstName} ${comment.author.lastName}`;
}
```

Narrowed to 'AuthorModel'

```
function getAuthorName(comment: CommentModel): string | undefined {
  return (
    comment.author && `${comment.author.firstName} ${comment.author.lastName}`
  );
}
```

Narrowed to 'AuthorModel'

up

flow

Fallback values

```
getAuthorName(comment) || 'Anonymous'
```

(careful: `""` and `0` are also falsy)

What about indirect null checks (e.g. filter)?

```
function extractAuthors(comments: CommentModel[]): AuthorModel[] {
  return comments
    .map(comment => comment.author)
    .filter(author => author != null);
}
```

TS2322: Type '(AuthorModel | undefined)[]' is not assignable to type 'AuthorModel[]'.
Type 'AuthorModel | undefined' is not assignable to type 'AuthorModel'.
Type 'undefined' is not assignable to type 'AuthorModel'.

Make 'extractAuthors' return '(Foo.AuthorModel | undefined)[]' Alt+Shift+Enter More actions... Alt+Enter

up

flow

Type predicates

```
function extractAuthors(comments: CommentModel[]): AuthorModel[] {
  return comments
    .map(comment => comment.author)
    .filter((author): author is AuthorModel => author != null);
}
```

up

flow

Type predicate helper function

- definition (e.g. in `utils.ts`):

```
export function isNotNullOrUndefined<T>(obj: T | null | undefined): obj is T {
  return obj != null;
}
```

- example of usage:

```
function extractAuthors(comments: CommentModel[]): AuthorModel[] {
  return comments
    .map(comment => comment.author)
    .filter(isNotNullOrUndefined);
}
```

Type predicates in RxJS

- RxJS filtering operators support type predicates too
  - e.g. `filter`, `find`, `first`, `skipWhile`, `takeWhile`

```
(property) AngularFireAuth.authState: Observable<User | null>

this.afAuth.authState
  .pipe(
    filter(isNotNullOrUndefined),
    switchMap( project: user => from(user.getIdToken())),
  )
  .subscribe( next: token => {
    localStorage.setItem('token', token);
  });
```

Typescript 3.7 - Assert signatures

```
function assertString(input: any): assert input is string {
  if (typeof input !== 'string') {
    throw new Error('Input must be a string!');
  }
}

function doSomething(input: string | number | null): void {
  assertString(input);

  // input's type is just 'string' here
}
```

up

flow

Typescript 3.7 - Null coalescing

```
// fallback for all falsy values
> undefined || 'default'
'default'
> null || 'default'
'default'
> 0 || 'default'
'default'
> '' || 'default'
'default'

// fallback for null or undefined only
> undefined ?? 'default'
'default'
> null ?? 'default'
'default'
> 0 ?? 'default'
0
> '' ?? 'default'
''
```

Typescript 3.7 - Optional chaining

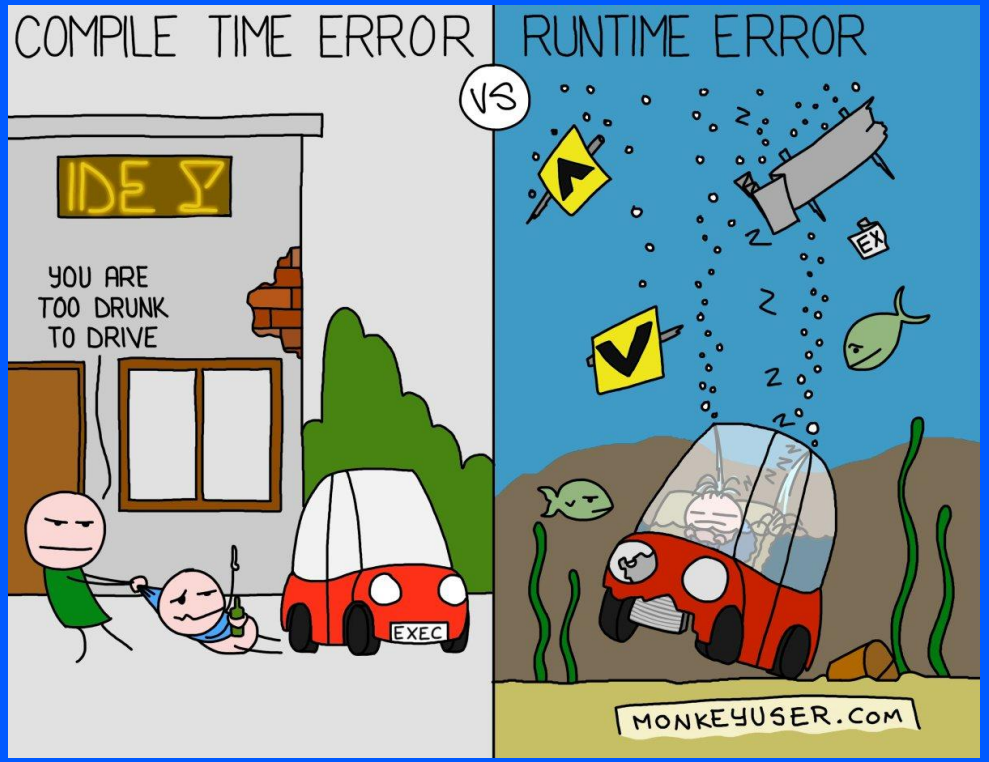
```
// without optional chaining
obj && obj.key1 && obj.key1.key2
obj && obj.key1 && obj.key1.key2 || 'default'
array && array[0] && array[0].key
obj.method && obj.method()

// with optional chaining
obj?.key1?.key2
obj?.key1?.key2 ?? 'default'
array?.[0]?.['key']
obj.method?.()
```

Summary

How to avoid common JS runtime errors

- ➔ set `strictNullChecks` flag in `tsconfig.json`
- ➔ avoid using `!` operator
  - add `no-non-null-assertion` rule to TSLint
- ➔ let Typescript infer non-nulls
  - use type predicates where necessary (e.g. use utility function)



Q&A

Matěj Chalk

Front-end developer @FlowUp

matejchalk

matejchalk