The 'never' Type in TypeScript

Embracing the Void: TypeScript’s Secret Weapon

What is the 'never' Type?

- TypeScript’s bottom type
- Represents the type of values that never occur
- No value is assignable to 'never' (except 'never' itself)
- The type of expressions that always throw or never return

```ts
export function throwError(message: string): never {
  throw new Error(message)
}

export function infiniteLoop(): never {
  while (true) {
    console.log('I\'m running forever!')
  }
}
```

When is 'never' Inferred?

Common Scenarios

- Functions that never return
- Unreachable code branches
- Exhaustive checks in switch statements
- Empty arrays or intersections of incompatible types

The Power of 'never'

- Catches logical errors
- Ensures all cases are handled
- Makes impossible states truly impossible
- Enhances type safety in complex scenarios

'never' in Action: Unreachable Code Detection

```ts
export function processResponse(response: 'yes' | 'no') {
  if (response === 'yes') {
    console.log('yes')
  }
  else if (response === 'no') {
    console.log('no')
  }
  else {
    const unreachable: never = response
    console.log('We should never get here!', unreachable)
  }
}
```

What's happening here?

- We've covered all possible values of `response`
- The `else` block should be unreachable
- TypeScript infers `response` as `never` in this block
- Assigning to `unreachable` variable catches the error

Exhaustive Checks with 'never'

```ts
interface Circle { kind: 'circle', radius: number }
interface Square { kind: 'square', size: number }
interface Triangle { kind: 'triangle', base: number, height

type Shape = Circle | Square

export function assertNever(x: never): never {
  throw new Error(`Unexpected object: ${x}`)
}

export function area(shape: Shape) {
  switch (shape.kind) {
    case 'circle': return Math.PI * shape.radius ** 2
    case 'square': return shape.size ** 2
      // case "triangle": return (shape.base * shape.height
    default:
      return assertNever(shape)
  }
}
```

The Power of Exhaustiveness

- Ensures all cases are handled
- Compiler error if a case is missed
- Refactor with confidence
- Makes your code more maintainable

'never' in Conditional Types

```ts
type ArrayElement<T> = T extends Array<(infer E)> ? E : never

export type StringArrayElement = ArrayElement<Array<string>>
export type NumberArrayElement = ArrayElement<Array<number>>
export type NeverElement = ArrayElement<string>
```

What's happening here?

- 'never' acts as a fallback in conditional types
- Helps create more precise mapped types
- Allows for powerful type transformations

Best Practices with 'never'

- Use it to mark impossible cases in your domain logic
- Leverage it for exhaustive checking in switch statements
- Employ it in advanced type manipulations and conditional types
- Be cautious of 'any' - it can be assigned to 'never', breaking its guarantees

Remember

- 'never' represents the logically impossible
- If you're handling a 'never', something's probably wrong!
- It's a powerful tool for creating safer, more robust code
- Use it wisely, and it will never let you down!

Questions?

Where 'never' is always an option, but rarely the answer!

Thank You!

Remember: In TypeScript, 'never' say never!