TYPESCRIPT

Hello

Kamil Richert

Senior Software Engineer at Atlassian

TYPESCRIPT

TypeScript rozszerza JavaScript o możliwość typowania. Dzięki czemu jesteśmy w stanie wyłapać więcej błędów zanim oprogramowanie trafi na produkcję.

Co nam daje TypeScript?

1. Ułatwia kontrolę nad aplikacją
2. Większa czytelność kodu
3. Wymusza mniejsze funkcje
4. Podpowiedzi w edytorze kodu
5. Sprawdza poprawność typów podczas kompilacji
6. Każdy kod JS jest prawidłowym kodem TS
7. TS finalnie jest kompilowany do JS

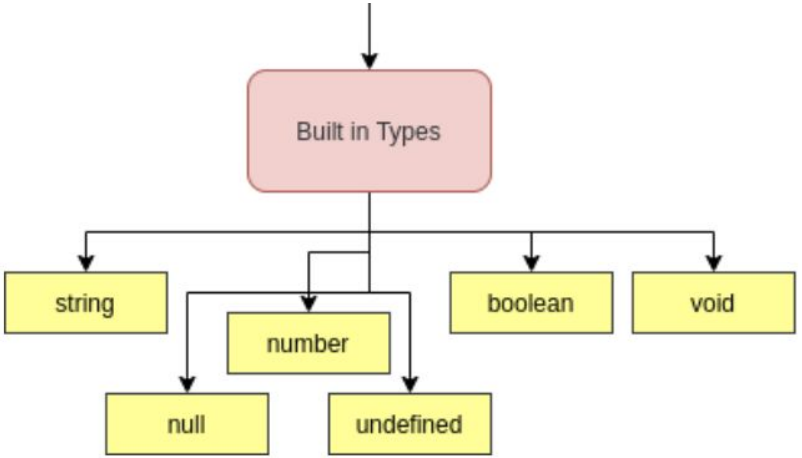
Jakie narzuca utrudnienia

1. Zwalnia czas wydawania oprogramowania
2. Dodatkowa konfiguracja przy starcie projektu

Podstawowe typy

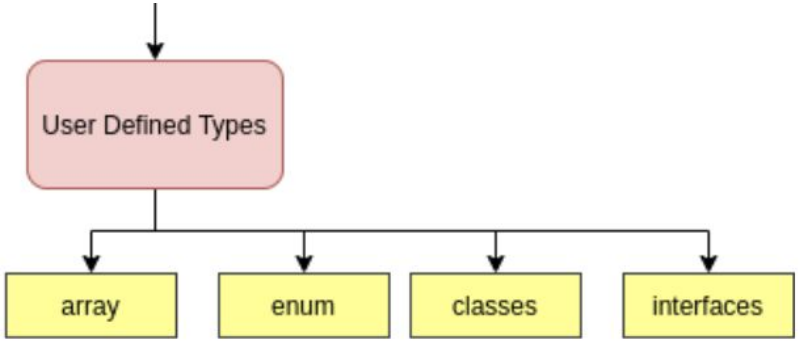
- string
- number
- boolean
- void
- null
- undefined

- never
- any
- unknown



Typy złożone

- array
- enum
- classes
- interfaces



Typowanie funkcji

Możemy typować zarówno deklaracje funkcji, jak i funkcje przypisywane do zmiennych. Przy funkcjach strzałkowych obowiązkowe są nawiasy

```
function catchPokemon(pokemonName: string, pokeball: Pokeball): boolean {
    // try to catch

    return true
}


const catchPokemon = (pokemonName: string, pokeball: Pokeball): boolean => {
    // try to catch
    return true
}
```

Jeśli funkcja nic nie zwraca, używamy typu *void*

Inferencja typów

Przy deklaracji wartości TS zapamiętuje typ, także nie musimy za każdy razem typować ręcznie

```
let trainerName = 'Ash'; // type name!
let maxPokemonCount = 6; // type number!
let gonnaCatchThemAll = true // type boolean;
```

```
let trainerName: string
Type 'boolean' is not assignable to type 'string'. (2322)
Peek Problem (Alt+F8)    No quick fixes available
trainerName = false;
maxPokemonCount = '3';
```

Słowo kluczowe as

```
interface Pokeball {
    name: string;
    chanceRate: number;
    rarity: PokeballRarity;
}

let ultraball: Pokeball = {};

let greatball: Pokeball = {} as Pokeball;
```

```
let ultraball: Pokeball

Type '{}' is missing the following properties from type
'Pokeball': name, chanceRate, rarity (2739)

Peek Problem (Alt+F8)    No quick fixes available
```

Słowo kluczowe **as** pozwoli nam powiedzieć TypeScriptowi, że dany obiekt na na pewno będize danego typu nawet jeśli teraz tak nie wygląda

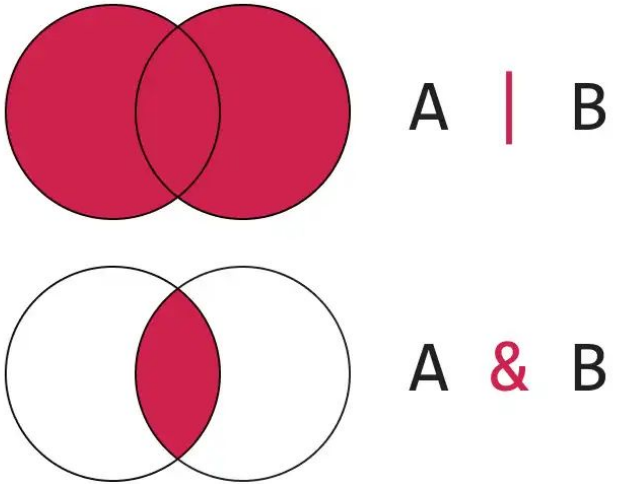
Union & Intersection

Union

type C = type A | type B

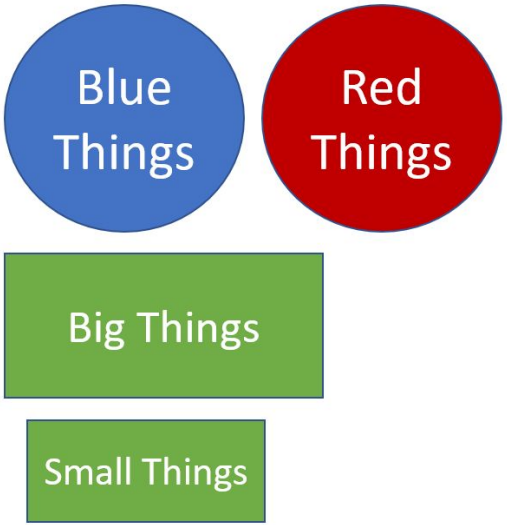
Intersection

type C = type A & type B

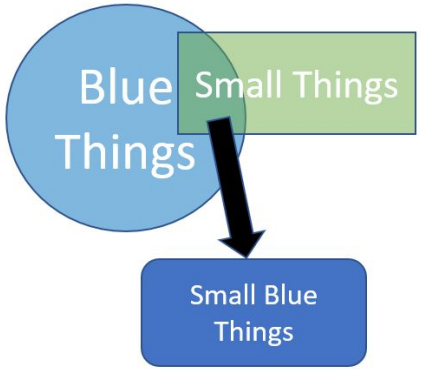


Union & Intersection

Consider classifying objects four ways: blue, red, big, and small

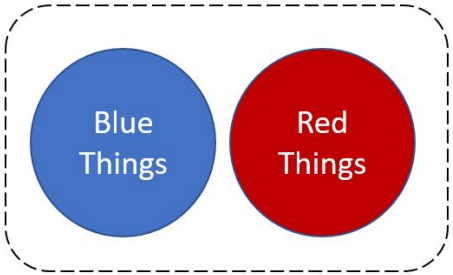


If we *intersect* **blue** with **small**, we get a new set:



The *intersection* of these sets has the *union* of its properties

If we *union* **blue** with **red**, we get a new set:



Blue or Red Things

The *union* of these sets has the *intersection* of its properties, which in this case is empty

Typy generyczne

Jeżeli chcemy stworzyć typ, który ma być reużywalny, ale zmienia się na przykład tylko 1 element to możemy wykorzystać generyczny typ.

```
identity<Type>(arg: Type): Type {
 return arg;
}
```

Klasy

- public
- private
- protected
- static
- readonly
- getter
- setter

```
class Pokemon() {
        constructor(name) {}
}
```

Public

```
class Pokemon {
    constructor(public name: string,
                public pokeType: PokemonType,
                public attack: PokemonAttack[]) {}
}

const pikachu = new Pokemon('Pikachu', 'lighting', []);


pikachu.
```

```
attack                    (property) Pokemon.at…
name
pokeType
```

Public

```
class Pokemon {
    constructor(public name: string,
                private pokeType: PokemonType,
                private attack: PokemonAttack[]) {}
}

const pikachu = new Pokemon('Pikachu', 'lighting', []);

pikachu.
```

```
name                                (property) Pokemon.na…
```

Protected

```
class Pokemon {
    constructor(public readonly name: string,
                private pokeType: PokemonType) {}
}

class LightingPokemon extends Pokemon {
    constructor(public readonly name: string) {
        super(name, 'lighting');
    }

    sayHello(): void {
        console.log(`I am ${this.pokeType} pokemon`);
    }
}

const pikachu = new LightingPokemon('Pikachu');

pikachu.
```

```
name                    (property) LightingPo…
sayHello
```

```
class Pokemon {
    constructor(public readonly name: string,
                protected pokeType: PokemonType) {}
}

class LightingPokemon extends Pokemon {
    constructor(public readonly name: string) {
        super(name, 'lighting');
    }

    sayHello(): void {
        console.log(`I am ${this.pokeType} pokemon`);
    }
}

const pikachu = new LightingPokemon('Pikachu');

pikachu.
```

```
name                    (property) Lig
sayHello
```

Static

```
class Pokemon {
    static currentWorld = 'Kanto';

    constructor(public name: string,
                private pokeType: PokemonType,
                private attack: PokemonAttack[]) {}
}

const pikachu = new Pokemon('Pikachu', 'lighting', []);

pikachu.currentWorld;

Pokemon.currentWorld;
```

Readonly

```
class Pokemon {
    static currentWorld = 'Kanto';

    constructor(public readonly name: string,
                private pokeType: PokemonType,
                private attack: PokemonAttack[]) {}
}

const pikachu = new Pokemon('Pikachu', 'lighting', []);

pikachu.name = 'Raichu';
```

Getter, setter

```
class Pokemon {
    constructor(private _name: string) {}

    get name(): string {
        return this.name.toUpperCase();
    }

    set name(value: string) {
        // sideeffects
        console.log('changing name');

        this._name = value;
    }
}

const pikachu = new Pokemon('pikachu');

pikachu.name;           // fires getter function
pikachu.name = 'ditto'; // fires setter function
```

Narzędzia typowania

Required
Partial
Pick
Omit
…

https://www.typescriptlang.org/docs/handbook/utility-types.html

```
type PartialCars = Partial<Cars>;

type RequiredUser = Required<User>;

type PickCars = Pick<Cars, 'model'>

type OmitCars = Omit<Cars, 'id'>
```

tsconfig

Po prawej strony przykładowy plik konfiguracyjny.

tsconfig.json to nic innego to ustawienia jak ma działać kompilator tsc, na co ma zwracać uwagę itp.

Reguły TS są bardzo przejrzyście opisane i udokumentowane

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": [
      "dom",
      "dom.iterable",
      "esnext"
    ],
    "allowJs": true,
    "skipLibCheck": true,
    "esModuleInterop": true,
    "allowSyntheticDefaultImports": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noFallthroughCasesInSwitch": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx"
  },
  "include": [
    "src"
  ]
}
```

Dzięki

Znajdziecie mnie:
https://www.linkedin.com/in/kamil-richert/
https://github.com/krichert