



TypeScript Základne typy

kurz Vývoj progresívnych
webových aplikácií

Eduard Kuric



TypeScript playground

- **Online transpilátor:**
- <https://www.typescriptlang.org/play>
- Kompilátor vs transpilátor
- **Transpilátor** – kompilátor, ktorý transformuje/konvertuje kód napísany v jednom jazyku do kódu v inom jazyku, ktorý má podobnú úroveň abstrakcie (alebo inej verzie rovnakého jazyka), transpilátory sú podmnožina kompilátorov,
- **Kompilátor** - prekladá kód z jedného jazyka do iného jazyka, spravidla na nižšej úrovni (človekom čitateľný kód na strojové inštrukcie).

Type Annotations : type

- Špecifikuje typ identifikátorov akými sú premenné, funkcie, objekty, ...
- Syntax : type

kde **type** je ľubovoľný platný typ

```
// premenne
```

```
let variableName: type;
```

```
let variableName: type = value;
```

```
// konstanta
```

```
const constantName: type = value;
```

Anotácia :number

- Keď je identifikátor anotovaný typom, možno ho použiť iba s daným typom

- Ak sa použije nesprávny typ, transpilátor vypíše chybu

- Použitá anotácia number:

```
let counter: number;
```

```
counter = 1;           // ok
```

```
let counter: number = 1;    // ok
```

```
counter = 'Hello';
```

```
// error, Type '"Hello"' is not  
assignable to type 'number'.
```

Ďalšie jednoduché typy

- `let name: string = 'John';`
- `let active: boolean = true;`
- `let age: number = 25.5;`

Pole, objekt

```
let names: string[] = [ 'John', 'Jane' ]
```

```
let person: {  
    name: string;  
    age: number  
};
```

```
person = {  
    name: 'John',  
    age: 25  
}; // ok
```

Argumenty funkcie, návratový typ

```
let greeting : (name: string) => string;
```

```
greeting = function (name: string)  
{  
    return `Hi ${name}`;  
};
```

```
greeting = function ()  
{  
    console.log('Hello');  
}; //error, nezhoduje sa navratovy typ
```

:number (floating point)

- `let price: number;`
- `price = 9.95;`
- `let x: number = 100,
 y: number = 200;`
- `let bin = 0b100;`
- `let anotherBin: number = 0B010;`
- `let hexadecimal: number = 0xA; //0XA
// $2^{53} - 1$, na konci literal n`
- `let big: bigint = 9007199254740991n;
// alebo Bigint constructor`
- `BigInt(9007199254740991)`

:string

- `let firstName: string = 'John';`
- `let title: string = "Web Developer";`
- `// tzv template string, backtick (`)`
- `let description = `This TypeScript string can span multiple lines`;`
- `// string interpolation`
- `let profile: string = `I'm ${firstName}. I'm a ${title}`;`

:boolean

- JavaScript ma typ `:Boolean`
(non-primitive boxed object)
 - dobrým zvykom je v TS vyhýbať sa Boolean typu
- TS má `boolean` typ, s malým **b**
- `let pending: boolean;`
- `pending = true;`

:object

- Reprezentuje hodnotu, ktorá nemá ani jeden primitívnych typov
- Primitívne typy v TS:
 - number
 - bigint
 - string
 - boolean
 - null
 - undefined
 - symbol

:object /2

- `let employee: object;`
- `employee = {
 firstName: 'John',
 lastName: 'Doe',
 age: 25,
 jobTitle: 'Web Developer'
};`
- `employee = "Jane"; // error`
`// error TS2322: Type '"Jane"' is not assignable to type 'object'.`

:object /3

- `console.log(employee.hireDate);`
`// error TS2339: Property 'hireDate'`
`does not exist on type 'object'.`
- Pozn.: v JavaScripte zafunguje, vráti `undefined`

:object /4 explicitné typy

```
let employee: {  
    firstName: string;  
    lastName: string;  
    age: number;  
    jobTitle: string;  
} = {  
    firstName: 'John',  
    lastName: 'Doe',  
    age: 25,  
    jobTitle: 'Web Developer'  
};
```

object vs Object

- `object` - reprezentuje neprimitívnu hodnotu
- `Object` — obsahuje ďalšie “veci”, ktoré sú prítomné vo všetkých JS objektoch, napr. `toString()`, `valueOf()`
 - Primitívna aj neprimitívna hodnota môže byť priradená `Object` typu

empty type

- {}
- Nemusí mať žiaden atribút, ale môže:

```
function acceptsEmpty(obj: {}): void {  
    console.log(obj);  
}
```

```
let notEmpty: {a: string} = {a: 'test'};
```

```
// ok  
acceptsEmpty(notEmpty);
```


array

- `let skills: string[];`
- `skills[0] = "Problem Solving";`
- `skills[1] = "Programming";`
- `skills.push('Software Design');`
- `let skills: string[];`
- `skills = ['Problem Solving', 'Software Design', 'Programming'];`
- `skills.push(100);`
`// error`
Argument of type 'number' is not assignable to parameter of type 'string'.

array /2

- `console.log(typeof(skills[0]));`
`//string`
- `console.log(series.length); // 3`
- `forEach()`, `map()`, `reduce()`, `filter()`
- `let series = [1, 2, 3];`
- `let doubleIt = series.map(e => e*2);`
- `console.log(doubleIt);`
`// [2, 4, 6]`

array, mixed type

- `let scores : (string | number)[];`
- `scores = ['Programming', 5,
 'Software Design', 4];`

tuple – ntica

- Počet elementov je fixný (spravidla)
- Typy elementov sú známe, nemusia byť rovnaké, poriadok je dôležitý
- `let skill: [string, number];`
- `skill = ['Programming', 5];`
- `skill = [5, 'Programming']; // error`
- `let color: [number, number, number] = [255, 0, 0];`

tuple – voliteľný element

- let bgColor, headerColor: [number, number, number, **number?**];
- bgColor = [0, 255, 255, 0.5];
- headerColor = [0, 255, 255];

enum – vymenovaný typ

- `enum Month { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };`
- `console.log(isItSummer(Month.Jun));`
- `console.log(isItSummer(6)); //ok`
- `// zmena zac. indexu`
- `enum Month { Jan = 1, Feb, ... };`

any

- Ak nepoznáme typ v čase písania programu
- Alebo hodnota neznámeho typu príde z API tretej strany
- Transpilátor nebude robiť kontrolu typu

```
let result: any;  
result = 10.123;  
console.log(result.toFixed()); // 10  
result.willExist();  
// ok, nevyhodi warning ani počas  
transpilácie, willExist() môže byť  
dostupná v runtime
```

void

- Návratový typ pre funkcie, ktoré nemajú vrátiť hodnotu

```
function log(message): void {  
    console.log(messsage);  
}
```

- Premenná:

```
let useless: void = undefined; // ok  
useless = 1 ; // error  
useless = null;  
// ok, ak --strictNullChecks nie je  
specifikovaný
```


never

- Typ, ktorý neobsahuje žiadne hodnoty
- Použitie napr. pri funkcii, ktorá vždy vyhadzuje chybu

```
function raiseError(message: string): never {  
    throw new Error(message);  
}
```

union

- `let result: number | string;`
- `result = 10; // OK`
- `result = 'Hi'; // OK`
- `result = false; // error`
- `function add(a: number | string, b: number | string)
 {}`

aliases

- Používame, ak chceme zdefinovať nový názov pre existujúci typ
- **`type alias = existingType;`**
- `type chars = string;`
- `let message: chars; // string type`
- `type alphanumeric = string | number;`
- `let input: alphanumeric;`

string literal

- Typ, ktorý akceptuje len určitý reťazec
- `let click: 'click';`
- `click = 'click'; // ok`
- `click = 'dblclick'; // error`
- `let mouseEvent: 'click' | 'dblclick' | 'mouseup' | 'mousedown';`
`// zvykne sa kombinovat s union a alias typmi`

Odvodenie typu

- Explicitná type anotácia

```
let counter: number;
```

- **Ak explicitne neurčíme typ, TypeScript robí implicitné odvodenie typov, “best common type” algoritmus**

- Implicitné odvodenie typu premennej:

```
let counter = 0; // number
```

- Implicitné odvodenie argumentu funkcie:

```
function setCounter(max=100) { ... }  
// number
```

Odvodenie typu /2

- Implicitné odvodenie návratovej hodnoty

```
function increment(counter: number)  
{ return counter++; } // number
```

```
let items = [1, 2, 3, null]; // number[]
```

```
let items = [0, 1, null, 'Hi'];  
// (number | string)[]
```

```
let arr = [new Date(), new RegExp('\d+')];  
// (RegExp | Date)[]
```

Kedy použiť explicitný typ?

- a kedy ponechať uhádnutie typu na TypeScript transpilátor (implicitné odvodenie)?
- V praxi, čo najviac používať implicitné odvodenie.
- Explicitný typ používame:
 - Keď deklarujeme premennú a neskôr jej priradíme hodnotu.
 - Keď chceme premennú, ktorú nemožno odvodiť.
 - Pri funkciách, čo najviac pridávať anotácie typu (návratové hodnoty, ak funkcia vracia `any` type)