TypeScript

Часть II

Старков Дима

TypeScript?

Спасет от выстрелов себе в ногу

ESNext прямо сейчас

Средство против TypeError

Пишет код за вас

Документация к коду

Но...

Много дополнительного кода?

Нас спасет вывод типов!

TypeScript крут.
Но можем ли мы описать весь JavaScript?

Вспомним **Type**Script 1.0

Интерфейсы

Классы

Обобщенные типы

Перегрузки функций

Чего еще желать?

```
// String.split
split(separator: ?, limit: number): string[]
```

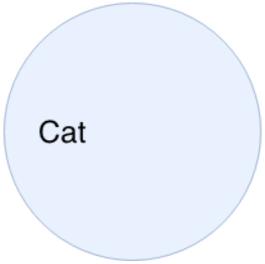
```
// String.split
split(separator: string | RegExp, limit: number): string[]
```

```
// String.split
split(separator: string | RegExp, limit: number): string[]
```

Решение: Union Types

Intersection Types

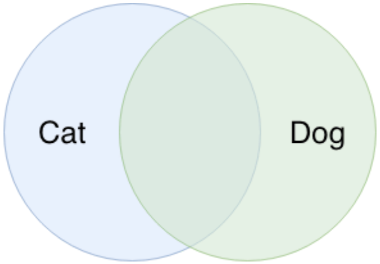
```
type Cat = {
    purr()
}
```



Intersection Types

```
type Cat = {
    purr()
}

type Dog = {
    woof()
}
```

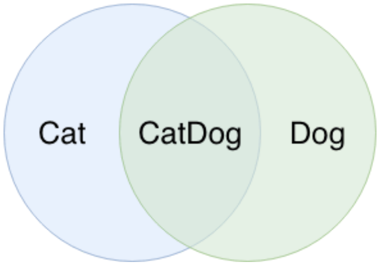


Intersection Types

```
type Cat = {
    purr()
}

type Dog = {
    woof()
}

type CatDog = Cat & Dog
```



Type Alias

```
// String.split
split(separator: string | RegExp, limit: number): string[]
```

Type Alias

```
type StringOrRegExp = string | RegExp

// String.split
split(separator: StringOrRegExp, limit: number): string[]
```

Type vs Interface

```
type Point = {
    x: number
    y: number
}
```

```
interface Point {
    x: number
    y: number
}
```

implements **interface**

Type1 | Type2 – не интерфейс!

Тип ≡ Множество

Можем объединять типы |

Можем пересекать типы &

Можем вычитать из одного типа другой

Фух, теперь точно всё...

А вот и нет!

```
function get(obj, keyName) {

    return obj[keyName]
}
```

А вот и нет!

```
function get(obj: any, keyName: string): any {
    return obj[keyName]
}

// TypeError: Cannot read property 'prototype' of null
get(null, 'prototype')
```

Что делать?

А вот и нет!

```
function get(obj: any, keyName: string): any {
    return obj[keyName]
}

// TypeError: Cannot read property 'prototype' of null
get(null, 'prototype')
```

Может быть любым – **Generics?**

А вот и нет!

```
function get(obj: any, keyName: string): any {
    return obj[keyName]
}

// TypeError: Cannot read property 'prototype' of null
get(null, 'prototype')
```

`keyName ∈ Object.keys(obj)` –?

А вот и нет!

```
function get(obj: any, keyName: string): any {
    return obj[keyName]
}

// TypeError: Cannot read property 'prototype' of null
get(null, 'prototype')
```

`obj[keyName]` – ?

Хотим знать список полей объекта и типы значений на этапе компиляции

Решение: Lookup Types и keyof

Lookup типы

```
interface IUser {
    login: string
    age: number
    gender: 'male' | 'female'
}

let login: IUser['login']
let login: string

let loginOrAge: IUser['login' | 'age']
let loginOrAge: string | number
```

keyof

```
interface IUser {
    login: string
    age: number
    gender: 'male' | 'female'
}

let key: keyof IUser
let key: 'login' | 'age' | 'gender'
```

Наша простая функция

```
function get(obj, keyName) {
    return obj[keyName]
}
```

Наша ~~простая~~ функция

```
function get<T>(obj: T, keyName: keyof T): T[keyof T] {
    return obj[keyName]
}
```

Наша ~~простая~~ функция

```
function get<T>(obj: T, keyName: keyof T): T[keyof T] {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')
```

Наша ~~простая~~ функция

```
function get<{ a: 1 }>(obj: T, keyName: keyof T): T[keyof T] {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')
```

Наша ~~простая~~ функция

```
function get<{ a: 1 }>(obj: T, keyName: 'a'): T['a'] {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')
```

Наша ~~простая~~ функция

```
function get<{ a: 1 }>(obj: T, keyName: 'a'): number {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')
```

Наша ~~простая~~ функция

```
function get<T>(obj: T, keyName: keyof T): T[keyof T] {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')

// Argument of type '"c"'
// is not assignable to parameter of type '"a" | "b"'.
let c: undefined = get({ a: 1, b: 2 }, 'c')
```

Наша ~~простая~~ функция

```
function get<T, K extends keyof T>(obj: T, keyName: K): T[K] {
    return obj[keyName]
}

let a: number = get({ a: 1 }, 'a')



let c: undefined = get({ a: 1, b: 2 }, 'c')
```

А что там в es5?

```
interface IUser {
    login: string
    age: number
    gender: 'male' | 'female'
}

const user = { login: 'dimastark', age: 22, gender: 'male' }
const readonlyUser: ? = Object.freeze(user)
```

А что там в es5?

```
interface IFrozenUser {
    readonly login: string
    readonly age: number
    readonly gender: 'male' | 'female'
}

const user = { login: 'dimastark', age: 22, gender: 'male' }
const readonlyUser: IFrozenUser = Object.freeze(user)
```

Решение: Mapped Types

Mapped Types

```
interface IUser {
    login: string
    age: number
    gender: 'male' | 'female'
}

type Readonly<T> = {
    readonly [P in 'login' | 'age' | 'gender']: T[P];
};

const user = { login: 'dimastark', age: 22, gender: 'male' }
const readonlyUser: Readonly<IUser> = Object.freeze(user)
```

Mapped Types + keyof

```
interface IUser {
    login: string
    age: number
    gender: 'male' | 'female'
}

type Readonly<T> = {
    readonly [P in keyof T]: T[P];
};

const user = { login: 'dimastark', age: 22, gender: 'male' }
const readonlyUser: Readonly<IUser> = Object.freeze(user)
```

infer

```
type ValueOf<T> = T extends {
    [key: string]: infer U
} ? U : never;

ValueOf<{ a: string, b: string }>  // string
ValueOf<{ a: string, b: number }>  // string | number
```

infer

```
type ReturnType<T> = T extends (
    (...args: any) => infer R
) ? R : never;

ReturnType<(a: number) => string>  // string
ReturnType<(s: string) => number>  // number
ReturnType<{ a: number }>          // never
```

Mapped Types

```
interface IUser {
    login: string
    birthDate: {
        year: number
        month: number
        day: number
    }
    gender: 'male' | 'female'
}
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ readonly s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ readonly s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ readonly s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ readonly s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: DeepReadonly<{ readonly s: string }>[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { readonly s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

DeepReadonly<{
    readonly array: { readonly s: string }[]
}>
```

Mapped Types

```
type DeepReadonly<T> = {
    readonly [P in keyof T]:
        T[P] extends (infer U)[] ? DeepReadonly<U>[] :
        T[P] extends object ? DeepReadonly<T[P]> :
        T[P];
};

{
    readonly array: { readonly s: string }[]
}
```

Перерыв

Utility Types

Pick – выбираем поля объекта

```
Pick<{
    a: string;
    b: number;
    c: boolean;
}, 'a' | 'b' | 'd'>
```

```
{
    a: string;

    c: boolean;
}
```

Utility Types

Omit – исключаем поля объекта

```
Omit<{                                {
    a: string;
    b: number;
    c: boolean;                           c: boolean;
}, 'a' | 'b' | 'd'>                   }
```

Utility Types

Partial – делаем все поля опциональными

```
Partial<{
    a: string | undefined;
    b: number;
    c?: boolean;
}>
```

```
{
    a?: string | undefined;
    b?: number;
    c?: boolean;
}
```

Utility Types

Required – делаем все поля обязательными

```
Required<{
    a: string;
    b?: number | undefined;
    c?: boolean;
}>
```

```
{
    a: string;
    b: number | undefined;
    c: boolean;
}
```

Utility Types

Readonly – делаем все поля readonly

```
Readonly<{
    a: string;
    b: number;
    c?: boolean;
}>
```

```
{
    readonly a: string;
    readonly b: number;
    readonly c?: boolean;
}
```

Utility Types

ReturnType – тип возвращаемого значения

```
ReturnType<() => number>                       number
ReturnType<() => string>                       string
ReturnType<() => boolean>                      boolean
```

Utility Types

Parameters – тип аргументов функции

```
Parameters<(s: string | RegExp) => void>     [string | RegExp]
Parameters<(a: number, b: number) => void>   [number, number]
Parameters<(...nums: number[]) => void>      number[]
Parameters<(...args: any[]) => void>         any[]
```

Type Guards

Условие, дающее гарантию о безопасном сужении типа

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);
    } else if (typeof n === 'number') {
        return -n;
    } else {
        return !n;
    }
}
```

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);
    } else if (typeof n === 'number') {
        return -n;
    } else {
        return !n;
    }
}
```

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);  // n is string
    } else if (typeof n === 'number') {
        return -n;
    } else {
        return !n;
    }
}
```

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);
    } else if (typeof n === 'number') {
        return -n;  // n is number
    } else {
        return !n;
    }
}
```

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);
    } else if (typeof n === 'number') {
        return -n;
    } else {
        return !n;  // n is boolean
    }
}
```

Union Type Guard

```
function negate(n: string | number | boolean) {
    if (typeof n === 'string') {
        return '-'.concat(n);
    } else if (typeof n === 'number') {
        return -n;
    }

    return !n;  // n is boolean
}
```

instanceof Type Guard

```
function addShape(shapes: Shape[], obj: object) {
    if (obj instanceof Shape) {
        shapes.push(obj)
    }

    throw new TypeError('Argument is not instanceof Shape')
}
```

instanceof Type Guard

```
function addShape(shapes: Shape[], obj: object) {
    if (obj instanceof Shape) {
        shapes.push(obj)
    }

    throw new TypeError('Argument is not instanceof Shape')
}
```

instanceof Type Guard

```
function addShape(shapes: Shape[], obj: object) {
    if (obj instanceof Shape) {
        shapes.push(obj)
    }

    throw new TypeError('Argument is not instanceof Shape')
}
```

in Type Guard

```
function checkProp(obj: object, name: string): boolean {
    if (name in obj) {

        return true;
    }

    throw new TypeError(`"${name}" property is missing!`)
}
```

in Type Guard

```
function checkProp(obj: object, name: string): boolean {
    if (name in obj) {

        return true;
    }

    throw new TypeError(`"${name}" property is missing!`)
}
```

in Type Guard

```
function checkProp(obj: object, name: string): boolean {
    if (name in obj) {
        console.log(obj[name]);
        return true;
    }

    throw new TypeError(`"${name}" property is missing!`)
}
```

User Defined Type Guard

User Defined Type Guard

```typescript
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return 'r' in o
        ? Math.PI * o.r * o.r
        : o.a * o.a;
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return 'r' in o
        ? Math.PI * o.r * o.r
        : o.a * o.a;
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return 'a' in o
        ? o.a * o.a
        : Math.PI * o.r * o.r;
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return 'a' in o
        ? o.a * o.a
        : Math.PI * o.r * o.r;
}

function isCircle(o: any): o is Circle {
    return 'r' in o && typeof o.r === 'number';
}

function isSquare(o: any): o is Square {
    return 'a' in o && typeof o.a === 'number';
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return 'a' in o
        ? o.a * o.a
        : Math.PI * o.r * o.r;
}

function isCircle(o: any): o is Circle {
    return 'r' in o && typeof o.r === 'number';
}

function isSquare(o: any): o is Square {
    return 'a' in o && typeof o.a === 'number';
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return isSquare(o)
        ? o.a * o.a
        : Math.PI * o.r * o.r;
}

function isCircle(o: any): o is Circle {
    return 'r' in o && typeof o.r === 'number';
}

function isSquare(o: any): o is Square {
    return 'a' in o && typeof o.a === 'number';
}
```

User Defined Type Guard

```
type Circle = { r: number };
type Square = { a: number };


function area(o: Circle | Square): number {
    return isCircle(o)
        ? Math.PI * o.r * o.r
        : o.a * o.a;
}

function isCircle(o: any): o is Circle {
    return 'r' in o && typeof o.r === 'number';
}

function isSquare(o: any): o is Square {
    return 'a' in o && typeof o.a === 'number';
}
```

ES Next 🚆

Декораторы классов

```
type Class = {
    new(...args: any[]): any
};


function decorator<T extends Class>(C: T) {
    return class extends C {
        oldProperty = 'override';
        newProperty = 'new property';
    }
}

@decorator
class SomeClass {
    oldProperty = 'old property';
}
```

Декораторы классов

```typescript
type Class = {
    new(...args: any[]): any
};


function decorator<T extends Class>(C: T) {
    return class extends C {
        oldProperty = 'override';
        newProperty = 'new property';
    }
}

@decorator
class SomeClass {
    oldProperty = 'old property';
}
```

Декораторы классов

```
type Class = {
    new(...args: any[]): any
};


function decorator<T extends Class>(C: T) {
    return class extends C {
        oldProperty = 'override';
        newProperty = 'new property';
    }
}

@decorator
class SomeClass {
    oldProperty = 'old property';
}
```

Декораторы классов

```
type Class = {
    new(...args: any[]): any
};


function decorator<T extends Class>(C: T) {
    return class extends C {
        oldProperty = 'override';
        newProperty = 'new property';
    }
}

@decorator
class SomeClass {
    oldProperty = 'old property';
}
```

Декораторы классов

```
type Class = {
    new(...args: any[]): any
};


function decorator<T extends Class>(C: T) {
    return class extends C {
        oldProperty = 'override';
        newProperty = 'new property';
    }
}

@decorator
class SomeClass {
    oldProperty = 'old property';
}
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    get prop() {
        return 'prop!';
    }
}

Object.keys(new SomeClass());  // ['prop']
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    @hidden get prop() {
        return 'prop!';
    }
}
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    @hidden get prop() {
        return 'prop!';
    }
}
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    @hidden get prop() {
        return 'prop!';
    }
}
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    @hidden get prop() {
        return 'prop!';
    }
}
```

Декораторы методов и свойств класса

```
function hidden(
    target: any,
    key: string,

    descriptor: PropertyDescriptor
) {
    descriptor.enumerable = false;
}

class SomeClass {
    @hidden get prop() {
        return 'prop!';
    }
}

Object.keys(new SomeClass());  // []
```

Декораторы методов и свойств класса

```
function greet(target: any, key: string, descriptor: PropertyDescriptor) {
    const method = descriptor.value;


    descriptor.value = function () {
        return 'Hello from ' + method.apply(this);
    };
}

class SomeClass {
    private prop = 'prop!';

    @greet getProp() {
        return this.prop;
    }
}
```

Декораторы методов и свойств класса

```
function greet(target: any, key: string, descriptor: PropertyDescriptor) {
    const method = descriptor.value;


    descriptor.value = function () {
        return 'Hello from ' + method.apply(this);
    };
}

class SomeClass {
    private prop = 'prop!';

    @greet getProp() {
        return this.prop;
    }
}
```

Декораторы методов и свойств класса

```
function greet(target: any, key: string, descriptor: PropertyDescriptor) {
    const method = descriptor.value;


    descriptor.value = function () {
        return 'Hello from ' + method.apply(this);
    };
}

class SomeClass {
    private prop = 'prop!';

    getProp() {
        return this.prop;
    }
}
```

Декораторы методов и свойств класса

```
function greet(target: any, key: string, descriptor: PropertyDescriptor) {
    const method = descriptor.value;


    descriptor.value = function () {
        return 'Hello from ' + method.apply(this);
    };
}

class SomeClass {
    private prop = 'prop!';

    getProp() {
        return this.prop;
    }
}
```

Декораторы методов и свойств класса

```
function greet(target: any, key: string, descriptor: PropertyDescriptor) {
    const method = descriptor.value;


    descriptor.value = function () {
        return 'Hello from ' + method.apply(this);
    };
}

class SomeClass {
    private prop = 'prop!';

    getProp() {
        return this.prop;
    }
}
```

Optional Chaining

```
let s;

s = (   // 👎
    object
    && object.nested
    && object.nested.array
    && object.nested.array[0]
    && object.nested.array[0].toString
    && object.nested.array[0].toString()
);

s = object?.nested?.array?.[0]?.toString?.();  // 👍
```

Nullish Coalescing

```javascript
let value = options.value || 'default';

// 👎
value = 0 || 'default';
value = '' || 'default';
value = false || 'default';

// 👍
value = 0 ?? 'default';
value = '' ?? 'default';
value = false ?? 'default';
```

Ссылочки

TypeScript Handbook

TypeScript Deep Dive

TypeScript Playground

TypeScript – Тьюринг полная система типов

Вопросы?

Спасибо!