Яндекс

TypeScript (часть 2)

Татьяна Игнатенко, разработчик интерфейсов

00

Дополнение
к предыдущей лекции

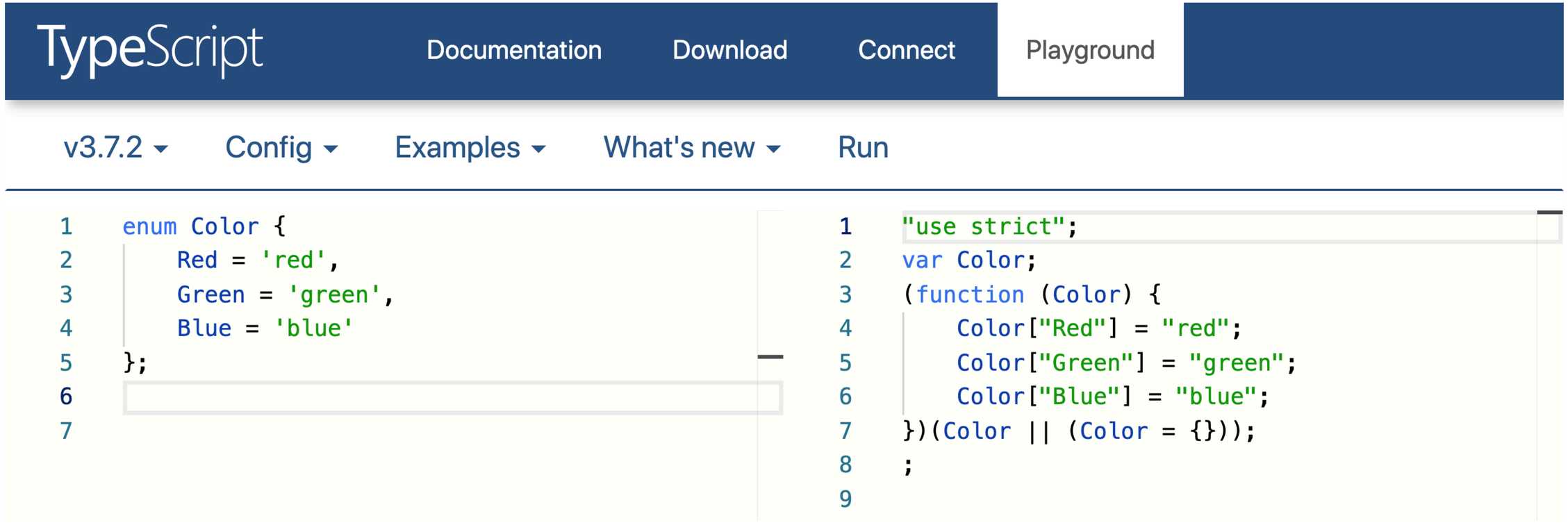
Enum-ы существуют во время RunTime

- Числовые enum-ы имеют двойной mapping

[Playground]

Enum-ы существуют во время RunTime

- Строковые enum-ы имеют односторонний mapping



[Playground]

Enum-ы существуют во время RunTime

> Смешанные enum-ы имеют двусторонний и односторонний mapping

[Playground]

Enum-ы

- Строковые типы 👍

01

Совместимость типов

- Виды типизации
- Проверка совместимости объектов
- Особенности совместимости классов
- Совместимость функций

Виды типизации

- **Номинальная**
  совместимость должна быть явно указана (наследована) при определении типа

  - Java, C++ и другие

- **Структурная (🐤)**
  совместимость определяется структурой самого типа

  - TypeScript и другие

Проверка совместимости объектов

```
class Person1 { name: string; age: number; }

class Person2 = { name: string; age: number; }

class Named { name: string }

let person1: Person1;
let person2: Person2;
let named: Named;

person1 = new Person1(); // OK
person2 = new Person2(); // OK

named = new Person1(); // OK
named = new Person2(); // OK

person1 = new Named(); // ERROR
person2 = new Named(); // ERROR
```

Особенности совместимости классов

> **Не учитывают статические члены и конструктор**

```
class Calculator {
  calc(expression) {}
}

class CashedCalculator {
  static cashe:  = {};
  static MAX_CASHED_ITEMS_COUNT: number = 1000;

  calc(expression) {}
}

let simpleCalculator: Calculator;
let cashedCalculator: CashedCalculator;

simpleCalculator = new CashedCalculator(); // OK
cashedCalculator = new Calculator(); // OK
```

Особенности совместимости классов

> **Private/protected поля должны быть из одного и того же класса**

В итоге экземпляру класса **с private/protected полями** можно присвоить только экземпляр **этого класса или класса-наследника**

Совместимость функций

Функция Y совместима с X (возможно Y = X), если

1. Если в Y присутствуют все параметры X с совместимым типом
2. Возвращающее значение Y совместимо с возвращающим значением X

Совместимость функций

Параметры

```
let x = (a: number) => 0;
let y = (b: number, s: string) => 0;

y = x; // OK
x = y; // Error
```

Возвращающее значение

```
let x = () => ({name: "Alice"});
let y = () => ({name: "Alice", location: "Seattle"});

x = y; // OK
y = x; // Error
```

02

Вывод типов

Вывод типов

- Инициализация переменных
- Установка значений по умолчанию для параметров функции
- Определение типа возвращаемого значения функцией

```
let foo = 123;
let bar = “Hello";

const add = (a = 5, b = 10) => a + b;
```

Вывод типов

- Присваивание функции в переменную с указанным типом

```
window.onmousedown = function(mouseEvent) {
  console.log(mouseEvent.button);   // OK
  console.log(mouseEvent.kangaroo); // Error
};
```

Более сложные случаи

Поиск наиболее общего типа

```
let x = [0, 1, null]; // (number | null)[]

let zoo1 = [new Animal(), new Elephant(), new Snake()]; // Animal[]

let zoo2 = [new Elephant(), new Snake()]; // (Elephant | Snake)[]

zoo1.push(new Tiger()); //OK

zoo2.push(new Tiger()); // ERROR
```

03

Intersection and Union

- Intersection and Union
- Проблемы Union-ов
- Решения

Intersection

> **Cat & Dog - тип, который включает все свойства типов Cat и Dog**

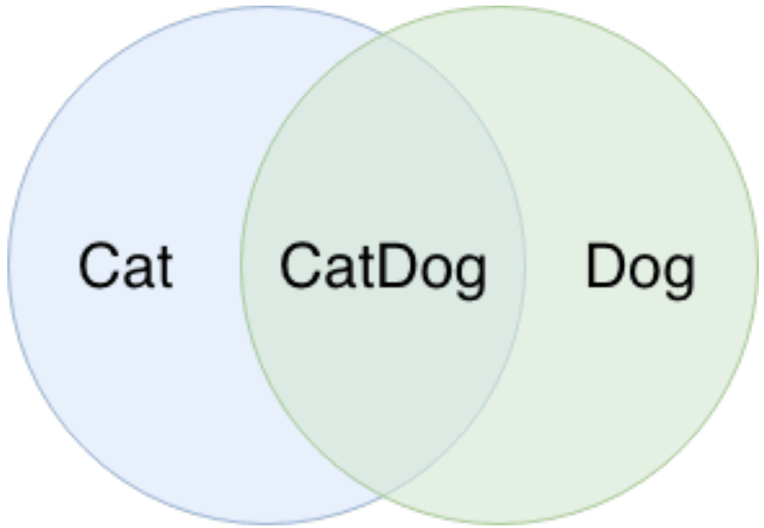
```ts
type Cat = {
  name: string;
  purr(): void;
}

type Dog = {
  name: string;
  woof(): void;
}

type CatDog = Cat & Dog;

let o: CatDog = // …;

o.purr(); // OK
o.woof(); // OK
```



Union

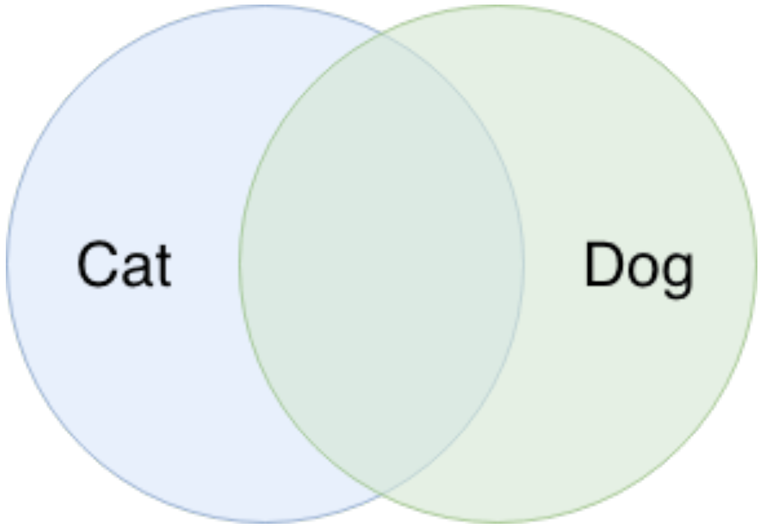
> **Cat | Dog - описывает значение, которое может быть одним из нескольких типов**

```ts
type CatOrDog = Cat | Dog;

let o: CatOrDog = // …;

o.purr(); // ERROR
o.woof(); // ERROR

o.name; // OK
```



Проблема Union-ов

```
function getSmallPet(): Fish | Bird {
  // ...
}

let pet = getSmallPet();

pet.eat(); // OK
pet.swim();   // ERROR
```

Проблема Union-ов

```
let pet = getSmallPet();

if (pet.swim) { // Error
  pet.swim(); // Error
} else if (pet.fly) { // Error
  pet.fly(); // Error
}
```

Решение 1: type assertion

```
let pet = getSmallPet();

if ((pet as Fish).swim) { // OK
  (pet as Fish).swim(); // OK
} else if ((pet as Bird).fly) { // OK
  (pet as Bird).fly(); // OK
}
```

Решение 2: type guards (typeof)

```
function func(smt: string | number) {
  if (typeof smt === ‘number') {
    return smt.toFixed(2); // OK
  }

  if (typeof smt === ‘string') {
    return smt.split(‘’); // OK
  }

  throw new Error(`Expected string or number, got '${smt}'.`);
}
```

Решение 2: type guards (instanceof)

```
if (pet instanceof Fish) {
  pet.swim(); // OK
} else if (pet instanceof Bird) {
  pet.fly(); // OK
}
```

Решение 3: type guards
(функция, возвращающая type predicate)

```
function isFish(pet: Fish | Bird): pet is Fish {
  return (pet as Fish).swim !== undefined;
}

function move(pet: Fish | Bird) {
  if (isFish(pet)) {
    return pet.swim(); // OK
  }
  return pet.fly(); // OK
}
```

Решение 4: type guards (“in” оператор)

```
function move(pet: Fish | Bird) {
  if ("swim" in pet) {
    return pet.swim(); // OK
  }
  return pet.fly(); // OK
}
```

03

Type aliases

- Type aliases
- Type aliases vs Interfaces

Type aliases

> **Похоже на Interfaces, но могут описывать примитивы, unions intersections и tuples**

```ts
type Name = string;
type Padding = number | string;
type FIO = [string, string, string];

type Animal = { name: string };
type Cat = Animal & { purrs: true };
type CatDog = Cat & Dog;

type NameResolver = () => string;
```

Type aliases vs Interfaces

- Type для Unions и Intersections
- Interface с implements и extends
- В других случаях, что больше нравится

TypeScript: Deep Dive [Style Guide]

03

Literal types

- String Literal types
- Numeric Literal types
- Enum Literal types

Literal types

- Помогают ограничить область значений типа
- => Помогают ловить больше багов
- => Лучше документируют код

String literal types

- Описывают строку конкретными значениями

```
type Easing = "ease-in" | "ease-out" | "ease-in-out";
"img"; // тоже тип
```

- Позволяют отличать overloads по значению строки

```
function createElement(tagName: "img"): HTMLImageElement;
function createElement(tagName: "input"): HTMLInputElement;
// ... more overloads ...
function createElement(tagName: string): Element {
  // ... code goes here ...
}
```

Numeric Literal Types

```ts
function rollDice(): 1 | 2 | 3 | 4 | 5 | 6 {
  // ...
}

function foo(x: number) {
  if (x !== 1 || x !== 2) {
    //          ~~~~~~~
    // Operator '!==' cannot be applied to types '1' and '2'.
  }
}
```

Enum members

> **Enum-members тоже могут выступают в роли отдельных типов**

```
enum ShapeKind { Circle,  Square}

interface Circle { kind: ShapeKind.Circle;  radius: number; }

interface Square { kind: ShapeKind.Square;  sideLength: number; }

let c: Circle = {
  kind: ShapeKind.Square, // Error
  radius: 100,
}
```

03

Сложные типы

- Generics
- Lookup types и keyof
- Mapped types

Generics

> **Функции, интерфейсы, type aliases, классы можно параметризовать**

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

class GenericNumber<T> {
  zeroValue: T;
  add: (x: T, y: T) => T;
}
```

Generics

> **На параметризацию можно накладывать ограничения**

```ts
function read<B extends Book>(book: B): void {
  // ...
}

function getProperty<T, K extends keyof T>(obj: T, key: K) {
  return obj[key];
}
```

Lookup types

```
type t = Cat[“name”] // string
```

keyof

```
type TodoKeys = keyof Todo; // "id" | "text" | “priority"
```

Lookup types and keyof

```
function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {
  return obj[key];
}

let name: string = getProperty(taxi, 'manufacturer');
let year: number = getProperty(taxi, 'year');

let unknown = getProperty(taxi, ‘unknown’); // ERROR
```

Распространенные задачи

```ts
interface Person { name: string; age: number; }


interface PersonPartial {
  name?: string;
  age?: number;
}

interface PersonReadonly {
  readonly name: string;
  readonly age: number;
}
```

Mapped types

> **Преобразует каждое свойство старого типа схожим образом**

```
type Readonly<T> = { readonly [P in keyof T]: T[P]; }
type Partial<T> = { [P in keyof T]?: T[P]; }
type Nullable<T> = { [P in keyof T]: T[P] | null }

type PersonPartial = Partial<Person>;
type ReadonlyPerson = Readonly<Person>;
type PersonNullable = Nullable<Person>;
```

Яндекс

Спасибо

Татьяна Игнатенко

Разработчик интерфейсов

tanigna@yandex-team.ru

@tanigna