

# Лекция 12

# TypeScript

Дмитрий Зайцев  
Мартин Комитски



образование

# План на сегодня

- Введение
- Настройка окружения, Hello World
- Основные принципы и возможности TS
- Практика

# Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях



# Введение

# Typescript. Введение

Typescript is:

- A JavaScript that scales.
- A typed superset of JavaScript that compiles to plain JavaScript.

<https://www.typescriptlang.org>



# Typescript. Зачем он нужен? Что дает?

- Дает возможность привести в порядок код больших приложений
- Привносит новые концепции классических ЯП
- Дополнительные возможности в IDE
- Дополнительные возможности из ECMAScript

# Typescript. Требования

- node
- npm
- tsc
- Продвинутая IDE или текстовый редактор с расширенной поддержкой TS, например, VS Code

# Типы данных



# Typescript. Типы данных

```
1.  // Boolean
2.  let isDone: boolean = false;
3.
4.  // Number
5.  let decimal: number = 6;
6.  let hex: number = 0xf00d;
7.  let binary: number = 0b1010;
8.  let octal: number = 0o744;
9.
10. // String
11. let color: string = "blue";
12. color = 'red';
13.
14. // Array
15. let list: number[] = [1, 2, 3];
16. let list: Array<number> = [1, 2, 3];
17.
18. // Tuple
19. let x: [string, number];
20. x = ["hello", 10]; // OK. Как получить error?
```

# Typescript. Типы данных

```
1.  // Enum
2.  enum Color {Red, Green, Blue}
3.  let c: Color = Color.Green;
4.
5.  // Any
6.  let notSure: any = 4;
7.  notSure = "maybe a string instead";
8.  notSure = false; // okay, definitely a boolean
9.
10. // Unknown aka type-safe any
11. let notSure: unknown = 5;
12. let sure: number = notSure; // Error
13.
14. // Void
15. function warnUser(): void {
16.     console.log("This is my warning message");
17. }
18.
19. // Null, Undefined
20. // Not much else we can assign to these variables!
21. let u: undefined = undefined;
22. let n: null = null;
```

# Typescript. Типы данных

```
1.  // Never
2.  // Function returning never must have unreachable end point
3.  function error(message: string): never {
4.      throw new Error(message);
5.  }
6.
7.  // Inferred return type is never
8.  function fail() {
9.      return error("Something failed");
10. }
11.
12. // Function returning never must have unreachable end point
13. function infiniteLoop(): never {
14.     while (true) {
15.     }
16. }
```

# Typescript. Типы данных

```
1. // Object
2. declare function create(o: object | null): void;
3.
4. create({ prop: 0 }); // OK
5. create(null); // OK
6.
7. create(42); // Error
8. create("string"); // Error
9. create(false); // Error
10. create(undefined); // Error
```

# Typescript. Типы данных. Type assertions

```
1. // Type assertions
2. let someValue: any = "this is a string";
3.
4. let strLength: number = (<string>someValue).length;
5.
6. let someValue: any = "this is a string";
7.
8. let strLength: number = (someValue as string).length;
```

# Typescript. Типы данных.

## Пересечение и объединение типов

1. `type TRandomIntersection = number & string;` (Плохой пример, вернёмся на интерфейсах)
2. `type TRandomUnion = number | string;`

# Type Inference

# TypeScript. Типы данных. Type Inference

1. `// Type Inference`
2. `let x = 3; // typeof x === 'number'`



# Функции

# Typescript. Функции

```
1. // Typing the function
2. function add(x: number, y: number): number {
3.     return x + y;
4. }
5.
6. let myAdd = function(x: number, y: number): number { return x + y; };
7.
8. // Writing the function type
9. let myAdd: (x: number, y: number) => number =
10.     function(x: number, y: number): number { return x + y; };
11.
12. let myAdd: (baseValue: number, increment: number) => number =
13.     function(x: number, y: number): number { return x + y; };
```

# Typescript. Функции

```
1. // Inferring the types
2. // myAdd has the full function type
3. let myAdd = function(x: number, y: number): number { return x + y; };
4.
5.
6. // The parameters 'x' and 'y' have the type number
7. let myAdd: (baseValue: number, increment: number) => number =
8.     function(x, y) { return x + y; };
```

# Typescript. Функции

```
1. // Optional Parameters
2. function buildName(firstName: string, lastName: string) {
3.     return firstName + " " + lastName;
4. }
5.
6. let result1 = buildName("Bob");           // error, too few parameters
7. let result2 = buildName("Bob", "Adams", "Sr."); // error, too many parameters
8. let result3 = buildName("Bob", "Adams");    // ah, just right
```

# Typescript. Функции

```
1. // Optional Parameters
2. function buildName(firstName: string, lastName?: string) {
3.     if (lastName)
4.         return firstName + " " + lastName;
5.     else
6.         return firstName;
7. }
8.
9. let result1 = buildName("Bob");           // works correctly now
10. let result2 = buildName("Bob", "Adams", "Sr."); // error, too many parameters
11. let result3 = buildName("Bob", "Adams");    // ah, just right
```

# Typescript. Функции

```
1. // Default Parameters
2. function buildName(firstName = "Will", lastName: string) {
3.     return firstName + " " + lastName;
4. }
5.
6. let result1 = buildName("Bob");           // error, too few parameters
7. let result2 = buildName("Bob", "Adams", "Sr."); // error, too many parameters
8. let result3 = buildName("Bob", "Adams");    // okay and returns "Bob Adams"
9. let result4 = buildName(undefined, "Adams"); // okay and returns "Will Adams"
```

# Typescript. Функции

```
1.  // Overload
2.  function sum(x: any, y: any) {
3.      if (typeof x === 'number') {
4.          return x + y;
5.      } else {
6.          return `${x} ${y}`;
7.      }
8.  }
9.
10. function sum(x: number, y: number)
11. function sum(x: string, y: number) {
12.     if (typeof x === 'number') {
13.         return x + y;
14.     } else {
15.         return `${x} ${y}`;
16.     }
17. }
```

# Интерфейсы



# Typescript. Интерфейсы

```
1.  // Our first interface
2.  function printLabel(labeledObj: { label: string }) {
3.      console.log(labeledObj.label);
4.  }
5.
6.  let myObj = {size: 10, label: "Size 10 Object"};
7.  printLabel(myObj);
8.
9.  // Our second first interface
10. interface LabeledValue {
11.     label: string;
12. }
13.
14. function printLabel(labeledObj: LabeledValue) {
15.     console.log(labeledObj.label);
16. }
17.
18. let myObj = {size: 10, label: "Size 10 Object"};
19. printLabel(myObj);
```

# Классы

# Typescript. Классы

```
1.  // Classes
2.  class Greeter {
3.      greeting: string;
4.      constructor(message: string) {
5.          this.greeting = message;
6.      }
7.      greet() {
8.          return "Hello, " + this.greeting;
9.      }
10. }
11.
12. const greeter = new Greeter("world");
```

# Typescript. Классы

```
1. // Public, private, and protected modifiers
2. class Animal {
3.     public name: string;
4.     public constructor(theName: string) { this.name = theName; }
5.     public move(distanceInMeters: number) {
6.         console.log(`${this.name} moved ${distanceInMeters}m.`);
7.     }
8. }
9.
10. // Readonly, static
11. // Abstract
```

# Generics

# TypeScript. Generics

```
1. // Generic
2. function echo<T>(arg: T): T {
3.     return arg;
4. }
```

# Enums

# TypeScript. Enums

```
1. enum Direction {  
2.     Up,  
3.     Down,  
4.     Left,  
5.     Right,  
6. }  
7.  
8. enum Direction {  
9.     Up = "UP",  
10.    Down = "DOWN",  
11.    Left = "LEFT",  
12.    Right = "RIGHT",  
13. }
```



# Typescript?

Вопросы?



# Практика



Перерыв! (0 минут)

Препоd (с)

# TypeScript 2

# План на сегодня

1. Возможности TypeScript
  - Модули
  - Utility Types
  - Decorators
  - Mixins
2. React TSX

# Модули

# Модули

- export
- import
- modules/first.ts
- modules/second.ts
- modules/third.ts
- modules/types.ts

# Export

1. `export default () => ...`
2. `export {}`
3. `export { ... as ... }`
4. `export { ... as ... } from ...`
5. `export * as ... from ...`



# Import

1. `import ... from ...`
2. `import * as ... from`
3. `import {...} from ...`

# TypeScript

Внешние модули. Ambient  
.D.TS

# TypeScript

Порядок импорта при использовании

```
import ... from '^^^'
```

1. .ts
2. .tsx

# TypeScript

.D.TS - Ambient module declaration

```
1. declare module "Backend" {
2.     export interface ITeacher {
3.         name: string;
4.         level: number;
5.         exp?: number;
6.     }
7.
8.     export interface ILesson {
9.         title: string;
10.        day: number;
11.        hours: number;
12.        teacher: ITeacher;
13.    }
14.
15.    export function makeLesson(title: string, day: number, hours: number, teacher: ITeacher): ILesson;
16. }
```

```
declare module "CumbersomeBackend"
```

# Namespaces

~~Упорядочивание запутывание~~ упорядочивание пространства имен объектов

- namespaces/validators.ts
- namespaces/mergeQuirks.ts

```
1. namespace Shapes {  
2.     export namespace Polygons {  
3.         export class Triangle { }  
4.         export class Square { }  
5.     }  
6. }
```



```
1. const square = new Shapes.Polygons.Square()
```

1. **import** polygons = Shapes.Polygons
2. **const** square = **new** polygons.Square()

1. **let** t = **new** shapes.Shapes.Triangle() // плохо
2. **let** t = **new** shapes.Triangle() // хорошо

- Модули и пространства имен нужны для упорядочивания кода
- Использовать и модули и пространства имен одновременно - избыточно
- Проще всего работать с модулями

# Declaration merging

# Полезные ссылки

- <https://www.typescriptlang.org/>
- [Серия коротких роликов про TS](#)
- ?

# TypeScript. Declaration merging

```
1.  interface Car {  
2.      manufacturer: string;  
3.      name: string;  
4.      vehicleInfo: string;  
5.  }  
6.  
7.  interface Car {  
8.      horsepower: number;  
9.      torque: number;  
10.     stickers: string[];  
11. }  
12.  
13. const realCar: Car {  
14.     manufacturer: 'VAZ',  
15.     name: 'Granta',  
16.     vehicleInfo: 'Realnii avtomobil dlya realnoi jizni',  
17.     horsepower: 98,  
18.     torque: 145,  
19.     stickers: ['my life - my rules', 'dolbit normalno'],  
20. }
```

# Typescript. Declaration merging

## **НЕФУНКЦИОНАЛЬНЫЕ СВОЙСТВА**

- атрибуты интерфейсов должны быть уникальными или одного типа

## **ФУНКЦИОНАЛЬНЫЕ СВОЙСТВА**

- каждое новое свойство считается перегрузкой
- последний интерфейс имеет превосходство над предыдущими
- специализированное свойство будет иметь превосходство над обычными



# Typescript. Declaration merging

```
1.  interface Cloner {  
2.      clone(animal: Animal): Animal;  
3.  }  
4.  
5.  interface Cloner {  
6.      clone(animal: Sheep): Sheep;  
7.  }  
8.  
9.  interface Cloner {  
10.     clone(animal: Dog): Dog;  
11.     clone(animal: Cat): Cat;  
12. }
```

# TypeScript. Declaration merging

```
1. interface Cloner {  
2.     clone(animal: Dog): Dog;  
3.     clone(animal: Cat): Cat;  
4.     clone(animal: Sheep): Sheep;  
5.     clone(animal: Animal): Animal;  
6. }
```

# Typescript. Declaration merging

```
1.  interface Document {  
2.      createElement(tagName: any): Element;  // any  
3.  }  
4.  
5.  interface Document {  
6.      createElement(tagName: "div"): HTMLDivElement;  // specialized  
7.      createElement(tagName: "span"): HTMLSpanElement;  // specialized  
8.  }  
9.  
10. interface Document {  
11.     createElement(tagName: string): HTMLElement;  // general  
12.     createElement(tagName: "canvas"): HTMLCanvasElement;  // specialized  
13. }
```

# Typescript. Declaration merging

```
1. interface Document {  
2.     createElement(tagName: "canvas"): HTMLCanvasElement;  
3.     createElement(tagName: "div"): HTMLDivElement;  
4.     createElement(tagName: "span"): HTMLSpanElement;  
5.     createElement(tagName: string): HTMLElement;  
6.     createElement(tagName: any): Element;  
7. }
```

# Typescript. Namespace merging

```
1. namespace Prison {  
2.     export class Inspector { }  
3.     export class Prisoner { }  
4. }  
5.  
6. namespace Prison {  
7.     export interface Apart { facilities: string[]; }  
8.     export class Yard { facilities: string[] }  
9. }
```

# Typescript. Namespace merging

```
1. namespace Prison {  
2.     export class Inspector { }  
3.     export class Prisoner { }  
4.  
5.     export interface Apartments { facilities: string[]; }  
6.     export class Yard { facilities: string[] }  
7. }
```

# Typescript. Namespace merging

Доступны в общем пространстве имен будут только экспортированные объекты.

`namespaces/mergeQuirks.ts`

# UTILITY TYPES

## \U1F4AA



# Typescript

- **PARTIAL**

Все свойства необязательные

- **READONLY**

Все свойства – readonly

- **REQUIRED**

Все свойства обязательны

- **NONNULLABLE**

Все свойства, кроме null, undefined

# TypeScript

`utilityTypes/commonProps.ts`

# Typescript

- **PICK**

Использовать выбранные свойства

- **OMIT**

Использовать все свойства, кроме выбранных

# TypeScript

`utilityTypes/pickNOmit.ts`

# Typescript. Record

Присваивание свойств одного типа к другому типу.

```
1.  interface PageInfo {  
2.      title: string;  
3.  }  
4.  
5.  type Page = 'home' | 'about' | 'contact';  
6.  
7.  const x: Record<Page, PageInfo> = {  
8.      about: { title: 'about' },  
9.      contact: { title: 'contact' },  
10.     home: { title: 'home' },  
11.  };
```

# TypeScript. Exclude. Extract.

## EXCLUDE

Взять только те свойства, которых нет во втором объекте

1. `type T0 = Exclude<"a" | "b" | "c", "a">; // "b" | "c"`
2. `type T1 = Exclude<"a" | "b" | "c", "a" | "b">; // "c"`
3. `type T2 = Exclude<string | number | (() => void), Function>; // string | number`

## EXTRACT

Взять только те свойства, которые есть во втором объекте

1. `type T0 = Extract<"a" | "b" | "c", "a" | "f">; // "a"`
2. `type T1 = Extract<string | number | (() => void), Function>; // () => void`

# TypeScript

- **PARAMETERS**
- **CONSTRUCTORPARAMETERS**
- **RETURNTYPE**
- **INSTANCETYPE**
- **THISPARAMETERTYPE**

# Decorators



# Typescript. Decorators

OOP Design pattern

Расширение функционала вызываемого объекта без его модификации

*O* from solid

/oop/commonDecorators.ts

```
1.  const noisy = (func) => {  
2.      const wrapper = (...args) => {  
3.          console.log('ppphhhhffffchfiu')  
4.          const res = func(...args)  
5.          console.log('khhhhhheeeeeeffph')  
6.          return res  
7.      }  
8.      return wrapper  
9.  }
```

# Typescript. Decorators

>> A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter.

```
1.  const sum = (a, b) => (a + b)
2.  const noisySum = noisy(sum)
3.  noisySum(1, 15)
```

# Typescript. Decorators

```
tsc --target ES5 --experimentalDecorators
```

```
"experimentalDecorators": true
```

```
oop/typescriptDecorators.ts
```

# TypeScript. Decorators

```
1.  // first
2.  @first @second myFunction
3.
4.  // second
5.  @first
6.  @second
7.  myFunction
```

# Typescript. Сигнатура декоратора

1. `target` - метод, к которому применяется декоратор
2. `key` - имя метода
3. `value` - дескриптор свойства

# Typescript. Decorators

Дескриптор свойства

Object.getOwnPropertyDescriptor

oop/typescriptDecorators.ts

## СОЗДАНИЕ ДЕКОРАТОРА. Decorator factories

```
1. function enumerable(value: boolean) {  
2.     return function (target: any, propertyKey: string, descriptor: PropertyDescriptor) {  
3.         descriptor.enumerable = value;  
4.     };  
5. }
```

# Mixins

Примеси для классов

# Typescript. Mixins

```
1. function applyMixins(derivedCtor: any, baseCtors: any[]) {  
2.     baseCtors.forEach(baseCtor => {  
3.         Object.getOwnPropertyNames(baseCtor.prototype).forEach(name => {  
4.             Object.defineProperty(derivedCtor.prototype, name,  
Object.getOwnPropertyDescriptor(baseCtor.prototype, name));  
5.         });  
6.     });  
7. }
```



# Typescript. Mixins

```
1.  // Activatable Mixin
2.  class Activatable {
3.      isActive: boolean;
4.      activate() {
5.          this.isActive = true;
6.      }
7.      deactivate() {
8.          this.isActive = false;
9.      }
10. }
```

# Typescript. Mixins

```
1.  // Disposable Mixin
2.  class Disposable {
3.      isDisposed: boolean;
4.      dispose() {
5.          this.isDisposed = true;
6.      }
7.  }
```

# Typescript. Mixins

```
1. class SmartObject {  
2.     constructor() {  
3.         setInterval(() => console.log(this.isActive + " : " + this.isDisposed), 500);  
4.     }  
5.  
6.     interact() {  
7.         this.activate();  
8.     }  
9. }
```

# Typescript. Mixins

oop/mixins.ts

```
1. interface SmartObject extends Disposable, Activatable {}  
2. applyMixins(SmartObject, [Disposable, Activatable]);  
3.  
4. const smartObj = new SmartObject();  
5. setTimeout(() => smartObj.interact(), 1000);
```

# Typescript.TSX

tsx/button.tsx

# Typescript.Типы

<https://github.com/microsoft/TypeScript/blob/master/src/lib/dom.generated.d.ts>

# Typescript. Еще типы

<https://github.com/DefinitelyTyped/DefinitelyTyped/tree/master/types>

# TypeScript2?

Вопросы?





# Домашнее задание №12

1. Разработать утилиту на TypeScript
2. ?

Расширенное описание задания, подсказки, а также презентации с лекций всегда есть в репозитории.

Срок сдачи

*29 декабря*

# Спасибо за внимание!



# Пока!

Присоединяйтесь к сообществам про образование в VK

- [VK Джуниор](#)
- [VK Образование](#)

