~~Java~~TypeScript

Web Технологии (ИС) 22/23

Днес

- JavaScript в един слайд (преговор)
- Какво не му харесваме?
- Каква (лоша) стратегия можем да приложим
- Как е по-добре да подходим
- (вечни мъдрости за компютрите)

Какво беше съществено за JavaScript?

- прототипно базирано ООП
- функциите са „първокласни“ обекти
- динамично типизиран
- слабо типизиран

Какво му е на JavaScript?

- прототипно базирано ООП
- функциите са „първокласни“ обекти
- *динамично типизиран*
- слабо типизиран

Какво му е на JavaScript?

- прототипно базирано ООП
- функциите са „първокласни“ обекти
- *динамично типизиран*
- **слабо** типизиран

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

console.log(sum(a, b))
console.log(sum(a, c))
```

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

console.log(sum(a, b))
console.log(sum(a, c))
```

CROWD PARTICIPATION MOMENT

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

console.log(sum(a, b))  // 31
console.log(sum(a, c))
```

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

console.log(sum(a, b))   // 31
console.log(sum(a, c))   // 15LOL
```

Идиоматичен (макар и странен) начин да направим нещо n пъти

```
[...new Array(10)].forEach(() => console.log('hi!'))
```

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, b))].forEach(() => console.log('hi!'))
```

Компютрите са ужасни, защото правят **точно каквото им кажеш**

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, b))].forEach(() => console.log('hi!'))
```

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, b))].forEach(() => console.log('hi!'))
```

НЕ МИГАЙТЕ!

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, c))].forEach(() => console.log('hi!'))
```

Защо това е чак такъв проблем?

```
function sum(a, b) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, c))].forEach(() => console.log('hi!'))
```

Мигнахте ли?

Защо това е чак такъв проблем?

```
new Array(10) // [ <10 empty items> ]
new Array('10baba') // [ '10baba' ]


new Array(10).length // 10
new Array('10baba').length // 1
```

Единственото по-лошо нещо от код, който се чупи
е код, който **не се** чупи.

Искаме код, който да се чупи

```
function sum(a, b) {
 if (typeof a !== 'number') {
   throw 'First argument must be a number'
 }
 if (typeof b !== 'number')  {
       throw 'Second argument must be a number'
 }
 return a + b
}
```

Този обаче освен грозен е и бавен

```
function sum(a, b) {
 if (typeof a !== 'number') {
   throw '😞'
 }
 if (typeof b !== 'number')   {
       throw '😞'
 }
 return a + b
}
```

- 4 от 5-те логически реда се занимават с проверки на типове
- всяко изпълнение на функцията винаги включва проверките
- реално **не сме решили проблема**, просто ще има явни вместо тихи грешки в продукция

- това е много прост малък пример, в живия живот е далеч по-страшно

Така би било чудесно

```
function sum(a: number, b:number) {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, b))].forEach(() => console.log('hi!'))
```

Така би било чудесно

```
function sum(a: number, b:number):number {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, b))].forEach(() => console.log('hi!'))
```

Така би било чудесно

```
function sum(a: number, b:number):number {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, c))].forEach(() => console.log('hi!'))
```

Така би било чудесно

```
function sum(a: number, b:number):number {
 return a + b
}

const a = 15
const b = 16
const c = 'LOL'

[...new Array(sum(a, c))].forEach(() => console.log('hi!'))

// Argument of type 'string' is not assignable to parameter of
type 'number'.ts(2345)
```


TS


TS

Кратка справка

- технически е надмножество на ECMAScript 2015, но стандартния компилатор може да бъде настроен да поддържа и особености на по-нови версии
- Apache License 2.0

Кратка справка

- технически е надмножество на ECMAScript 2015, но стандартния компилатор може да бъде настроен да поддържа и особености на по-нови версии
- Apache License 2.0
- създаден е и се поддържа от **Microsoft**

Благини в TypeScript спрямо JavaScript

- силна типизация
- (опционално) статична типизация
- компилационна стъпка, която да ни предпази от много възможни грешки, за които иначе трябва да пишем тестове

Типови анотации

Първото нещо, което виждаме като разлика с JavaScript

```
const age:number = 42
const name:string = 'Stancho'
const verified:boolean = true

const numbers:number[] = [1, 2, 3, 4, 5]
```

Типови анотации

„Съставни“ типове

```
const accounts:{
 age: number;
 name: string;
 verified: boolean;
} = {
 age: 42,
 name: 'Stancho',
 verified: true
}
```

Типови анотации

Можем да ги изнесем в интерфейси

```
interface Account {
 age: number;
 name: string;
 verified: boolean;
}

const account:Account = {
 age: 42,
 name: 'Stancho',
 verified: true
}
```

Типови анотации

Функциите също имат типове

```
type BinaryNumberOperation = (a:number, b:number) => number

const MatchOver = (
 f1: BinaryNumberOperation,
 f2: BinaryNumberOperation,
 numbers: [number, number][]
 ) => numbers.filter(
   ([n1, n2]) => f1(n1, n2) === f2(n1, n2)
   ).length === numbers.length
```

Интерфейси и именувани типови

```
interface Account {
 age: number;
 name: string;
 verified: boolean;
}
```

обаче

```
type BinaryNumberOperation =
  (a:number, b:number) => number
```

Интерфейси и именувани типове

```
interface Account {
 age: number;
 name: string;
 verified: boolean;
}
```

обаче

```
type BinaryNumberOperation =
  (a:number, b:number) => number
```

Интерфейси и именувани типове

```
interface Account {
 age: number;
 name: string;
 verified: boolean;
}
```

Описва „фòрмата“ на обект

```
type BinaryNumberOperation =
  (a:number, b:number) => number
```

Дава (потенциално още едно) ново име на някой тип

Интерфейси и именувани типове

Интерфейси за „форми“, именувани типове за по-сложни конструкции

```
interface Rectangle {
 width: number;
 height: number;
}

interface Circle {
 radius: number
}

type Shape = Rectangle | Circle
```

Интерфейси и именувани типове

Но технически не е грешно и така

```
type Rectangle = {
 width: number;
 height: number;
}

type Circle = {
 radius: number
}

type Shape = Rectangle | Circle
```

Литерални типове

```
let a:'a'|'b' = 'something else'
```

Литерални типове

```
let a:'a'|'b' = 'something else'
```

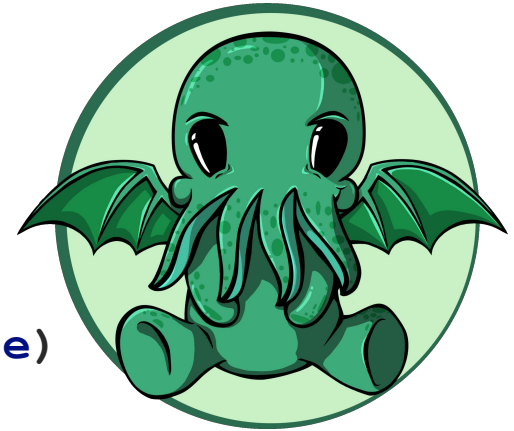
```
Type '"something else"' is not assignable to type
'"a" | "b"'.ts(2322)
```

Можем откровено да прекалим

```
let monstrosity:
  | "this exact string"
  | 3.141593
  | ((n:number, s:string, r:Rectangle) => Circle)
  | [BinaryNumberOperation][]
  | { sparrow_type: string; velocity: number }
  | "Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn";
```

Можем откровено да прекалим

```
let monstrosity:
  | "this exact string"
  | 3.141593
  | ((n:number, s:string, r:Rectangle) => Circle)
  | [BinaryNumberOperation][]
  | { sparrow_type: string; velocity: number }
  | "Ph'nglui mglw'nafh Cthulhu R'lyeh wgah'nagl fhtagn";
```



Типови параметри / Generics

Типовете не винаги трябва да са ясни

```
function swap<T>(items:T[], i1:number, i2:number):T[] {
 const result:T[] = [...items]

 result[i1] = items[i2]
 result[i2] = items[i1]

 return result
}
```

Типови параметри

Типовете не винаги трябва да са ясни

```
function sort<T>(
  items:T[],
  comparator:(a:T, b:T) => -1|0|1
):T[] {
  ...
}
```

Стесняване на типове

Можем да конкретизираме типа на променлива описана с по-сложен тип

```
interface Person {
 firstName: string
 lastName: string
 nickname: string
}

interface Pet {
 name: string
}

type Being = Pet | Person
```

Стесняване на типове

Можем да конкретизираме типа на променлива описана с по-сложен тип

```
function extractName(being: Being): string {
 if ('firstName' in being) {
   return `${being.firstName} "${being.nickname}"
${being.lastName}`
 } else {
   return being.name
 }
}
```

Стесняване на типове

Можем да конкретизираме типа на променлива описана с по-сложен тип

```
interface Person {
 firstName: string
 lastName: string
 nickname: string
}

interface Pet {
 name: string
}
```

```
interface Country {
 internationalName: string
 formOfGovernment: string
}

type Being = Pet | Person | Country
```

Стесняване на типове

Можем да конкретизираме типа на променлива описана с по-сложен тип

```
function extractName(being: Being): string {
 let name = ''
 if ('firstName' in being) {
   const {firstName, lastName, nickname} = being
   name = `${firstName} "${nickname}" ${lastName}`
 } else if ('name' in being){
   name = being.name
 } else {
   assertExhausted(being)
 }

 return name
}
```

Стесняване на типове

Можем да конкретизираме типа на променлива описана с по-сложен тип

```
function assertExhausted(arg:never): void {
 throw (`${arg} exists`)
}
```

Среди за разработка

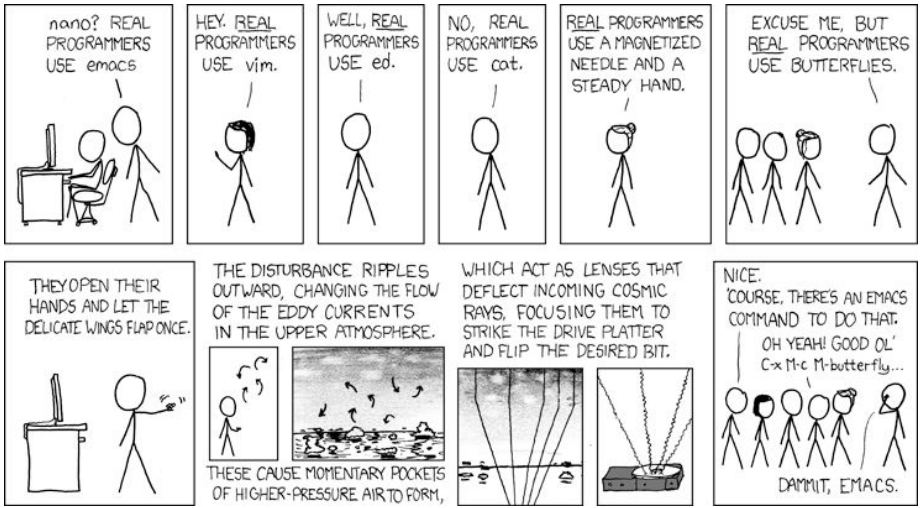
TypeScript е език като всеки друг. Може да ползвате средата, с която сте свикнали най-много.

Има инструменти и инструкции как да настроите всеки относително популярен редактор или IDE за **пълноценна** работа с TypeScript.

Среди за разработка

TypeScript е език като всеки друг. Може да ползвате средата, с която сте свикнали най-много.

Има инструменти и инструкции как да настроите всеки относително популярен редактор или IDE за **пълноценна** работа с TypeScript.



Среди за разработка

TypeScript е език като всеки друг. Може да ползвате средата, с която сте свикнали най-много.

Има инструменти и инструкции как да настроите всеки относително популярен редактор или IDE за **пълноценна** работа с TypeScript.

...

Следват думи, от които душата ми страда

Що се отнася до TypeScript

VSCode is the standard text editor[*]

[*] https://www.gnu.org/fun/jokes/ed-msg.en.html

VSCode

- технически е open source проект разпространяван под MIT лиценз (https://github.com/microsoft/vscode)
- дефакто това, което сваляте от официалния сайт (https://code.visualstudio.com/) има някои добавки:
  - лесен достъп до разширения за VSCode от пазара на Microsoft (https://marketplace.visualstudio.com/vscode)
  - телеметрия за това какво правите, която отива при Microsoft

VSCodium

- може да си спестите телеметрията, като си го компилирате сами
- или, ако все пак предпочитате да запазите разума си, VSCodium (https://vscodium.com/) предоставя изпълними файлове получени от компилиране на официалното хранилище
  - **плюс**: Microsoft не получават телеметрия
  - **минус**: пазара за разширения на Microsoft позволява директна интеграция само с официалните им дистрибуции, VSCodium използва https://open-vsx.org/, където някои от разширенията от пазара на Microsoft не са достъпни