TypeScript
или зачем так сложно

Антон Лобов
Разработчик
JetBrains

Anton.Lobov@JetBrains.com
@zhuravlik26

Антон Лобов

Разработчик
JetBrains

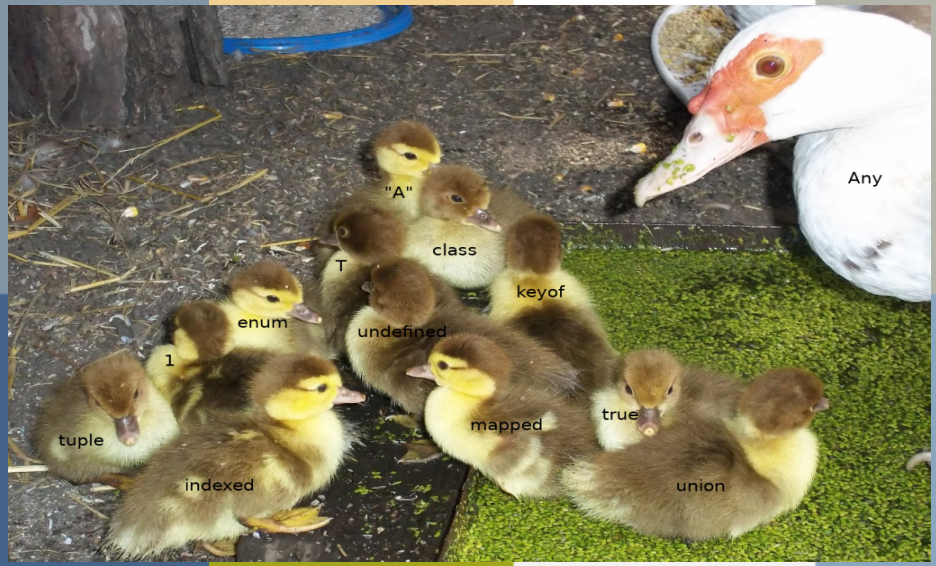
ReSharper → WebStorm

Anton.Lobov@JetBrains.com
@zhuravlik26

1.

Что такое TypeScript?

и его система типов



TypeScript

- Синтаксическое надмножество последней версии ECMAScript
- Компилируется в ECMAScript 3+
- Строгая типизация для кода

Тип данных

- Множество значений и операций над этими значениями
- Переменная имеет тип ‘A’ === набор значений и операций фиксирован
- Динамический = слабо фиксирован
- Статический = жестко фиксирован

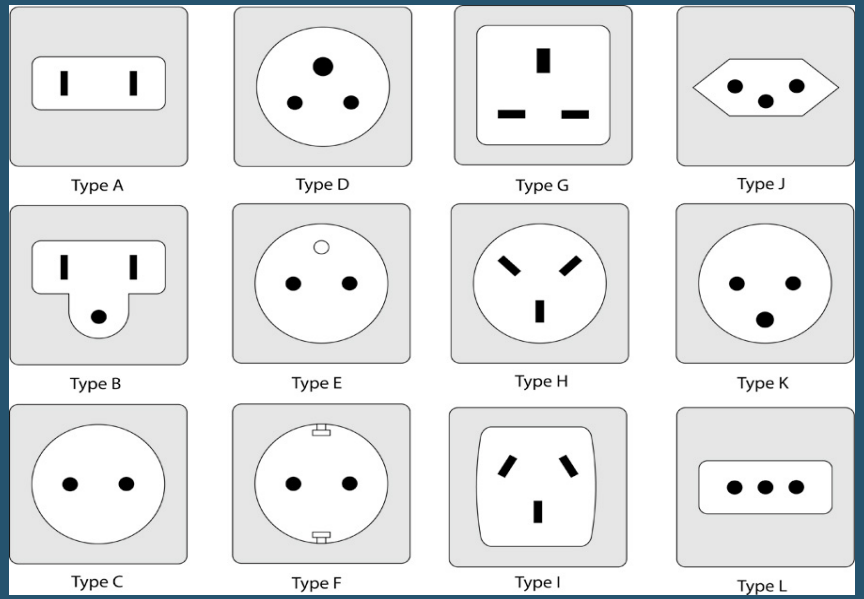
Динамическая типизация

ожидание

реальность

Строгая типизация

ожидание

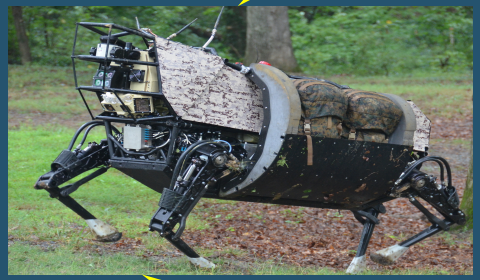


реальность

Иерархическая типизация

ILegged

IWinged



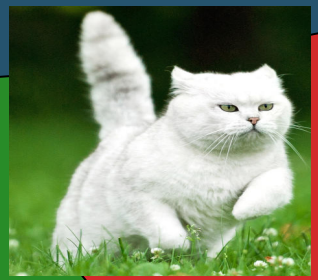
IBeast

IMachine

Структурная типизация

type Legged

{ legsCount: number; }



type Beast

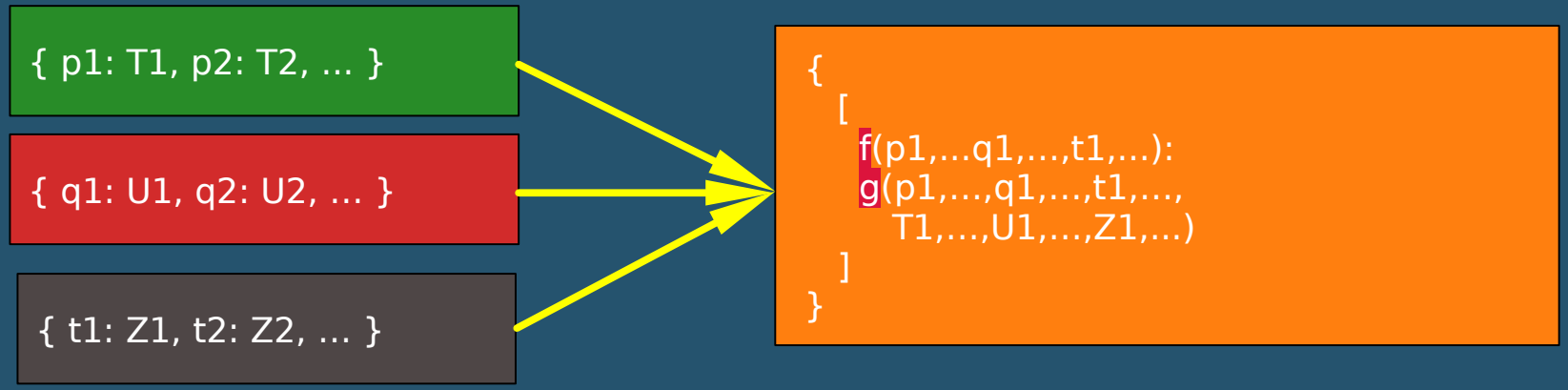
{ lifespan: number;
breed: string; }

type Machine

{ qaPassed: boolean;
fuelAmount: number; }

Операция над типами

- Пара отображений (f, g)
- Первое f – над множеством ключей
- Второе g – над ключами и их типами



Операции над типами TS

- Union (f: intersect, g: T → T)
- Intersection (f: combine, g: T → T)
- Indexed (f: *, g: p → prop(T, nameof p))
- Keyof (f: *, g: p → nameof p)
- Mapped (f: custom-enum, g: custom(T))
- ....

Воображаемый мир

- Все типы стираются при компиляции
- Все существующие JS-библиотеки описаны .d.ts-файлами
- Качество вывода и проверки типов зависит напрямую от качества .d.ts-файлов

- declare var $: any;

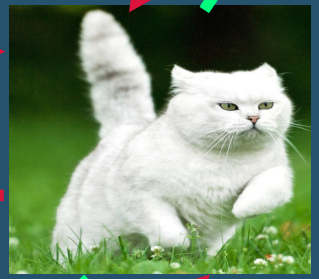
2.

Приключения с типами

когда средства разработки беспомощны перед изяществом языка

Приключение первое: Структурная совместимость

type Beast = { lifespan: number; breed: string; }



type Legged = { legsCount: number; run(): void; }

Структурная совместимость

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

class Girl {
  age: number;
  name: string;
}

var boy: Boy = new Girl();
```

Структурная совместимость

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

class Girl {
    age: number;
    name: string;
}

var boy: Boy = new Girl(); // it works!
```

Структурная совместимость и rename

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

class Girl {
? name: string;
  age: number;
}

var boy: Boy = new Girl();
boy.name = "Vasya";
```

Переименовали?

```
class Boy {
  firstName: string;
  age: number;
}

class Girl {
  name: string;
  age: number;
}

var boy: Boy = new Girl();
boy.firstName = "Vasya";
```

Compilation failed

Эпизод 2: скрытая угроза

```
class Boy { name: string; age: number; football: boolean; /*..*/ }
class Girl { name: string; age: number; dancing: boolean; /*..*/ }

type Child = { name: string; age: number; };
function add(child: Child) { /*add*/ }

add(new Boy("Vasya", 15, true));
add(new Girl("Lena", 18, false));
```

Эпизод 2: скрытая угроза

```
class Boy { name: string; age: number; football: boolean; /*..*/ }
class Girl { name: string; age: number; dancing: boolean; /*..*/ }

type Child = { name: string; age: number; };
function add(child: Child) { /*add*/ }

add(new Boy("Vasya", 15, true));
add(new Girl("Lena", 18, false));
```

Q: Как переименовать?
A: Никак.

Самая умная IDE предложит пользователю список всех объектов с полем ‘name’.
Сколько у вас в коде объектов с полем ‘name’?

Приключение второе. Контекстная типизация

```
type Person = { name: string; age: number };

function addPerson(person: Person) { /*add*/ }

addPerson({ name: 'Ivan', age: 50 });
```

Приключение второе.
Контекстная типизация

```
type Person = { name: string; age: number };

function addPerson(person: Person) { /*add*/ }

addPerson({ name: 'Ivan', age: 50 });
```

Умная IDE должна уметь вывести контекстный тип
и переименовать оба.
Но если рядом встречается литерал с полем ‘name’,
то его уже нельзя трогать.

Контекстная типизация. Сайд-эффект

```
type Person = { name: string; age: number };

function addPerson(person: Person) { /*add*/ }

var ivan = { name: 'Ivan', age: 50 };
var ivan2: { name: string; age: number } = ivan;

addPerson(ivan);
```

Переименование должно пройти всю цепочку
Или опять выдать все типы с полем ‘name’.
Обычно происходит второе.

Приключение третье. Литеральные типы

- Литеральные перечисления, keyof, mapped

- type T = “a” | “b” | “c”
- class A { a; b; }
- keyof –тип: keyof A
  → “a” | ”b”
- mapped: {[P in keyof A | T]: string}
  → {a: string; b: string; c: string}

Что не так с keyof

```
interface Dog {
  name: string;
  id?: number;
  weight: number;
}

type Doggy = Pick<Dog, 'name' | 'id'>;

function choose<T, K extends keyof T>(x: T, prop: K): T[K] /*..*/

function f(o: Doggy) {
   choose(o, 'name');
   console.log(o.id, o.name);
}
```

Что не так с keyof

```
interface Dog {
  name: string;
  id?: number;
  weight: number;
}

type Doggy = Pick<Dog, 'name' | 'id'>;

function choose<T, K extends keyof T>(x: T, prop: K): T[K] /*..*/

function f(o: Doggy) {
   choose(o, 'name');
   console.log(o.id, o.name);
}
```

Литеральные радости

```
interface Dog {
  name: string;
  id?: number;
  weight: number;
}

type Doggy = Pick<Dog, 'name' | 'id'>;

function choose<T, K extends keyof T>(x: T, prop: K): T[K] /*..*/

function f(o: Doggy) {
   choose(o, 'name');
   console.log(o.id, o.name);
}
```

https://github.com/Microsoft/TypeScript/issues/15370

Design Limitation

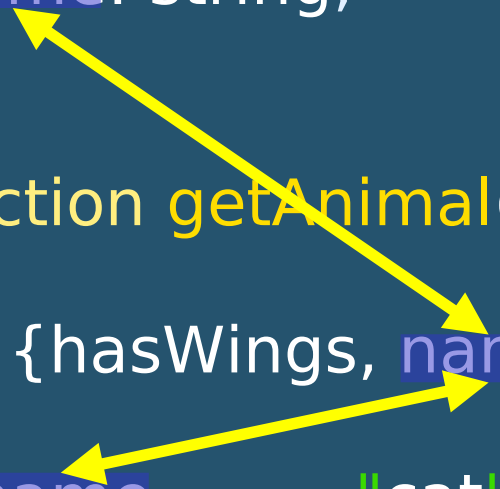
Однако умная IDE с таким успешно справится.
Пока что справится.

Приключение IV. Деструктуризация

```
class Animal {
  hasWings: boolean;
  name: string;
}

function getAnimal(): Animal { /*..*/ }

var {hasWings, name} = getAnimal();

if (name === "cat"){
  assert(!hasWings);
}
```

Деструктуризация и rename

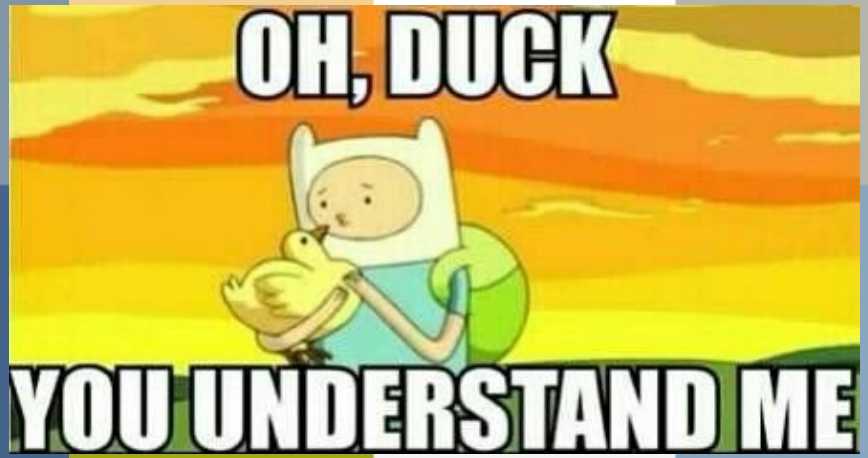
```
class Animal {
  hasWings: boolean;
  name: string;
}

function getAnimal(): Animal { /*..*/ }

var {hasWings, name} = getAnimal();

if (name === "cat"){
  assert(!hasWings);
}
```



3.

Зачем так сложно

или зачем нужна такая система типов как у языка TypeScript



Сложная модель типов

- Class
- Interface
- Primitives (string/number/boolean/symbol)
- Nullable values (null/undefined)
- Enum
  - Const enum
  - Literal enum
- Object literal types
  - Fresh
  - Non-fresh
- Tuple
- Alias
- Union
- Intersection
- Types-constraints
  - X is T
  - this is T
- This-type
- For-in variable
- Generics
- Literal types
  - String
  - Numeric
  - Boolean
  - Enum
  - Freezed/widened
- Never
- Keyof
- Indexed types
- Mapped types
- Object-literal-rest-property-type
- .....

Простая модель типов

- Любой объект в JS – это набор пар ключ-значение
- Любой тип в TS – это набор пар ключ-тип
- Вся модель типов TS
  - Примитивы
  - Ключи
  - Комбинации ключ-тип – объектные типы
  - Индексеры – маски над ключами

Способы задать один тип

```
class A1 { a: number, b: number }
type A2 = { a: number, b: number }
type Keys = “a” | “b”
type A3 = { [T in Keys]: number }
type A4 = { [T in Keys]: A1[T] }
type A5 = { [T in keyof A1]: A2[T] }
enum Items { first = “a”, second = “b”}
type A6 = { [T in Items]: number }
class Foo { a: number }; class Boo { b: number };
type A7 = Foo & Boo
```

“a” ↔ number, “b” ↔ number

4.

Средства разработки и типы

Что могут IDE?

Цепочки типов

```
if (a.foo == 5)
    return a.b(7) + x[“a”];
```

```
plus(a(constraint:props.foo=5)(prop=’b’)(inv:’literal 7’),
   x(indexedaccess=’a’))
```

- Чтобы вычислить тип выражения, цепочка разматывается

Ассоциативный кеш и поиск

```
var a = 5;

declaration($GLOBAL$.a, number:5)

a = 7;

assignment($GLOBAL$.a, number:7)
```

- a ← checkLocals, find(module1.a), …, find(moduleX.a), find($GLOBAL$.a)

Чем IDE лучше компилятора

Компилятор не рассчитан на работу с отзывчивостью на каждое нажатие клавиши

Умная IDE имеет более развитую систему типов и код-модель, чем компилятор

IDE и приключения с типами

TsKeyType – пара (Dog, “name”)
При операциях с типами мы не теряем информацию об исходном типе.
Компилятор ее теряет, потому что компилятору она не важна.

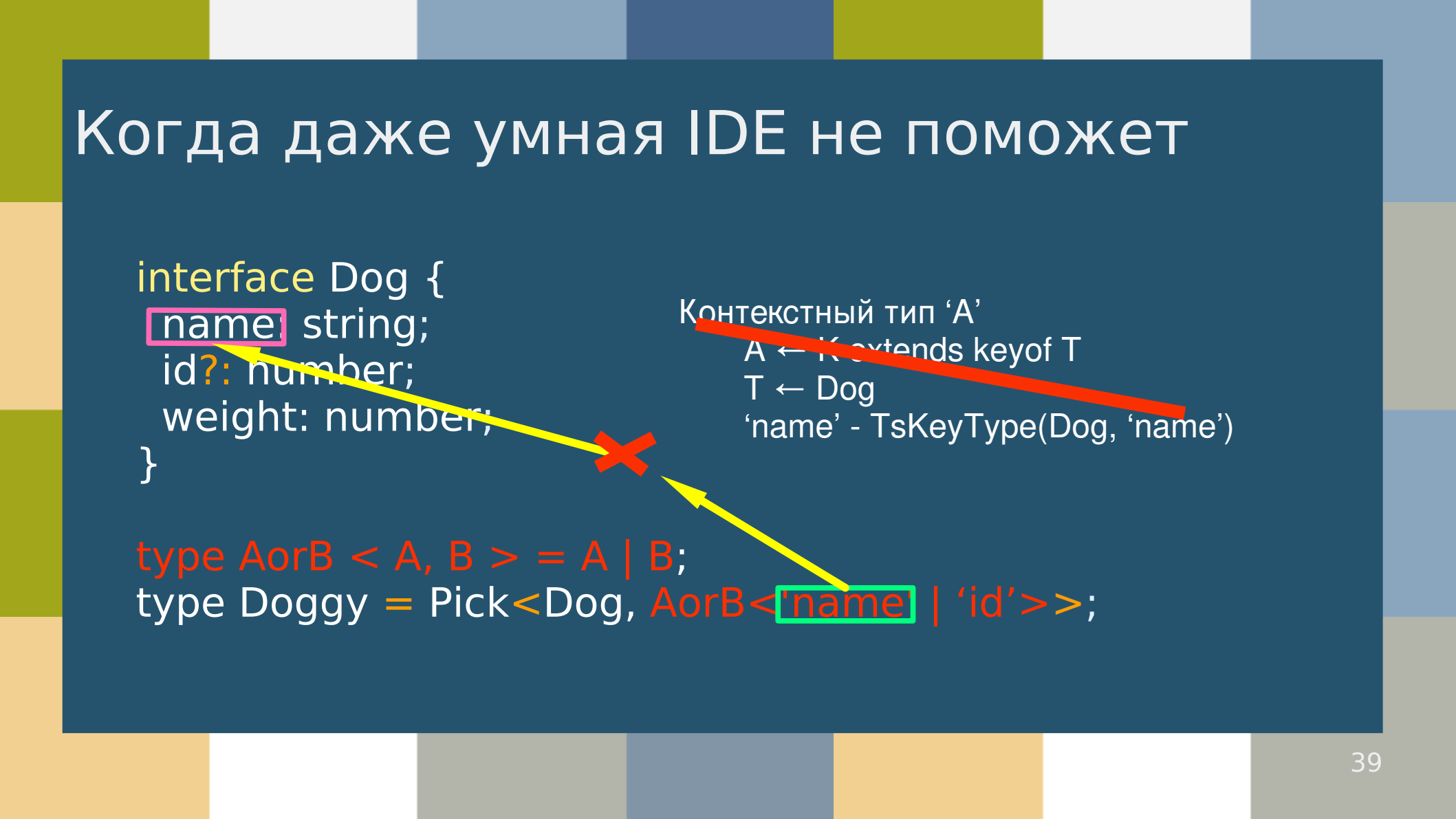
```
interface Dog {
  name: string;
  id?: number;
  weight: number;
}


type Doggy = Pick<Dog, 'name' | 'id'>;
```

Контекстный тип ‘K extends keyof T’
T ← Dog
‘name’ - TsKeyType(Dog, ‘name’)

Когда даже умная IDE не поможет

```
interface Dog {
  name: string;
  id?: number;
  weight: number;
}

type AorB < A, B > = A | B;
type Doggy = Pick<Dog, AorB<'name' | 'id'>>;
```

Контекстный тип ‘A’
A ← K extends keyof T
T ← Dog
‘name’ - TsKeyType(Dog, ‘name’)



5.

Кому жить хорошо?

TypeScript, разработчик и инструменты разработки

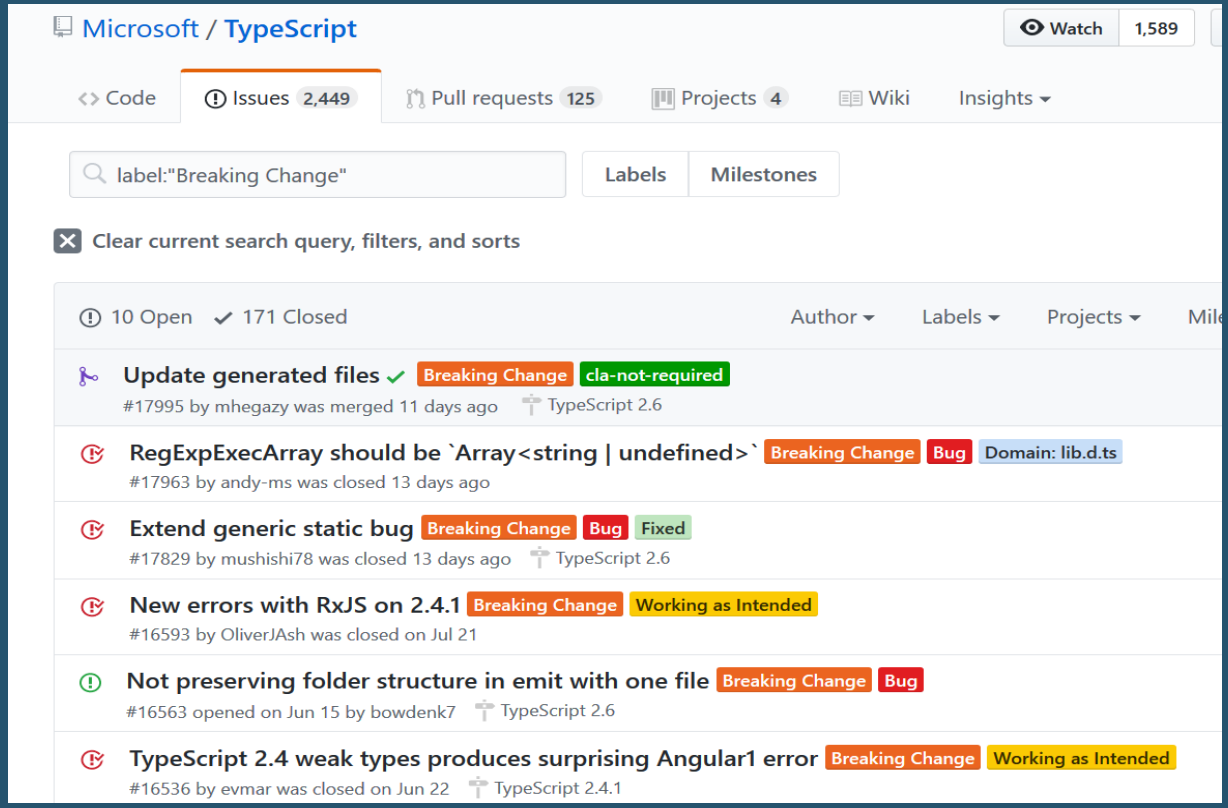
TypeScript

- Активное развитие
- Релизы каждые три месяца
- Поддержка со стороны сообщества

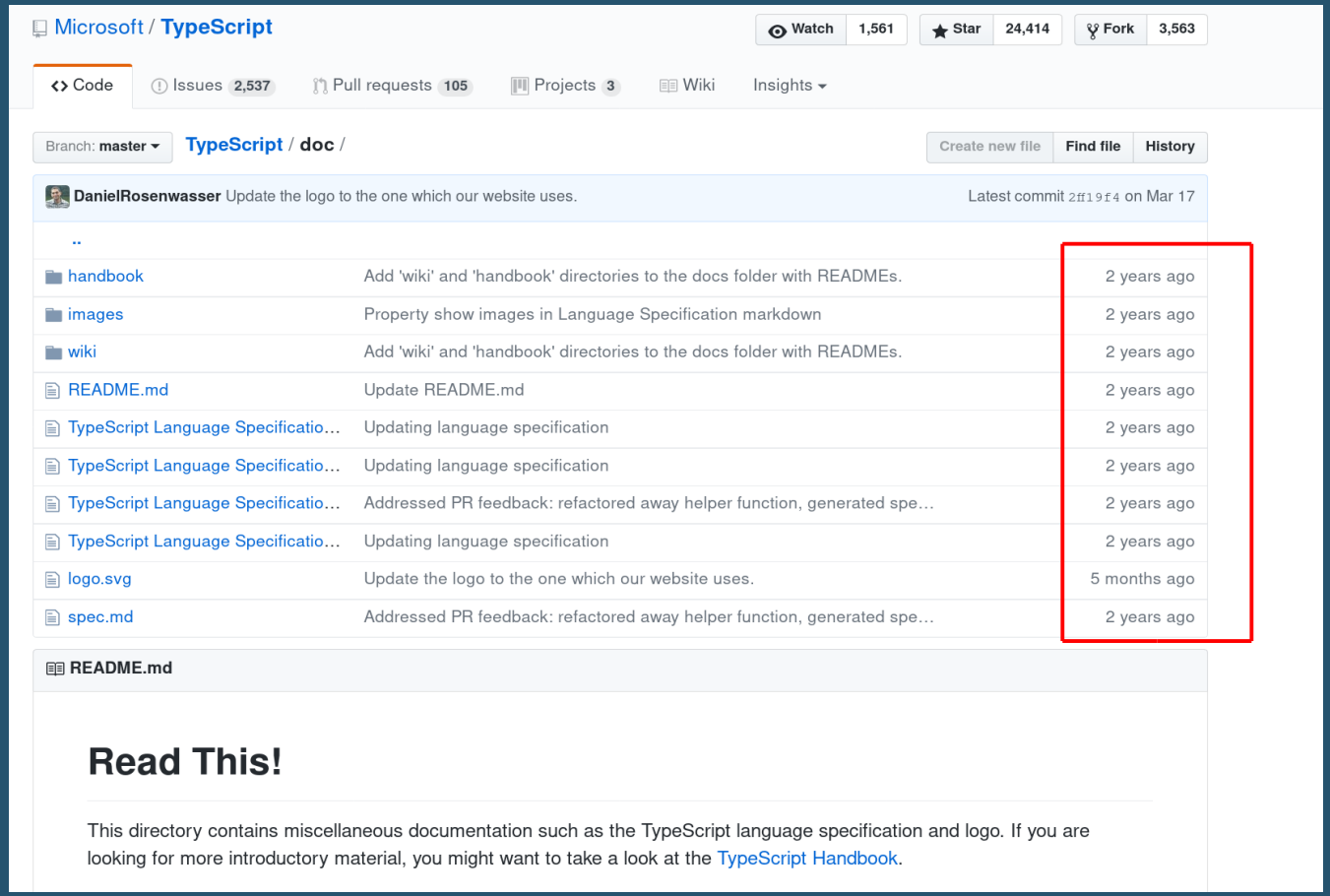
TypeScript

- Breaking changes каждые три месяца
- Несовместимый новый синтаксис
- Отсутствие документации за исключением описаний pull-реквестов и тестдаты

Breaking changes



Спецификация



Microsoft / TypeScript

Watch 1,561 | Star 24,414 | Fork 3,563

Code | Issues 2,537 | Pull requests 105 | Projects 3 | Wiki | Insights

Branch: master ▾ TypeScript / doc / Create new file | Find file | History

DanielRosenwasser Update the logo to the one which our website uses. Latest commit 2ff19f4 on Mar 17

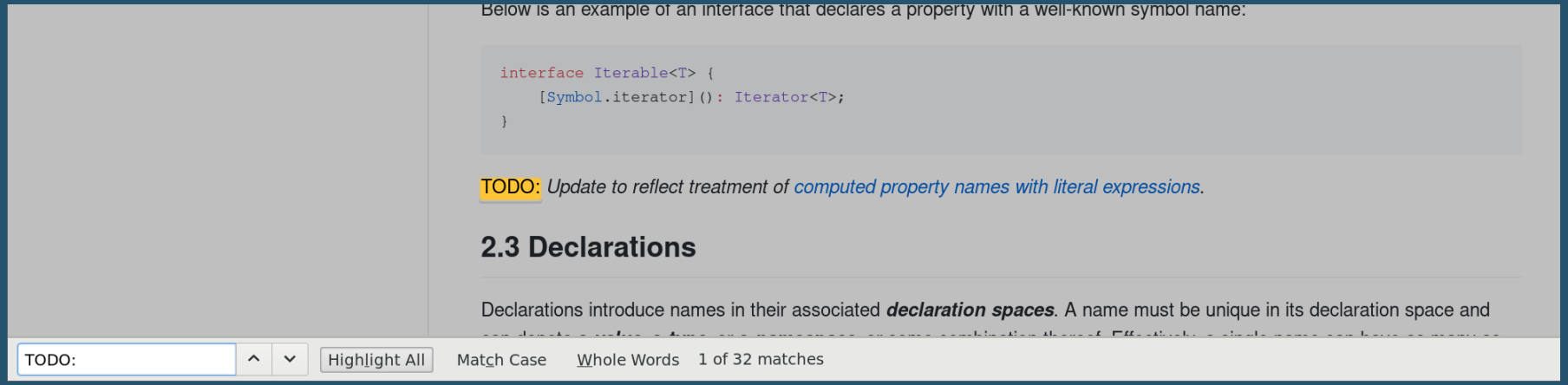
..		
handbook	Add 'wiki' and 'handbook' directories to the docs folder with READMEs.	2 years ago
images	Property show images in Language Specification markdown	2 years ago
wiki	Add 'wiki' and 'handbook' directories to the docs folder with READMEs.	2 years ago
README.md	Update README.md	2 years ago
TypeScript Language Specificatio…	Updating language specification	2 years ago
TypeScript Language Specificatio…	Updating language specification	2 years ago
TypeScript Language Specificatio…	Addressed PR feedback: refactored away helper function, generated spe…	2 years ago
TypeScript Language Specificatio…	Updating language specification	2 years ago
logo.svg	Update the logo to the one which our website uses.	5 months ago
spec.md	Addressed PR feedback: refactored away helper function, generated spe…	2 years ago

README.md

Read This!

This directory contains miscellaneous documentation such as the TypeScript language specification and logo. If you are looking for more introductory material, you might want to take a look at the TypeScript Handbook.

Спецификация



Below is an example of an interface that declares a property with a well-known symbol name:

```
interface Iterable<T> {
    [Symbol.iterator](): Iterator<T>;
}
```

TODO: *Update to reflect treatment of computed property names with literal expressions.*

2.3 Declarations

Declarations introduce names in their associated ***declaration spaces***. A name must be unique in its declaration space and

TODO: Highlight All Match Case Whole Words 1 of 32 matches

Что делать IDE?

- Поддерживать по коммитам
  - поддерживать фичи ASAP самим
  - ReSharper-way
- Поддерживать по коммитам
  - и уточнять у сервиса
  - WebStorm-way
- Полностью полагаться на сервис
  - медленнее и меньше фичей
  - VSCode, Visual Studio

Что делать разработчику

- Радоваться развитию языка
- Понимать, что есть фичи языка, пагубные для средств разработки, и не злоупотреблять ими
- Не ожидать от средств разработки быстрой поддержки новой версии языка

Спасибо за внимание!

Anton.Lobov@JetBrains.com
@zhuravlik26