Эволюция TypeScript

Все чудесатее и чудесатее

Старовойт Андрей

Обо мне

- Работаю в WebStorm с 2014 года
- Занимаюсь языковой поддержкой TypeScript в IDE

План

- Введение в TypeScript
- Зачем нужны типы
- Как TypeScript менялся
- Будущее языка

Введение

TypeScript

Первый публичный релиз в 2012 году

TypeScript

Первый публичный релиз в 2012 году

Создатель — **Anders Hejlsberg**
участвовал в создании C#, Delphi, J++

Anders Hejlsberg

TypeScript

Первый публичный релиз в 2012 году

Создатель — **Anders Hejlsberg**
участвовал в создании C#, Delphi, J++

Я
после очередной крутой фичи
ломающей вообще всё

Anders Hejlsberg

TypeScript

Надмножество JavaScript

TypeScript

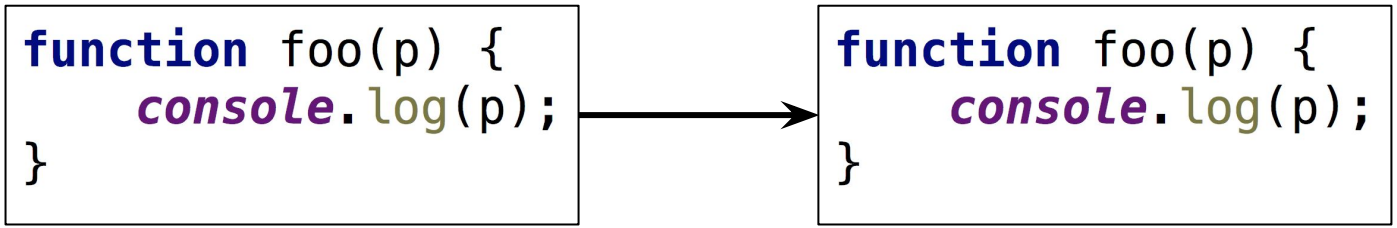
Надмножество JavaScript

```
function foo(p) {
    console.log(p);
}
```

JavaScript

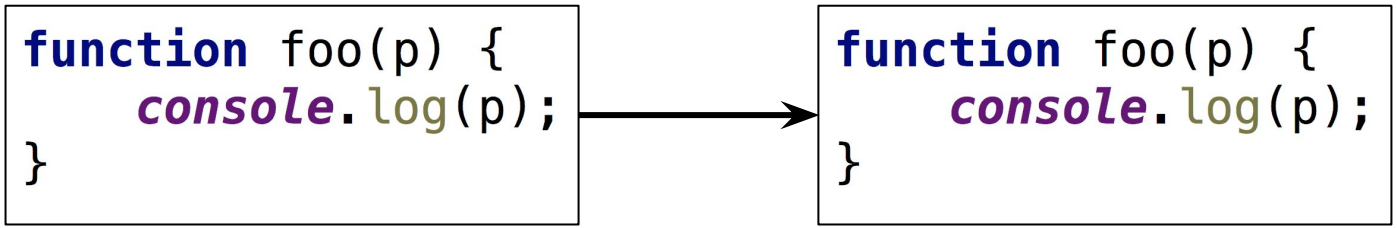
TypeScript

Надмножество JavaScript



TypeScript

Надмножество JavaScript *



* С поправкой на проверку типов

TypeScript

Компилируется в JavaScript (ES3, ES5, ES2015, ...)

TypeScript

Компилируется в JavaScript (ES3, ES5, ES2015, ...)

```
class MyClass {

}
```

TypeScript

Компилируется в JavaScript (ES3, ES5, ES2015, ...)

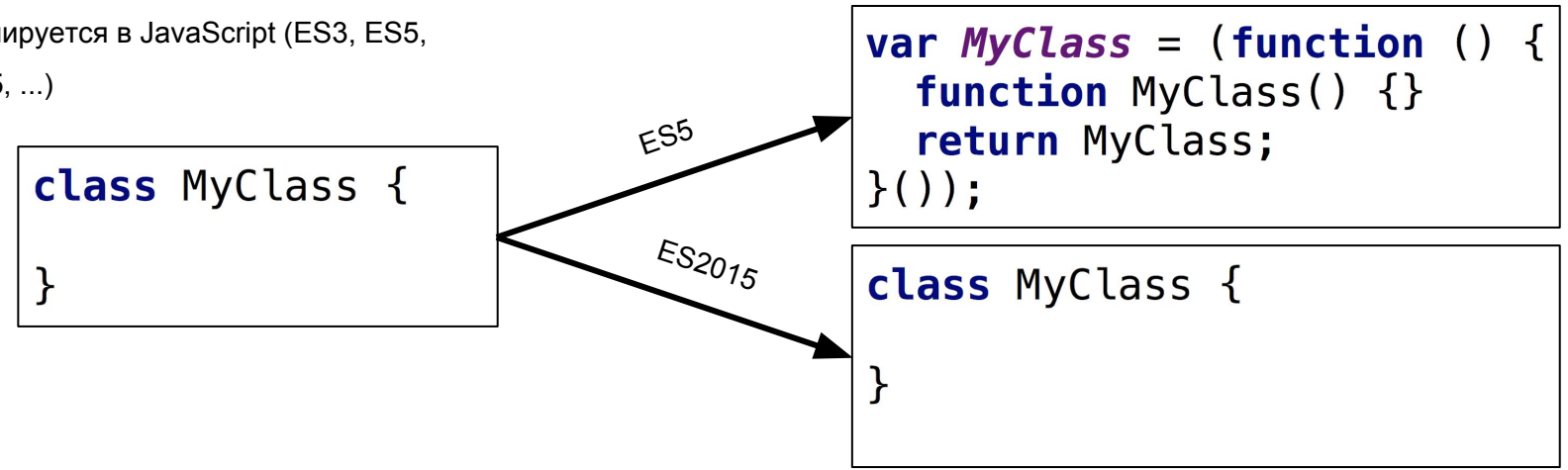
```
class MyClass {

}
```

ES5

```
var MyClass = (function () {
  function MyClass() {}
  return MyClass;
}());
```

TypeScript

Компилируется в JavaScript (ES3, ES5, ES2015, ...)



TypeScript

Статически типизированный язык программирования

TypeScript

Статически типизированный язык программирования

TypeScript

Статически типизированный язык программирования

```
function hello(text: string): string {
    return "hello " + text;
}
```

TypeScript

Структурная типизация

TypeScript

Структурная типизация

Тип определяется структурой, определение не играет роли

```
class Bar1 {
    bar() {}
}

class Bar2 {
    bar() {}
}

let bar1: Bar1 = new Bar2(); //ok
```

Назначение системы типов

Назначение системы типов

Статическая типизация решает проблемы:

- Документирование
- Обнаружение ошибок на этапе компиляции
- Управление поведением в runtime

Назначение системы типов

JSDoc пытался решить часть этих проблем в JavaScript:

```
/**
 * @param {number} p1
 * @param {number} p2
 * @param {number} p3
 * @returns {number}
 */
function sum(p1, p2, p3) {
    return p1 + p2 + p3;
}
```

Система типов TypeScript

TypeScript 1.0

TypeScript 1.0

- Interfaces
- Classes
- Generics
- Overloads

TypeScript 1.0

- Interfaces
- Classes
- Generics
- Overloads

Стандартный набор

ОО языка программирования

TypeScript 1.0

Решает ли проблемы?

- Документирование
- Обнаружение ошибок на этапе компиляции
- Управление поведением в runtime

TypeScript 1.0

Документирование и обнаружение ошибок

Многие стандартные функции и методы JavaScript невыразимы

- *Object.freeze()* — нет readonly (const) свойств
- *Object.assign()* — нет возможности манипулировать свойствами

TypeScript 1.0

Документирование и обнаружение ошибок

Проигрывает в выразительности даже JSDoc

TypeScript 1.0

Документирование и обнаружение ошибок

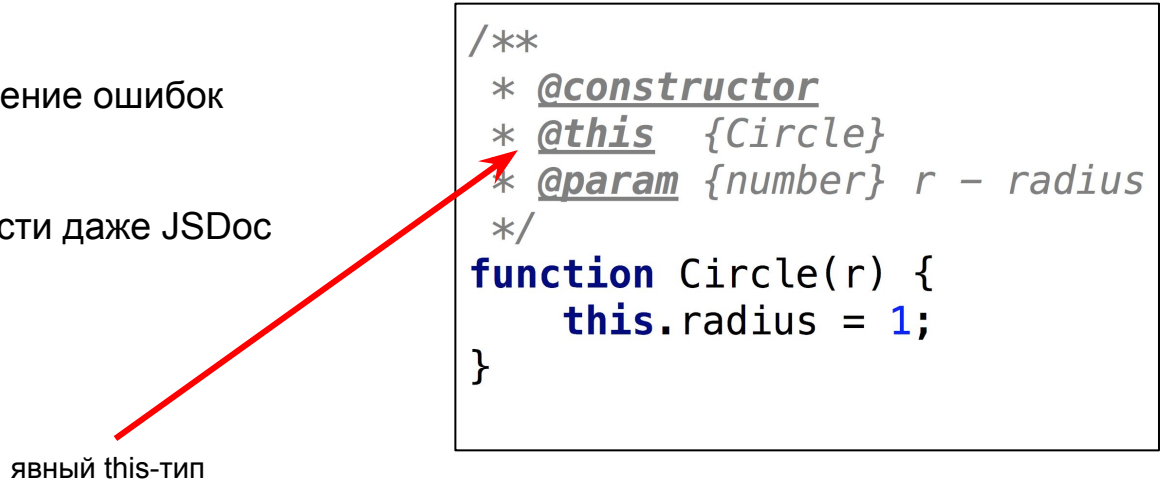
Проигрывает в выразительности даже JSDoc

```
/**
 * @constructor
 * @this  {Circle}
 * @param {number} r - radius
 */
function Circle(r) {
    this.radius = 1;
}
```

TypeScript 1.0

Документирование и обнаружение ошибок

Проигрывает в выразительности даже JSDoc



TypeScript 1.0

Управление поведением в runtime

TypeScript 1.0

Управление поведением в runtime

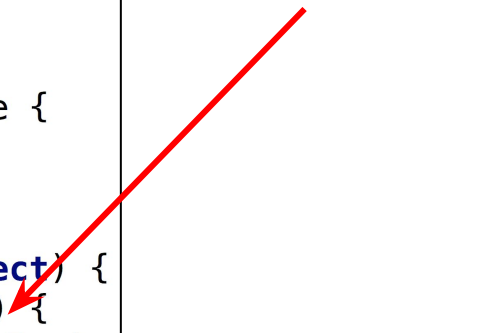
```
interface Base {
    myValue;
}
class Impl implements Base {
    myValue = "impl";
}

function showValue(p: object) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

TypeScript 1.0

Управление поведением в runtime

```
interface Base {
    myValue;
}
class Impl implements Base {
    myValue = "impl";
}

function showValue(p: object) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

Ошибка компиляции



TypeScript 1.0

Управление поведением в runtime

```ts
interface Base {
    myValue;
}
class Impl implements Base {
    myValue = "impl";
}

function showValue(p: object) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

Компиляция

```ts
class Impl {
    constructor() {
        this.myValue = "impl";
    }
}
function showValue(p) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}
showValue(new Impl());
```

TypeScript 1.0

Управление поведением в runtime

```ts
interface Base {
    myValue;
}
class Impl implements Base {
    myValue = "impl";
}

function showValue(p: object) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

Компиляция

```ts
class Impl {
    constructor() {
        this.myValue = "impl";
    }
}
function showValue(p) {
    if (p instanceof Base) {
        console.log(p.myValue);
    }
}
showValue(new Impl());
```

Нигде не определен

TypeScript 1.0

—

JavaScript ничего не знает о типах

TypeScript

TypeScript 1.0

Другие варианты

- В Java ***instanceof*** работает именно так, как мы ожидаем
- В C#, Kotlin есть ***is***
- EcmaScript 4 имел ***is*** оператор и знал о типах в runtime

- В TypeScript можно было реализовать по-другому

TypeScript 1.0

Остаются способы управления runtime на основе JavaScript

TypeScript 1.0

Остаются способы управления runtime на основе JavaScript

1. Оператор ***instanceof,*** работающий для цепочки prototype
2. Оператор ***typeof,*** работающий для ограниченного числа встроенных типов
3. “Примитивный” способ

TypeScript 1.0

Проверка ***instanceof*** в TypeScript

TypeScript 1.0

Проверка ***instanceof*** в TypeScript

```
class Base {
    myValue;
}
class Impl extends Base {
    myValue = "impl";
}
function showValue(p: Object) {
    if (p instanceof Base) {
        console.log((<Base>p).myValue);
    }
}
showValue(new Impl());
```

TypeScript 1.0

Проверка ***instanceof*** в TypeScript

```
class Base {
    myValue;
}
class Impl extends Base {
    myValue = "impl";
}
function showValue(p: Object) {
    if (p instanceof Base) {
        console.log((<Base>p).myValue);
    }
}
showValue(new Impl());
```

Компиляция

```
class Base {}
class Impl extends Base {
  constructor() {
    super();
    this.myValue = "impl";
  }
}
function showValue(p) {
  if (p instanceof Base) {
    console.log(p.myValue);
  }
}

showValue(new Impl());
```

TypeScript 1.0

“Примитивный” способ в JavaScript

TypeScript 1.0

“Примитивный” способ в JavaScript

```
function showValue(p) {
  if (p.marker) console.log(p.myValue);
}

showValue({
  marker: true,
  myValue: "value"
});
```

TypeScript 1.0

“Примитивный” способ в TypeScript

TypeScript 1.0

“Примитивный” способ в TypeScript

```
interface Base {
    marker,
    myValue
}
function showValue(p: Object) {
    if ((<any>p).marker) {
        console.log((<Base>p).myValue);
    }
}

showValue(<Base>{
    marker: true,
    myValue: "value"
});
```

TypeScript 1.0

“Примитивный” способ в TypeScript

```ts
interface Base {
    marker,
    myValue
}
function showValue(p: Object) {
    if ((<any>p).marker) {
        console.log((<Base>p).myValue);
    }
}

showValue(<Base>{
    marker: true,
    myValue: "value"
});
```

Компиляция

```js
function showValue(p) {
  if (p.marker) {
    console.log(p.myValue);
  }
}

showValue({
  marker: true,
  myValue: "value"
});
```

TypeScript 1.0

Проблемы системы типов TypeScript

- Не хватает выразительности для описания стандартной библиотеки
- Нет возможности управлять поведением в Runtime
- Добавляет лишнии операции приведения типа в привычные JavaScript-паттерны

TypeScript 1.4

Вышел в 2015 году

TypeScript 1.4

- Union types
- Type alias

TypeScript 1.4

Union type

```
let numberOrString: number | string;

numberOrString = 1; //ok
numberOrString = ""; //ok
numberOrString = true; //error
```

TypeScript 1.4

В стандартной библиотеке *String.replace()*

```
replace(searchValue: string | RegExp,
        replaceValue: string): string;
```

TypeScript 1.4

Type alias

```
type NumberOrString = number | string;

let numberOrString: NumberOrString;
```

TypeScript 1.5

Вышел в 2015 году

TypeScript 1.5

- Type guard для ***instanceof*** и ***typeof***

TypeScript 1.5

Type guard для ***instanceof***

```
class Base {}
class Extension extends Base {
    myValue = "extension";
}

function showValue(p: Base) {
    if (p instanceof Extension) {
        console.log(p.myValue);
    }
}

showValue(new Extension());
```

TypeScript 1.5

Type guard для ***instanceof***

```
class Base {}
class Extension extends Base {
    myValue = "extension";
}

function showValue(p: Base) {
    if (p instanceof Extension) {
        console.log(p.myValue);
    }
}

showValue(new Extension());
```

“cast” не нужен

TypeScript 1.5

Type guard для ***instanceof***

```
class Base {}
class Extension extends Base {
    myValue = "extension";
}

function showValue(p: Base) {
    if (p instanceof Extension) {
        console.log(p.myValue);
    }
}

showValue(new Extension());
```

Компиляция

```
class Base {}
class Extension extends Base {
  constructor() {
    super();
    this.myValue = "extension";
  }
}

function showValue(p) {
  if (p instanceof Extension) {
    console.log(p.myValue);
  }
}

showValue(new Extension());
```

TypeScript 1.5

Type guard для ***instanceof*** и ***typeof***

Плюсы:

- Избавляет от приведения типов
- Использует стандартный механизм JavaScript

TypeScript 1.5

Type guard для ***instanceof*** и ***typeof***

Минусы:

- Не работает для интерфейсов и type alias

TypeScript 1.6

Вышел в 2015 году

TypeScript 1.6

- Intersection types
- User-defined type guard

TypeScript 1.6

Intersection type

```
type Foo = { foo: string }
type Bar = { bar: string }

let fooBar: Foo & Bar;

fooBar = {foo: "", bar: ""}; //ok
fooBar = {foo: ""}; //error
```

TypeScript 1.6

В стандартной библиотеке *Object.assign()*

```
assign<T, U>(target: T, source: U): T & U;
```

TypeScript 1.6

—

Aliases
Union types
Intersection types

TypeScript 1.6

—

Aliases
Union types
Intersection types

TypeScript 1.6

Type alias, Union types, Intersection types

TypeScript 1.6

Type alias, Union types, Intersection types

- Позволяют свободно оперировать любыми наборами свойств

TypeScript 1.6

Type alias, Union types, Intersection types

- Позволяют свободно оперировать любыми наборами свойств
- Покрывают функциональность интерфейсов

TypeScript 1.6

Type alias, Union types, Intersection types

- Позволяют свободно оперировать любыми наборами свойств
- Покрывают функциональность интерфейсов

```
type Base = { base() };
type Extension = Base & { extended() };

class Impl implements Extension {
    base() {}
    extended() {}
}
```

TypeScript 1.6

User-defined type guard functions

TypeScript 1.6

User-defined type guard functions

- Давайте напишем собственную реализацию ***instanceof***

TypeScript 1.6

User-defined type guard functions

```
interface Base {
    myValue: string;
}
function showValue(p: object) {
    if (isBase(p)) {
        console.log(p.myValue);
    }
}
function isBase(p: any): p is Base {
    return (<any>p).myValue;
}
```

TypeScript 1.6

User-defined type guard functions

```ts
interface Base {
    myValue: string;
}
function showValue(p: object) {
    if (isBase(p)) {
        console.log(p.myValue);
    }
}
function isBase(p: any): p is Base {
    return (<any>p).myValue;
}
```

Компиляция

```ts
function showValue(p) {
    if (isBase(p)) {
        console.log(p.myValue);
    }
}
function isBase(p) {
    return p.myValue;
}
```

TypeScript 1.6

User-defined type guard functions

Плюсы:

- Избавляет от приведения типов
- Универсальное решение

TypeScript 1.6

User-defined type guard functions

Минусы:

- Нужно писать свою реализацию ***instanceof*** для каждого типа
- Нет статической проверки корректности guard function

TypeScript 1.6

User-defined type guard functions

Минусы:

- Нужно писать свою реализацию ***instanceof*** для каждого типа
- Нет статической проверки корректности guard function

```
function isBase(p: any): p is Base {
    return (<any>p).myVulue;
}
```

TypeScript 2.0

Вышел в 2016 году

TypeScript 2.0

- Literal types (boolean, number, enum)
- Discriminated union types

TypeScript 2.0

Literal types (boolean, number, enum)

TypeScript 2.0

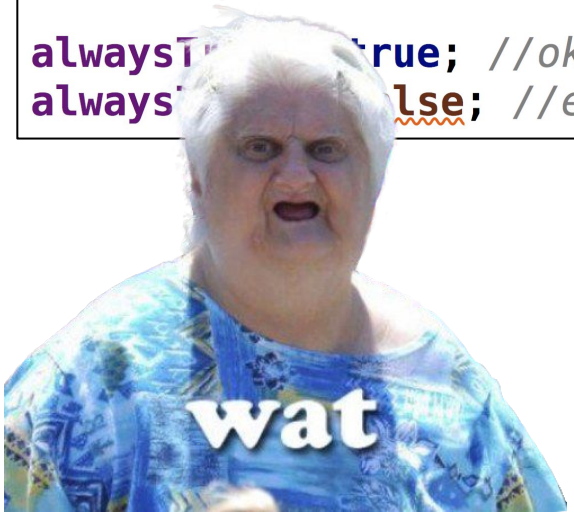
Literal types (boolean, number, enum)

```
let alwaysTrue: true;

alwaysTrue = true; //ok
alwaysTrue = false; //error
```

TypeScript 2.0

Literal types (boolean, number, enum)

```
let alwaysTrue: true;

alwaysT    rue; //ok
always     lse; //error
```



TypeScript 2.0

Discriminated union types

Частный случай:

тип это Union из нескольких известных вариантов (X | Y | ...)

- Маркируем все типы некоторым свойством с разными значениями
- По значению свойства определяем точный тип

TypeScript 2.0

Discriminated union types

TypeScript 2.0

Discriminated union types

```
interface Dog {
    cat:false;
}
interface Cat {
    cat:true;
    meow:string;
}
function showValue(p: Dog | Cat) {
    if (p.cat) {
        console.log(p.meow);
    }
}

showValue( p: {cat:true}); //error
showValue( p: {cat:true, meow:"meow"});
```

TypeScript 2.0

Discriminated union types

```
interface Dog {
    cat:false;
}
interface Cat {
    cat:true;
    meow:string;
}
function showValue(p: Dog | Cat) {
    if (p.cat) {
        console.log(p.meow);
    }
}

showValue( p: {cat:true}); //error
showValue( p: {cat:true, meow:"meow"});
```

TypeScript 2.0

Discriminated union types

```typescript
interface Dog {
    cat:false;
}
interface Cat {
    cat:true;
    meow:string;
}
function showValue(p: Dog | Cat) {
    if (p.cat) {
        console.log(p.meow);
    }
}

showValue( p: {cat:true}); //error
showValue( p: {cat:true, meow:"meow"});
```

Компиляция

```javascript
function showValue(p) {
  if (p.cat) {
    console.log(p.meow);
  }
}

//showValue({cat: true}); //error
showValue({cat: true, meow: "meow"});
```

TypeScript 2.0

Discriminated union types

Плюсы:

- Не требует собственной реализации ***instanceof***
- Статическая проверка корректности

TypeScript 2.0

Discriminated union types

Минусы:

- Работает только для Union типов
- Нужно маркировать все необходимые нам типы некоторым свойством-тегом

TypeScript 2.7

Вышел в начале 2018 года

TypeScript 2.7

- “***in***” guard

TypeScript 2.7

“***in***” guard

Частный случай:

тип это Union из нескольких известных вариантов (X | Y | ...)

- По наличию уникального свойства определяем точный тип

TypeScript 2.7

“***in***” guard

TypeScript 2.7

—

“***in***” guard

```
interface Square {
    size: number;
}
interface Circle {
    diameter: number;
    radius: number;
}
function showValue(p: Square | Circle) {
    if ("radius" in p) {
        console.log(p.diameter);
    }
}
showValue({radius: 1, size: 2}); //error
showValue({radius: 1, diameter: 2});
```

TypeScript 2.7

“***in***” guard

```ts
interface Square {
    size: number;
}
interface Circle {
    diameter: number;
    radius: number;
}
function showValue(p: Square | Circle) {
    if ("radius" in p) {
        console.log(p.diameter);
    }
}
showValue({radius: 1, size: 2}); //error
showValue({radius: 1, diameter: 2});
```

Компиляция

```ts
function showValue(p) {
    if ("radius" in p) {
        console.log(p.diameter);
    }
}
//showValue({ radius: 1, size: 2 }); //error
showValue({ radius: 1, diameter: 2 });
```

TypeScript 2.7

“***in***” guard

Плюсы:

- Не требует собственной реализации ***instanceof***
- Статическая проверка корректности

TypeScript 2.7

“***in***” guard

Минусы:

- Работает только для Union types
- ~~Нужно маркировать все необходимые нам типы некоторым свойством-тегом~~

TypeScript 2.7

Ограничение на Union types

TypeScript 2.7

Ограничение на Union types

Так ли это плохо?

TypeScript 2.7

Ограничение на Union types

```
interface BaseInterface {
    myValue: string;
}
class Impl implements BaseInterface {
    myValue = "impl"
}

function showValue(p: object | BaseInterface) {
    if ("myValue" in p) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

TypeScript 2.7

Ограничение на Union types

```ts
interface BaseInterface {
    myValue: string;
}
class Impl implements BaseInterface {
    myValue = "impl"
}

function showValue(p: object | BaseInterface) {
    if ("myValue" in p) {
        console.log(p.myValue);
    }
}

showValue(new Impl());
```

Компиляция

```ts
class Impl {
  constructor() {
    this.myValue = "impl";
  }
}
function showValue(p) {
  if ("myValue" in p) {
    console.log(p.myValue);
  }
}

showValue(new Impl());
```

TypeScript 2.7

Ограничение на Union types

Так ли это плохо?

TypeScript 2.7

Ограничение на Union types

Так ли это плохо?

➢ Нужно изменить подход к написанию кода

TypeScript 2.8

Последняя на данный момент версия, вышедшая в 2018 году

TypeScript 2.8

- 20+ сложных типов
- ***instanceof*** / ***typeof*** type guard
- Discriminated union types и “in” guard
- User-defined type guard

TypeScript Next

Что дальше?

TypeScript Next

TypeScript 2.8: *Object.assign()*

TypeScript Next

TypeScript 2.8: *Object.assign()*

```
interface Object {
    assign<T, U>(target: T, source: U): T & U;

    assign<T, U, V>(target: T,
                    source1: U,
                    source2: V): T & U & V;

    assign<T, U, V, W>(target: T,
                       source1: U,
                       source2: V,
                       source3: W): T & U & V & W;

    assign(target: object, ...sources: any[]): any;
}
```

TypeScript Next

TypeScript 2.8: *Object.assign()*

```
interface Object {
    assign<T, U>(target: T, source: U): T & U;

    assign<T, U, V>(target: T,
                    source1: U,
                    source2: V): T & U & V;

    assign<T, U, V, W>(target: T,
                       source1: U,
                       source2: V,
                       source3: W): T & U & V & W;

    assign(target: object, ...sources: any[]): any;
}
```

Сдались

TypeScript Next

Что дальше?

- Еще больше типов!
- Новые type guard
- Nominal types?
- ES features

Ссылки

- Доклад Антона Лобова про типы в TypeScript: http://goo.gl/AX3PKc
- Примеры кода из презентации: https://github.com/anstarovoyt/evolution-typescript
- Доки про типизацию во Flow: https://flow.org/en/docs/lang/nominal-structural/
- TypeScripts Type System is Turing Complete: https://github.com/Microsoft/TypeScript/issues/14833
- TSConf keynote https://youtu.be/wpgKd-rwnMw
- TalkScript with the TypeScript Team https://youtu.be/MxB0ldQfvT4

Спасибо за внимание