Типизация

Status	Done
URL	

Проблемы javaScript
Что такое типы?
Вариант решения 1: языки с собственным синтаксисом
Вариант решения 2: языки, которые расширяют синтаксис JS
Typescript: преимущества и особенности
    Чем хорош?
    От чего не спасает?
    Как начать пользоваться?
Типы
    Примитивные типы
    Структуры
    Функции
Классы
Enum
Объединения
Пересечения
Специальные типа
    unknown **и never**
    any
Какие типы писать не надо?
Особенности
    Структурная типизация
Типы vs значения
Как происходит проверка типов
Пересечение и объединение объектов
Что делать, если нужного типа нет?
Как читать километровые ошибки?
Вариация конфигураций
Миграция

Проблемы javaScript

- Об ошибках типов узнаем в рантайме
- Неявное приведение типов (может приводить к неожиданным результатам)
- Сигнатуры функция приходится помнить

Что такое типы?

- Ограничения для переменных
- Допустимое множество значений
- Описание поведения сущности
- Пометка на значении

Вариант решения 1: языки с собственным синтаксисом

- Elm, reason, dart
- + Ошибка на этапе компиляции
- - Непривычный синтаксис
- Остались нишевыми

Вариант решения 2: языки, которые расширяют синтаксис JS

- Flow (от Facebook) - официально прекратил развитие
- TS (Microsoft) - наиболее актуальный (победивший) язык

Typescript: преимущества и особенности

- Предупреждает до запуска кода
- IDE подскажет, как исправить
- Говорящие сигнатуры функция (самодокументация)
- Важно: TS ничего не знает о рантайме
  - Все типы исчезают из конечного кода на этапе компиляции
  - Typescript не сможет больше, чем js

Чем хорош?

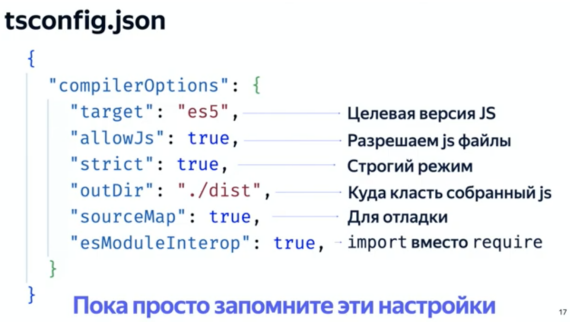
- Статическая типизация
- Разумные ограничения
- Низкий порог входа
- Совместимость с ks

От чего не спасает?

- Переопределение прототипов и глобальных объектов
- Вообще любой внешний код
- Причуды js
- Ошибки в бизнес логике

Как начать пользоваться?

```
npm i -D typescript //установка
npx tsc //запуск компилятора
```



Playground

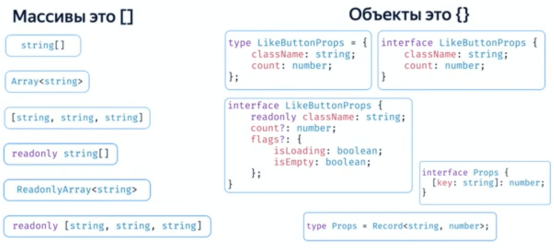
Типы

Примитивные типы

- string, number, bigint, boolean, undefined переносятся из js типов в ts типы без изменения
- null в js имеет тип object, в ts - отдельный типа null
- Объекты превращаются в структуры, function - в описанные функции
- Для каждого Symbol есть unique symbol

Структуры

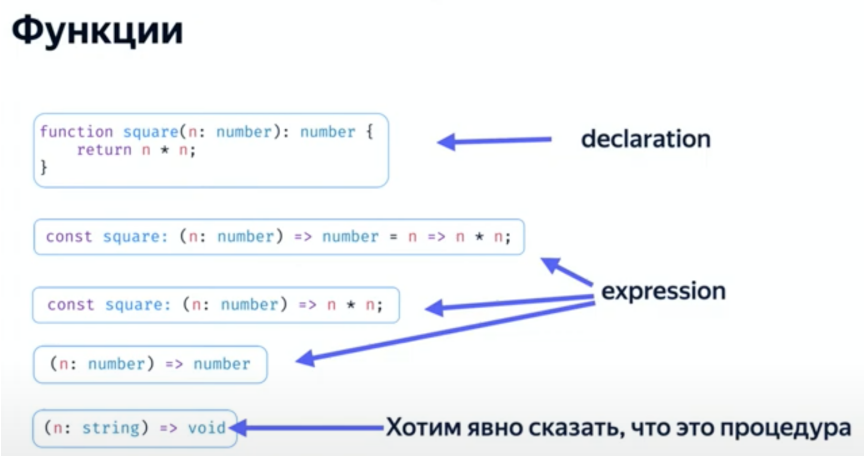
- Массивы
- Объекты



- type - когда тип нужно не дать расширить, interface - когда должно быть можно расширять

Функции

- Есть function declaration, есть function expression



- Возвращаем void чтобы явно показать, что это процедура

Классы

```
class Car {

  //тут тип number присвоится автоматом
  private speed = 0;
  public x = 0;

  //сокращенная запись this.name = name; this.weight = weight
  constructor (public name: string, private weight = 10){}

  public run(){
    this.speed += 100/this.weight;
  }

  public step(){
    this.x += this.speed;
  }
}

const  tesla = new Car('tesla', 5)
```

- Классы можно наследовать друг от друга с помощью extends
- Видимость полей может быть:
  - public - видно снаружи

    - private - видно только внутри класса
    - protected - видно только в наследниках
- Абстрактные классы и абстрактные методы
    - Заготовки для настоящих классов и методов. С их помощью нельзя создать реальные instances, но можно расширить реальными классами и методами и использовать их instances
- this задаем явно для сторонних функций

Enum

```
enum Color {
  Red, //0
  Green, //1
  Blue, //2
}

enum Color {
  Red = 'Red',
  Green = 'Green',
  Blue = 'Blue',
}

const bgColor = Color.Red;


const enum Color {
  Red = 'Red',
  Green = 'Green',
  Blue = 'Blue',
}
const bgColor = Color.Red;

//превращается в const bgcolor="Red"
```

Объединения

```
number|string //логическое ИЛИ

1|2|3|4|5 //литеральные типы - фиксируем одно значение
```

Пересечения

```
type LikeButtonProps = {
  className: string;
  count: number;
}

type PropsA = {
  className: string;
}

type PropsB = {
  count: number;
}
//логическое И
PropsA & PropsB //LikeButtonProps
```

Специальные типа

unknown и never

- unknown - значение не известно
- never - значение не существует

```
function main(text: unknown) {
  if (typeof text === 'number'){
    console.log(text, 'number')

    //(paramets) text:never
  }
}
```

any

- Значение может быть любым

- Игнорирует любые проверки

```
function main(text: string) {
  if (typeof text === 'number'){
    console.log(text, 'number')
  }
}
const text1 = 2023;
main(text1) //будет ошибка

const text2: any = 2023
main(text2) //ошибки не будет
```

- лучше не использовать any

Какие типы писать не надо?

- Все, что есть в ECMAScript (Map, Set, ArrayBuffer)
- Типы для браузера и nodejs (console, fetch, fs)
- Те, что можно достать из библиотек
- Тьюринг полные типы, если только вы не пишите шахматы на типах

Особенности

Структурная типизация

Не важно, как что называется, важно, как себя ведет

```
class A {
  constructor(x: number) {}
}

class B {
  constructor(x: number) {}
}

const a: A = new B(10)
// ^? const a: A
```

```
const b = new B(10)
// ^? const b: B
```

```
//Эмуляция уникальных классов
class A {
  readonly slug = 'A'
  constructor(x: number){}
}

class B {
  readonly slug = 'B'
  constructor(x: number){}
}

const a: A = new B(10) //Types of property 'slug' are incompatible
```

- Исключение - для enum и символов типизация номинативная

```
const enum A {R, G, B}
const enum B {R, G, B}

const color1: A = A.R
const color: A = B.R //type 'B.R' is not assignable to type 'A'
```

```
const symbol1 = Symbol();
const symbol2: unique symbol = symbol1 //typeof symbol1 is not assignable to type 'typ
eof symbol2'
```

```
const symbol0 = Symbol.for('0');
const symbol1 = Symbol.for('0');

const symbol2: typeof symbol0 = symbol1;//typeof symbol1 is not assignable to type 'ty
peof symbol2'
```

Типы vs значения

- enum - синтаксически единственная сущность в двух мирах
- Вывод типов возможен из значений, но не наоборот
- Валидация работает только в рантайме

Как происходит проверка типов

- При присваивании или вызове функции тип должен быть подтипом требуемого

```
type Primitive = string | number | boolean;
declare const a: number | string; //declare - где-то есть, неважно где

const c: Primitive = a;


declare const b: number | number[];
const d: Primitive = b; //type number[] is not assignable to type 'Primitive'
```

- Для объектных литералов дополнительно проверяются поля объекта

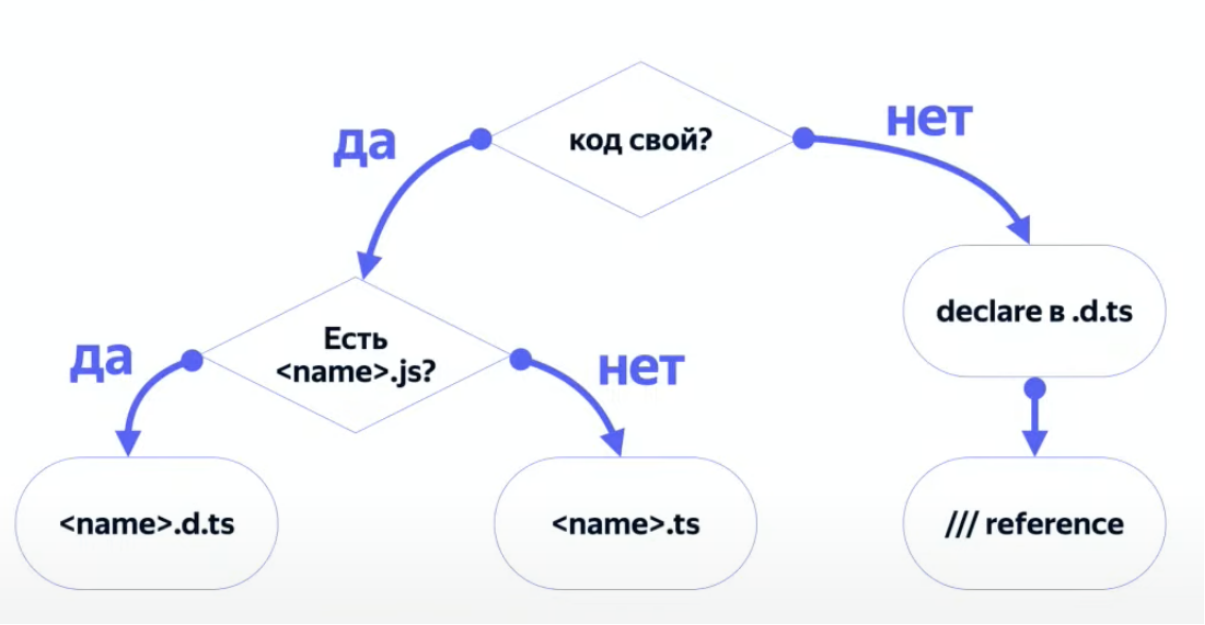
```
interface Options {
  isEmpty?: boolean;
  text?: string;
}

function getValue(options: Options){
  if (options.isEmpty){
    return null;
  }
  return options.text ?? '';
}

getValue({isEmpty: true});
getValue({text: 'shri'});
getValue({isEmpty: false, text: 'shri', empty: true});//будет ругаться на empty
```

Пересечение и объединение объектов

```
interface EventItem { name: string; timestamp: number; }
interface Step { name: string; deadline: number; }
interface Timeline { events: EventItem[]; roadmap: Step[]; }


function add(timeline: Timeline, valueL EventItem & Step){
  if ('timestamp' in value){
    timeline.events.push(value);
    // ^?(parameter) value: EventItem & Step
  } else {
    timeline.roadmap.push(value);
    // ^?(parameter) value: never
  }
}

const timeline: Timeline = { events: [], roadmap: []};
add(timeline, {name: 'start', timestamp: Date.now(0)}//property 'deadline' is missing
```

```
interface EventItem { name: string; timestamp: number; }
interface Step { name: string; deadline: number; }
interface Timeline { events: EventItem[]; roadmap: Step[]; }


function add(timeline: Timeline, valueL EventItem | Step){
  if ('timestamp' in value){
    timeline.events.push(value);
    // ^?(parameter) value: EventItem
  } else {
    timeline.roadmap.push(value);
    // ^?(parameter) value: Step
  }
}

const timeline: Timeline = { events: [], roadmap: []};
add(timeline, {name: 'start', timestamp: Date.now(0)}
```

Что делать, если нужного типа нет?



Как читать километровые ошибки?

- Читать текст ошибки
- В глубокой вложенности важна последняя часть
- Для вложенных объектов попробовать вывести промежуточный типа, используя дополнительный алиас (type t =)

Вариация конфигураций

- Конфиги ts наследуются
- Тулинг обычно настроен на tsconfig.json, но можно указать свой или переопределять настройки на лету
- После мирации с js желательно ужесточить конфиг, запретив js файлы
- Можно настроить кастомный jsx

Миграция

Миграция