## 1. Главная задача TypeScript?

Надстройка над js, которая позволяет устранить проблемы JS, такие как: типизация, сложный для отладки код. TypeScript помогает избавиться от типичных проблем JavaScript: ошибок типов в рантайме и неконтролируемо разрастающегося кода, сигнатуры функций которого находятся в лучшем случае в памяти разработчика, а в худшем и вовсе утрачены.

## 2. Может ли TypeScript исполняться в браузере?

Нет. Не может. Его нужно скомпилировать командой tsc в js.

## 3. Объясните процесс понижения уровня кода.

Понижение уровня кода осуществляется за счет упрощения написания кода программистом и скрытия от разработчика многих внутренних (системных) алгоритмов.

## 4. Какие настройки предоставляет TypeScript для управления уровнем строгости проверок?

tsconfig.json, установите флаг strict в секции compilerOptions в значение true

Из них две секции: Strict Checks и Linter Checks – содержат только опции тех самых флагов строгости. Помимо ещё часть интересующих нас сегодня флагов сокрыта в самой большой группе опций Advanced.

Группа Strict Checks. Пожалуй, флаги данной категории наиболее важные из всех. Вот они: strict, alwaysStrict, noImplicitAny, strictNullChecks, strictFunctionTypes, strictPropertyInitialization, noImplicitThis, strictBindCallApply.

## 5. Перечислите базовые типы данных, используемые в TypeScript.

В TypeScript имеются следующие базовые типы:

* **boolean**: логическое значение true или false
* **number**: числовое значение
* **string**: строки

другие типы:

* **Array**: массивы
* **кортежи**
* **Enum**: перечисления
* **Any**: произвольный тип
* **Symbol**
* **null и undefined**: соответствуют значениям null и undefined в javascript
* **Never**: также представляет отсутствие значения и используется в качестве возвращаемого типа функций, которые генерируют или возвращают ошибку

https://metanit.com/web/typescript/2.5.php

## 6. Каким образом в TypeScript аннотируются массивы?

Аннотирование нужно, чтоб определить статический тип данных в массивах и функциях.

Чтобы аннотировать тип массива, который вы используете, используйте определенный тип, за которым следует квадратная скобка : type[] :

let arrayName: type[];

Используйте аннотации TypeScript типа с синтаксисом : [type] , чтобы явно указать тип для переменной, функции, возвращаемого значения функции и т. д.

Источник: https://codezen.ru/osnovnoe-rukovodstvo-po-annotatsiyam-v-typescript

## 7. Что в TypeScript называется кортежем?

Кортежи (Tuples) также, как и массивы, представляют набор элементов, для которых уже заранее известен тип. В отличие от массивов кортежи могут хранить значения разных типов. Для определения кортежа применяется синтаксис массива:

// определение кортежа - кортеж состоит из двух элементов - строки и числа

let user: [string, number];

https://metanit.com/web/typescript/2.10.php

## 8. Как аннотировать функции с помощью TypeScript? Всегда ли это необходимо?

greeting = function (name: string) {

return `Hi ${name}`;

};

Используйте аннотации TypeScript типа с синтаксисом : [type] , чтобы явно указать тип для переменной, функции, возвращаемого значения функции и т. д.

Источник: <https://codezen.ru/osnovnoe-rukovodstvo-po-annotatsiyam-v-typescript>

## 9. Что в TypeScript называется объединением? Привести примеры использования.

Объединения или union не являются собственно типом данных, но они позволяют комбинировать или объединить другие типы. Так, с помощью объединений можно определить переменную, которая может хранить значение двух или более типов:

|  |  |
| --- | --- |
| 1  2  3  4  5 | let id : number | string;  id = "1345dgg5";  console.log(id); // 1345dgg5  id = 234;  console.log(id);  // 234 |

https://metanit.com/web/typescript/2.13.php

## 10. Объяснить псевдонимы типов, интерфейсы и их различия.

TypeScript позволяет определять псевдонимы типов с помощью ключевого слова **type**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | type id = number | string;    let userId : id = 2;  console.log(`Id: ${userId}`);  userId = "qwerty";  console.log(`Id: ${userId}`); |

Здесь для объединения number|string определяется псевдоним id. Далее мы можем использовать этот псевдоним для определения переменных.

**Расширение псевдонимов**

Одни псевдонимы могут заимствовать или расширять код других. Для этого применяется операция **&**. Например:

|  |  |
| --- | --- |
| 1  2 | type Person = {name: string; age: number};  type Employee = Person & {company: string}; |

В данном случае псевдоним Employee расширяет псевдоним Person, добавляя к нему свойство company, которое представляет тип string. То есть фактически мы имеем дело с типом:

https://metanit.com/web/typescript/2.14.php

## 11. Объяснить утверждения типов. Как производить некорректные утверждения типов?

Утверждение типов - общее[¶](https://scriptdev.ru/guide/035/#-)

При разработке приложений на языках со статической типизацией время от времени может возникнуть нестыковка из-за несоответствия типов. Простыми словами, приходится работать с объектом, принадлежащим к известному типу, но ограниченному более специализированным (менее конкретным) интерфейсом.

Утверждение типа синтаксис[¶](https://scriptdev.ru/guide/035/#_2)

Одним из способов указать компилятору на принадлежность значения к заданному типу является механизм утверждения типа при помощи угловых скобок <ConcreteType>,

## 12. Какие типы в TypeScript называются литеральными? Привести примеры использования.

Литералы - это *фиксированные* значения, которые являются примитивами JavaScript.

Вы можете использовать строковый литерал в качестве типа. Например:

|  |  |
| --- | --- |
|  | let foo: 'Hello'; |

Они не очень полезны сами по себе, но могут быть собраны в тип объединение для создания мощной (и полезной) абстракции, например:

type CardinalDirection =

| 'North'

| 'East'

| 'South'

| 'West';

function move(

distance: number,

direction: CardinalDirection

) {

// ...

}

move(1, 'North'); // Okay

move(1, 'Nurth'); // Ошибка!

## 13. Объяснить сужение типов. Способы сужения типов. Привести примеры.

Процесс приведения определенного типа к более конкретной версии с помощью защитников типа и присвоений называется сужением типа (narrowing).

В JS существует оператор для определения наличия указанного свойства в объекте - оператор in.

Для сужения типов также можно воспользоваться инструкцией switch или операторами равенства ===, !==, ==, !=, например:

function printAll(strs: string | string[] | null) {

if (strs !== null) {

if (typeof strs === 'object') {

for (const s of strs) {

// (parameter) strs: string[]

console.log(s)

}

} else if (typeof strs === 'string') {

console.log(strs)

// (parameter) strs: string

}

}

}

TS позволяет использовать данный оператор для сужения поетнциальных типов.

'value' in x, где 'value' - строка, а x - объединение, истинная ветка сужает типы x к типам

Рассмотрим интересный пример:

function printAll(strs: string | string[] | null) {

if (typeof strs === "object") {

for (const s of strs) {

// Object is possibly 'null'. Потенциальным значением объекта является 'null'

console.log(s);

}

} else if (typeof strs === "string") {

console.log(strs);

} else {

// ...

}

}

В функции printAll мы пытаемся проверить, является ли переменная strs объектом (массивом). Но, поскольку выражение typeof null возвращает object (по историческим причинам), мы получаем ошибку.

Таким образом, в приведенном примере мы выполнили сужение к string[] | null вместо желаемого string[].

Существует несколько различных конструкций, которые применяются в TypeScript для сужения:

* Защитник типов typeof
* Проверка на истинность (Truthiness narrowing)
* Сужение при проверке на равенство (Equality narrowing)
* Сужение типов с оператором in
* Сужение типов с instanceof
* Присваивания (Assignments)
* Анализ потока управления
* Использование предикатов типа (type predicates)
* Исключающие объединения (discriminated unions)

<https://it-dev-journal.ru/articles/suzhenie-tipov>

## 14. Что называется сигнатурой вызова? Конструкторской сигнатурой?

В JS функции, кроме того, что являются вызываемыми (callable), могут иметь свойства. Однако, тип-выражение не позволяет определять свойства функции. Для описания вызываемой сущности (entity), обладающей некоторыми свойствами, можно использовать сигнатуру вызова (call signature) в объектном типе:

type DescFn = {  
 description: string  
 (someArg: number): boolean  
}  
**function** doSomething(fn: DescFn) {  
 console.log(`Значением, возвращаемым ${fn.description} является ${fn(6)}`)  
}

**Сигнатуры конструктора (construct signatures)**

Как известно, функции могут вызываться с ключевым словом new. TS считает такие функции конструкторами, поскольку они, как правило, используются для создания объектов. Для определения типов таких функций используется сигнатура конструктора:

type SomeConstructor = {  
 new (s: string): SomeObject  
}  
**function** fn(ctor: SomeConstructor) {  
 **return** **new** ctor('Hello!')  
}

## 15. Объяснить концепцию перегрузки функций в TypeScript. Привести пример.

Перегрузка функций в TypeScript — это возможность создания нескольких методов с одним и тем же именем, но разными типами или количеством параметров. Компилятор использует количество, типы и порядок аргументов для определения того, какой метод следует вызвать.

**function** makeDate(timestamp: **number**): **Date**;

Потом описывается тип функции с тремя параметрами:

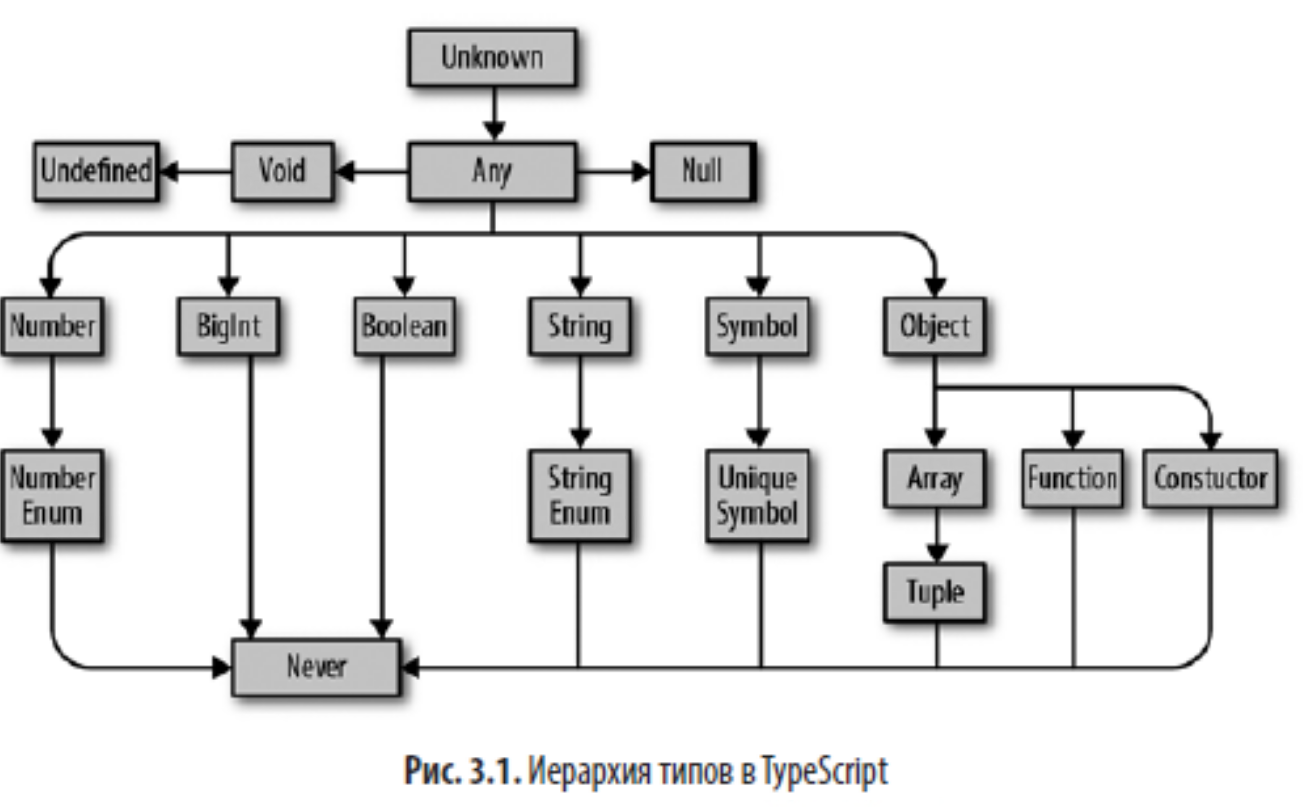
**function** makeDate(m: **number**, d: **number**, y: **number**): **Date**;

После этого идет реализация функции, которая объединяет два типа, описанных выше:

**function** makeDate(mOrTimestamp: **number**, d?: **number**, y?: **number**): **Date** {  
 **if** (d !== undefined && y !== undefined) {  
 **return** **new** Date(y, mOrTimestamp, d);  
 } **else** {  
 **return** **new** Date(mOrTimestamp);  
 }  
}

## 16. Что в TypeScript называется объектным типом?

Object (object) — ссылочный объектный тип. Ссылочный тип данных Object является базовым для всех ссылочных типов в TypeScript. Помимо того, что в TypeScript существует объектный тип Object , представляющий одноименный конструктор из JavaScript, также существует тип object, представляющий собой любое объектное значение.



## 17. Как TypeScript подходит к сравнению типов?

Классы в TypeScript являются одновременно значением и типом данных. Второе особенно важно в контексте типизации функций и методов. Такое поведение обусловлено структурной типизацией. При сравнении типов TypeScript сравнивает структуру, а не имена.

## 18. Перечислите модификаторы свойств в TypeScript, назовите их особенности.

В TypeScript три модификатора: public, protected и private.

Если к свойствам и функциям классов не применяется модификатор, то такие свойства и функции расцениваются как будто они определены с модификатором public

Если же к свойствам и методам применяется модификатор private, то к ним нельзя будет обратиться извне при создании объекта данного класса.

Модификатор protected во многом аналогичен private - свойства и методы с данным модификатором не видны из вне, но к ним можно обратиться из классов-наследников:

class User {

private \_name: string;

public get name(): string {

return this.\_name;

}

public set name(n: string) {

this.\_name = n;

}

}

let tom = new User();

tom.name = "Tom"; // срабатывает set-метод

console.log(tom.name); // срабатывает get-метод

## 19. Объяснить сигнатуры индексов. Привести примеры использования.

Динамические ключи (Index Signature) синтаксис для динамических ключей.

При создании объектов мы можем использовать разные значения в качестве ключей. Среди значений могут быть строки, числа или символы. Ровно такие же возможности TypeScript накладывает на свои объектные типы

type dynamicKeysObject = {

[key: string | number | symbol]: unknown;

};

const obj: dynamicKeysObject = {

name: 'John',

age: 30,

0: 'zero',

[Symbol('secret')]: 'symbol',

};

## 20. Назовите особенности использования классов в TypeScript.

Классы в TypeScript являются одновременно значением и типом данных. Второе нам особенно важно в контексте типизации функций и методов. Такое поведение обусловлено структурной типизацией. При сравнении типов TypeScript сравнивает их структуру, а не имена.

class User {

    readonly name: string = "Default user";

    age: number;

    constructor(userName: string, userAge: number) {

        this.name = userName;

        this.age = userAge;

    }

    print(){

        console.log(`name: ${this.name}  age: ${this.age}`);

    }

}

let tom = new User("Tom", 36);

tom.name = "Bob";       // ! Ошибка - поле name - только для чтения

tom.print();                    // name: Tom  age: 36

## 21. Объясните ключевые слова extends и implements в TypeScript. Назовите их различия.

Различие между implements и extends довольно простое: extends используется для наследования класса, а implements — для реализации интерфейса. Отношения между классами и интерфейсами определяют, когда что использовать.

## 22. Перечислите области видимости в TypeScript и назовите их особенности.

В TypeScript (и ES6) добавили ключевое слово let, которое позволяет

объявлять переменные с настоящей блочной областью видимости.

Блочная **область видимости** - переменная (let) может быть объявлена с именем переменной с областью видимости на всю функцию, если она объявлена в другом блоке.

* Локальная **область видимости** — идентификатор доступен только внутри определённой функции (процедуры).

## 23. Объяснить концепцию обобщенных типов (обобщение, дженерик) в TypeScript. Привести примеры.

Обобщённый тип (обобщение, дженерик) позволяет резервировать место для типа, который будет заменён на конкретный, переданный пользователем, при вызове функции или метода, а также при работе с классами.

Самое часто используемое в TypeScript обобщение – Promise<T>. Например, это обещание, возвращающее строку function a(): Promise<string> { return 'a' }.

function identity<T>(arg: T): T {

return arg;

}

const value = identity<number>(115);

**Общие функции или функции-дженерики (generic functions)**

Часто тип данных, возвращаемых функцией, зависит от типа передаваемого функции аргумента или же два типа возвращаемых функцией значений зависят друг от друга. Рассмотрим функцию, возвращающую первый элемент массива:

**function** firstElement(arr: any[]) {  
 **return** arr[0]  
}

## 24. Что в TypeScript называется ограничением обобщенного типа? Привести примеры.

Ограничения позволяют сузить перечень типов, которые можно будет передать в обобщённом аргументе, и расширяет возможности по работе с этими типами.

В нашем случае мы хотим ограничить принимаемое множество типов T условием: наличием свойства length. Для этого нужно создать некоторый интерфейс, где указано нужное свойство и расширить его, используя обобщённый тип T.

interface Lengthwise {

length: number;

}

function getLength<T extends Lengthwise>(arg: T): number {

return arg.length;

}