--init создает tsconfig.json

"exclude": ["./module.ts"],

массив из строк, который сообщает компилятору какие пути надо исключить из компиляции ts

"include": ["./src/\*\*/\*"]

Включить в компиляцию все, что лежит в папке src

"files": ["./module.ts"]

Только модуль

Разбор compilerOptions

"outDir": "./dist",

Складываем все в папку dist

 "rootDir": "./src",

Корневая директория

Tsc --watch автоматическая сборка

"lib": ["DOM"],

Подключаем только DOM api, подробнее <https://www.typescriptlang.org/tsconfig>

"allowJs": true,

"checkJs": true,

Если мы хотим писать больше на js, чем на ts

  "jsx": "react",

Jsx в разработке на реакт

"sourceMap": true,

Генерирует соурс мэпы. Позволяет смотреть исходники в браузере.

"noEmitOnError": false,

Допустим у нас есть ошибка в ts, то при tsc файл все равно будет скомпилирован, однако если true, то код не будет скомпилирован.

"strict": true,

Включает по умолчанию се или же мы можем сами настраивать ошибки. Например:

"noImplicitAny": true,

Идея в том, что если ts не понимает, что за тип в функции например function(data), тогда нужно выдавать ошибку. Короче лучше юзать стрикт мод

**Generic Types.**

// generic

const cars2: Array<string> = ['Ford', 'Audi']

с помощью них мы сможем явно указать ts, что в итоговом объекте у нас будут присутствовать наши ключи, и мы сможем работать с авто комплитом.

function mergeObjects<T, R>(a: T, b: R): T & R {

  return Object.assign({}, a, b)

}

const merged = mergeObjects({ name: 'Rodion' }, { age: 26 })

const merged2 = mergeObjects({ name: 2 }, { age: '26' })

console.log(merged.name)

console.log(merged2.age);

типы T и R являются generic и они подстраиваются под входящие данные.

Как это было бы без generic:

function mergeObjects<T, R>({name: string}, b: R): T & R {

  return Object.assign({}, a, b)

}

Че то такого.

Но могут быть проблемы

const merged3 = mergeObjects('aaa', 'bbb')

console.log(merged3);

мы должны понимать, что ф-ция mergeObjects работает только с объектами, в отличии от строк.

И мы можем явно указывать ограничения:

function mergeObjects<T extends object, R extends object>(a: T, b: R): T & R {

  return Object.assign({}, a, b)

}

**След.**

function withCount(value) {

  return {

    value,

    count: `В этом объекте ${value.length} символов`,

  }

}

Не у каждого входящего параметра может быть ключ length. Добавим generic тип

function withCount<T>(value: T): {value: T, count: string} {

  return {

    value,

    count: `В этом объекте ${value.length} символов`,

  }

}

Возвращает объект, у которого value = типу T, и count у которого тип string. Теперь другая проблема, поле length не факт что присутствует в типе T. И нам нужно явно это указать. Для этого мы создадим локальный интерфейс.

interface ILength{

    length: number

}

И для того, чтобы указать что наша функция работает с generic типом, но которая также включает себя интерфейс ключ length, мы просто наследуем.

function withCount<T extends ILength>(value: T): {value: T, count: string} {

  return {

    value,

    count: `В этом объекте ${value.length} символов`,

  }

}

console.log(withCount('Привет TS'));

console.log(withCount(['I', 'am', 'array']));

еще {length: 20}

function getObjectValue(obj: object, key: string){

    return obj[key]

}

Не любая строка является ключом у объекта. Есть ли такой ключ или нет:

const person = {

    name: 'Rodion',

    age: 26

}

console.log(getObjectValue(person, 'name'));

console.log(getObjectValue(person, 'job'));

Как избавится от этой ошибки. Generic тип R должен наследоваться от ключей данного объекта T, зависеть от них, для этого используем keyof

function getObjectValue<T extends object, R extends keyof T>(obj: T, key: R){

    return obj[key]

}

Теперь все правильно

**Работа с классами.**

class Collection<T> {

  constructor(private \_items: T[] = []) {}

  add(item: T) {

    this.\_items.push(item)

  }

  remove(item: T) {

    this.\_items = this.\_items.filter((i) => i !== item)

  }

  get items(): T[] {

    return this.\_items

  }

}

Проблема, если мы хотим создать коллекцию из объектов [{a: 1}, {b: 2}], это будет работать, но если мы хотим удалить объект {b: 2}

const objs = new Collection<object>([{ a: 1, b: 2 }])

objs.remove({ b: 2 })

console.log(objs.items)

то в итоге там все равно будет два элемента, поскольку разные ссылки. Это явная ошибка, мы можем сказать коллекции, что она работает исключительно с примитивными типами данных:

class Collection<T extends number | string | boolean> {

**след**

interface Car {

  model: string

  year: number

}

function createAndValidateCar(model: string, year: number): Car {

  const car: Car = {}

  if (model.length > 3) {

    car.model = model

  }

  if (year > 2000) {

    car.year = year

  }

  return car

}

Сейчас у нас есть ошибка, когда мы создаем объект car как пустой объект, в нем не хватает определенных полей, решение: Partial. Те мы временно создаем объект и в нем временно не хватает определенных ключей**, ts не суетись.**

function createAndValidateCar(model: string, year: number): Car {

  const car: Partial<Car> = {}

  if (model.length > 3) {

    car.model = model

  }

  if (year > 2000) {

    car.year = year

  }

  return car as Car

}

**Decorators**

Паттерн проектирования, синтаксический сахар. Есть 4 типа.

Мы можем добавлять это для класса, для свойств, для методов,