



1099,00

Рейтинг

RUVDS.com

RUVDS – хостинг VDS/VPS серверов



ru_vds 12 декабря 2017 в 14:08

Основы TypeScript, необходимые для разработки Angular-приложений

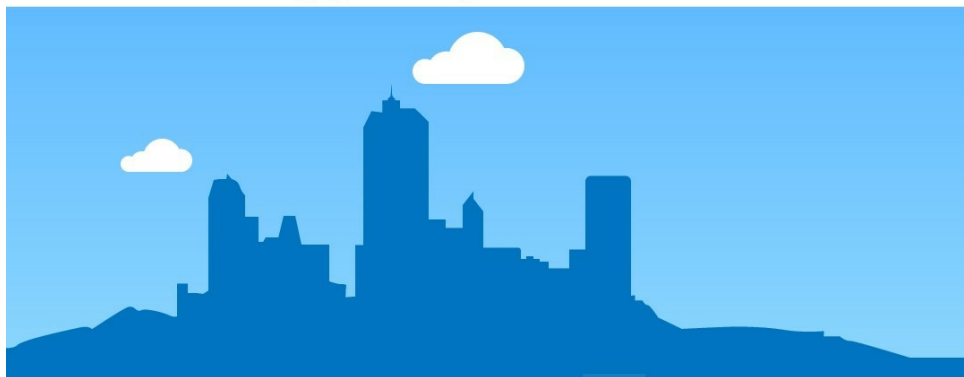
Автор оригинала: Abel Agoal

Блог компании RUVDS.com, Angular, JavaScript, TypeScript, Разработка веб-сайтов

Перевод

TypeScript — это надмножество JavaScript, то есть, любой код на JS является правильным с точки зрения TypeScript. Однако, TypeScript обладает некоторыми дополнительными возможностями, которые не входят в JavaScript. Среди них — строгая типизация (то есть, указание типа переменной при её объявлении, что позволяет сделать поведение кода более предсказуемым и упростить отладку), механизмы объектно-ориентированного программирования и многое другое. Браузеры не поддерживают TypeScript напрямую, поэтому код на TS надо транспилировать в JavaScript.

TypeScript



TypeScript применяется для разработки веб-приложений с использованием популярного фреймворка Angular. В этом материале мы рассмотрим основы TypeScript, необходимые для того, чтобы приступить к освоению Angular и к работе с ним.

Предварительная подготовка

Прежде чем пользоваться TypeScript, его надо установить:

```
sudo npm install -g typescript
```

Теперь можно переходить к изучению возможностей языка. Откройте текстовый редактор и создайте

ИНФОРМАЦИЯ

Дата основания	27 июля 2015 г.
Локация	Москва Россия
Сайт	ruvds.com
Численность	11–30 человек
Дата регистрации	18 марта 2016 г.

ССЫЛКИ

[VPS / VDS сервер за 130 руб/месяц](#)
ruvds.com

Дата-центры RUVDS в Москве, Санкт-Петербурге, Казани, Цюрихе, Лондоне
ruvds.com

Помощь и вопросы
ruvds.com

Партнерская программа RUVDS
ruvds.com

VPS (CPU 1x2ГГц, RAM 512Мб, SSD 10 Gb) — 190 рублей в месяц
ruvds.com

VPS Windows от 293 рублей.
Бесплатный тестовый период 3 дня.
ruvds.com

Все клиенты RUVDS застрахованы AIG
ruvds.com

[VDS в Швейцарии](#)
ruvds.com

Антивирусная защита виртуального сервера. Легкий агент для VPS.
ruvds.com

[VPS в Лондоне](#)
ruvds.com

ВИДЖЕТ

новый файл. Поместите в него следующий код:

```
function log(message) {  
  console(message);  
}  
let message = 'Hello Typescript';  
log(message);
```

Сохраните его как `main.ts` и, перейдя в терминале туда, где его сохранили, выполните следующую команду:

```
tsc main.ts
```

Данная команда создаёт новый файл, `main.js`, который является транспирированной JS-версией файла `main.ts`. Этот файл можно, например, выполнить с помощью Node.js:

```
node main.js
```

Обратите внимание на то, что вам не нужно выполнять команду `tsc` при сборке Angular-приложения, так как инструмент `ng serve` подготовит код к выполнению автоматически.

Типы данных в TypeScript

TypeScript поддерживает различные типы данных. Среди них можно отметить следующие:

```
let a: number    //например: 1, 2, 3  
let b: boolean   //например: true, false  
let c: string    //например: "abel agoi"  
let d: any       //такая переменная может содержать значения любых других типов  
let e: number[]  //числовой массив, например: [1, 3, 54]  
let f: any[]     //массив значений любых типов, например: [1, "abel agoi", true]
```

Обратите внимание на то, что TypeScript поддерживает ещё один тип, `enum`, о нём вы можете почитать самостоятельно.

Стрелочные функции

В JavaScript функции объявляют так:

```
let log = function (message) {  
  console.dir(message);  
}
```

В TypeScript того же эффекта можно добиться с помощью стрелочных функций, при объявлении которых используется последовательность символов `=>`. Вот как это выглядит:

```
let log = (message) => { //тут мы просто убираем слово function  
  console.dir(message);  
}  
//это можно сократить, приведя к следующему виду  
let log = (message) => console.dir(message);  
//а если функции передаётся всего один параметр, можно записать её ещё короче  
let log = message => console.dir(message); //однако, такой код плохо читается
```

Интерфейсы

Не рекомендуется писать функции, которым надо передавать очень много параметров. Например, это может выглядеть так:



ВИДЖЕТ

Виртуальный сервер
за **240Р** в месяц

CPU 2.2 ГГц
RAM 1 Гб
HDD 20 Гб

БЛОГ НА ХАБРЕ

Прыжки из стратосферы

1,4k 0

Что исследуют в стратосфере?

7,4k 17

Учебный курс по React, часть 26:
архитектура приложений, паттерн
Container/Component

3,5k 0

Йо-хо-хо и бутылка рому

6,1k 0

Сравнение систем космической связи

8,5k 12

Подробности о GraphQL: что, как и
почему

11,8k 23

Как согласовать полёт зонда в
стратосферу (с чем мы столкнёмся на
практике при запуске)

19,5k 100

```
let myFunction = ({
  a: number,
  b: number,
  c: string,
  d: any,
  e: number[],
  f: any[]
}) => {
  console.log('something');
}
```

Избегать таких конструкций можно, включив параметры в объект и передав функции единственный объект. Тут нам помогут интерфейсы. Этот механизм есть в TypeScript:

```
interface MyParameters {
  a: number,
  b: number,
  c: string,
  d: any,
  e: number[],
  f: any[]
  ...
  ...
}

let myFunction = (myParameters: MyParameters) {
}
```

Классы

Стоит выработать у себя привычку группировать связанные переменные (свойства) и функции (методы) в единую конструкцию, которая в программировании называется классом. Для того, чтобы опробовать это на практике, создайте файл `myPoints.ts` и поместите в него следующий код:

```
class MyPoint {

  x: number;
  y: string;
  draw() {
    //обратите внимание на то, что со свойством x мы работаем через this
    console.log("X is: " + this.x);
    console.log("X is: " + this.y);
  }
  getDistanceBtw(another: AnotherPoint) {
    //посчитать и вернуть расстояние
  }
}
```

Доступ к свойствам и методам классов

Мы сгруппировали связанные переменные и методы в единый класс. Теперь надо разобраться с тем, как с ними работать. Для этого нужно создать экземпляр класса:

```
let myPoint = new MyPoint() //MyPoint - это имя класса
myPoint.x = 2;               //устанавливаем свойство x
myPoint.y = "a";             //устанавливаем свойство y

myPoint.draw();              //вызываем метод draw
```

Файл `myPoints.ts` можно транспилировать и запустить то, что получилось:

```
tsc myPoint.ts | node myPoint.js
```

Обратите внимание на то, что прежде чем назначать свойствам класса значения, нужно создать его экземпляр. А есть ли способ задания значений свойств в ходе создания экземпляра класса? Такой способ есть и существует он благодаря конструкторам.

Учебный курс по React, часть 25:
практикум по работе с формами

3,6k

0

Когда «Zoë» != «Zoё», или почему
нужно нормализовывать Unicode-
строки

15,1k

37

JavaScript — лучший язык
программирования для начинающих.
Так это или нет?

23,4k

97

Конструкторы

Конструктор — это метод, который вызывается автоматически при создании экземпляров класса.

Конструктор позволяет задавать значения свойств. Вот пример работы с экземпляром класса, в котором возможности конструкторов не применяются:

```
let myPoint = new MyPoint()
myPoint.x = 2;
myPoint.y = "a";
```

То же самое, с использованием конструктора, можно переписать так:

```
let myPoint = new MyPoint(2, "a");
```

Однако, для того, чтобы вышеприведённый пример заработал, понадобится внести изменения в класс, задав его конструктор:

```
class MyPoint {
  x: number;
  y: string;
  constructor (x: number, y: string) {
    this.x = x;
    this.y = y;
  }
  draw() {
    //обратите внимание на то, что со свойством x мы работаем через this
    console.log("X is: " + this.x);
    console.log("X is: " + this.y);
  }
  getDistanceBtw(another: AnotherPoint) {
    //посчитать и вернуть расстояние
  }
}
```

Необязательные параметры в конструкторе

Что если мы решили использовать конструктор, но хотим, чтобы явное задание параметров при создании экземпляров класса было бы необязательно? Это возможно. Для этого надо использовать знак вопроса (?) в конструкторе. Этот знак позволяет определять параметры, которые, при создании экземпляра класса, задавать необязательно:

```
class MyPoint {
  x: number;
  y: string;
  constructor (x?: number, y?: string) { //обратите внимание на "?" перед ":"
    this.x = x;
    this.y = y;
  }
  draw() {
    //обратите внимание на то, что со свойством x мы работаем через this
    console.log("X is: " + this.x);
    console.log("X is: " + this.y);
  }
  getDistanceBtw(another: AnotherPoint) {
    //посчитать и вернуть расстояние
  }
}

//Этот код нормально отработает при создании экземпляра myPointA класса MyPoint
let myPointA = new MyPoint()
myPointA.x = 2;
myPointA.y = "a";
myPointA.draw();
```

```
//Этот код нормально отработает при создании экземпляра myPointB класса MyPoint
let myPointB = new MyPoint(2, "b");
myPointB.draw();
//Этот код нормально отработает при создании экземпляра myPointC класса MyPoint
let myPointC = new MyPoint(2); //обратите внимание на то, что значение Y мы тут не передаём
myPointC.draw();
```

Модификаторы доступа

Модификатор доступа — это ключевое слово, которое используется со свойством или членом класса для управления доступом к нему извне. В TypeScript есть три модификатора доступа: `public`, `protected` и `private`. По умолчанию все члены класса общедоступны — это аналогично использованию с ними модификатора доступа `public`, то есть, читать и модифицировать их можно извне. Использование модификатора доступа `private` позволяет запретить внешним механизмам работу с членами класса. Например, здесь мы использовали данный модификатор со свойствами `x` и `y`:

```
class MyPoint {
  ...

  private x: number;
  private y: string;
  //модификаторы доступа можно устанавливать и для методов
  public draw() {
    //нарисовать что-нибудь
  }
  ...
}
let myPoint = new MyPoint();
myPoint.x = 3;
```

Попытка использовать конструкцию `myPoint.x` при работе с экземпляром класса приведёт к ошибке, так как свойство объявлено с модификатором доступа `private`.

Вспомогательные средства конструкторов

Выше мы добавляли конструктор к нашему классу следующим образом:

```
private x: number;
public y: string;
constructor (x: number, y: string) {
  this.x = x;
  this.y = y;
}
```

TypeScript позволяет записать то же самое в сокращённой форме:

```
constructor (private x: number, public y: string) {}
```

Всё остальное будет сделано автоматически (готов поспорить, вы часто будете с этим встречаться в Angular-приложениях). То есть, нам не нужен следующий код:

```
private x: number;
public y: string;
и
this.x = x;
this.y = y;
```

Геттеры и сеттеры

Предположим, что сейчас класс `MyPoint` выглядит так:

```
class MyPoint {
  constructor (private x?: number, private y?: string) {}
  draw() {
```

```

    //нарисовать что-нибудь
  }
  drawAnotherThing() {
    //нарисовать ещё что-нибудь
  }
}

```

Мы совершенно точно знаем, что не сможем работать со свойствами `x` и `y` за пределами экземпляра класса `MyPoint`, так как они объявлены с использованием модификатора доступа `private`. Если же нужно как-то на них влиять или читать их значения извне, нам понадобится использовать геттеры (для чтения свойств) и сеттеры (для их модификации). Вот как это выглядит:

```

class MyPoint {
  constructor (private x?: number, private y?: string) {}
  getX() { //возвращает X
    return this.x;
  }
  setX(value) { //записывает в X переданное методу значение
    this.x = value;
  }
}
//так как установить x напрямую нельзя, после инициализации экземпляра класса myPoint
//мы используем сеттер setX() для установки значения X
let myPoint = new MyPoint();
myPoint.setX = 4;
console.log( myPoint.getX() ); //будет выведено 4;

```

Механизм сеттеров и геттеров позволяет задавать ограничения, применяемые при установке или чтении свойств класса.

Вспомогательные механизмы для работы с сеттерами и геттерами

Вместо того, чтобы использовать конструкцию вида `myPoint.setX()` для того, чтобы установить значение `x`, что если можно было бы поступить примерно так:

```

myPoint.X = 4; //работа с X так, как будто это свойство, хотя на самом деле это сеттер

```

Для того, чтобы подобный механизм заработал, при объявлении геттеров и сеттеров нужно поставить перед именем функций ключевые слова `get` и `set`, то есть, сравнивая это с предыдущими примерами, отделить эти слова пробелом от следующего за ними имени функции:

```

class MyPoint {
  constructor (private x?: number, private y?: string) {}
  get X() { //обратите внимание на пробел перед X
    return this.x;
  }
  set X(value) { //обратите внимание на пробел перед Y
    this.x = value;
  }
}

```

Кроме того, общепринятой практикой является использование знаков подчёркивания в начале имён свойств (`_`):

```

class MyPoint {
  constructor (private _x?: number, private _y?: string) {}
  get x() { //обратите внимание на пробел перед X
    return this._x;
  }
  set x(value) { //обратите внимание на пробел перед Y
    this._x = value;
  }
}

```

Когда вы приступите к созданию реальных приложений, вы обнаружите, что в них понадобится далеко не один класс. Поэтому хорошо бы найти средство, позволяющее создавать классы так, чтобы их можно было использовать в других файлах, и в классах, объявленных в этих файлах, то есть, нам нужны инструменты написания модульного кода. Такие инструменты уже имеются в TypeScript. Для того, чтобы их рассмотреть, приведём код в файле `myPoint.ts` к такому виду:

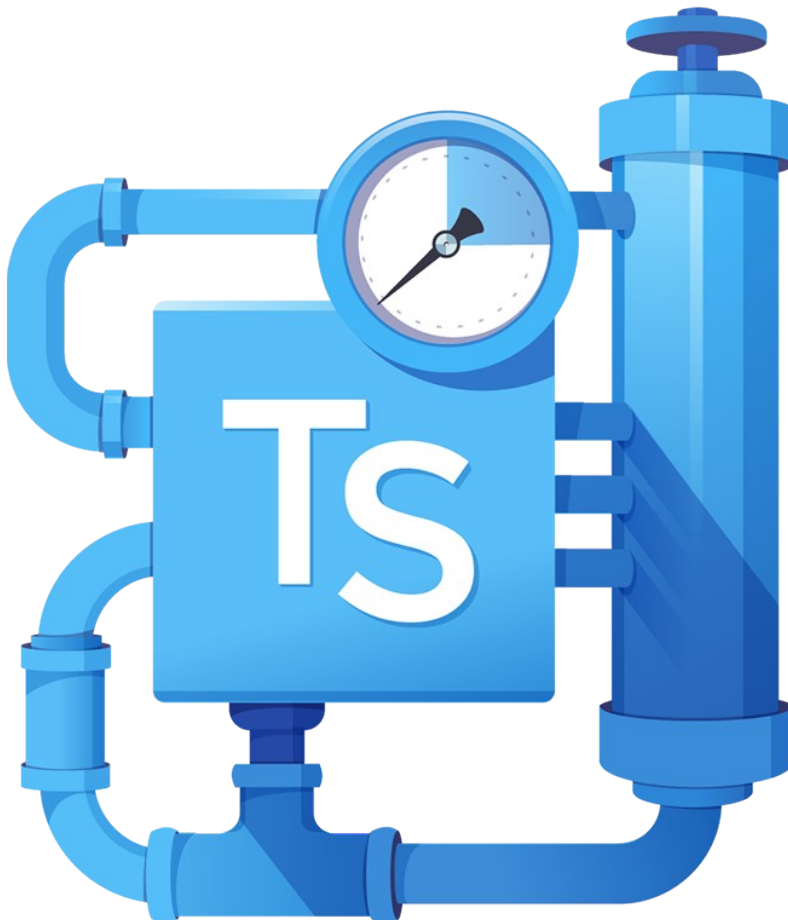
```
export class MyPoint { //обратите внимание на новое ключевое слово export
  constructor (private _x?: number, private _y?: string) {}
  get x() {
    return this._x;
  }
  set x(value) { //обратите внимание на пробел перед x
    this._x = value;
  }
  draw() {
    //нарисуем что-нибудь
  }
}
```

Благодаря ключевому слову `export` класс, описанный в файле `MyPoint.ts`, можно сделать видимым в других файлах, импортировав его в них с использованием ключевого слова `import`.

Для того, чтобы, например, пользоваться классом `MyPoint` в файле `main.ts`, его надо импортировать:

```
import { MyPoint } from './myPoint';
class Main {
  let MyPoint = new MyPoint(4, "go to go");
  MyPoint.draw();
}
```

Обратите внимание на то, что файлы `main.ts` и `myPoint.ts` находятся в одной и той же директории.





В этом материале мы рассмотрели основы TypeScript, которые необходимы для того, чтобы начать писать приложения на Angular. Теперь вы сможете понять устройство Angular-кода, а значит, у вас, кроме прочего, появится возможность эффективно осваивать руководства, которые посвящены этому фреймворку и предполагают знание TS.

Уважаемые читатели! Если вы хорошо разбираетесь в Angular — просим рассказать новичкам о том, как вы его изучали, и посоветовать хорошие учебные материалы по нему.

Теги: [JavaScript](#), [TypeScript](#), [Angular](#), [разработка](#)

+12 127 23,1k 27

**RUVDS.com** 1099,00
RUVDS – хостинг VDS/VPS серверов

**165,5** **1588,8** **659**
Карма Рейтинг Подписчики

@ru_vds
Пользователь

[Facebook](#) [Twitter](#) [ВКонтакте](#) [Google+](#)

Поделиться публикацией



ПОХОЖИЕ ПУБЛИКАЦИИ

20 декабря 2017 в 15:22

Node.js и JavaScript для серверной разработки

+13 22,3k 126 49

4 декабря 2017 в 12:01


Основы регулярных выражений в JavaScript

+22 17,4k 260 20

24 июля 2017 в 14:16

Почему мы выбрали TypeScript: история разработчиков из Reddit

+27 15,5k 47 55

**ВАКАНСИИ КОМПАНИИ RUVDS.COM** **Мой круг**

Full stack разработчик от 120000 до 150000 Р

Москва · Полный рабочий день

[Вакансии компании](#) [Создать резюме](#)

Комментарии 27

 **Drag13** 12 декабря 2017 в 14:49 +12

Пожалуйста, перестаньте советовать использовать `npm install whatever -g`

Если ты работаешь один — это нормально. Но как только ты начинаешь работать в команде это головная боль, особенно для TypeScript который работает сборщиком и валидатором одновременно.

Объясню почему.

1. Разные правила валидации

У меня стоит глобально версия 2.1.4, у коллеги 2.2.1. Обе мажорные версии и тем не менее они могут показывать разные ошибки. В итоге мой билд на соберется у него. (У меня локально все работает!)

2. Разная сборка

Если мы не ставим typescript per repository мы не можем гарантировать какой версией tsc будет обработан наш TS код. Соответственно что? Мы не можем быть на 100% уверены что наш локальный билд и билд на билд машине тот же. Не слишком ли круто?

3. Андейты. Мы крутые ребята у нас tsc глобален и мы его используем для всех репозиторий. И тут мы решаем его слегка обновить. И половина локальных репозиторий вдруг перестает собираться (см. п.1) Прекрасно.

Потратьте немного времени, ставьте tsc локально и просто пишите корректный npm script run. Берегите свои силы и нервы.



skoder 12 декабря 2017 в 15:03

0

По одному проекту потребовалось мне изучить азы Angular, чтобы написать небольшую прослойку между моим TypeScript проектом и Angular, дабы захватить аудиторию последнего. Вот и статья, судя по названию, должна была быть именно про это. Но про Angular ни слова.

EastJesus 12 декабря 2017 в 16:29

0

TS выглядит как смесь JS и C++

search 13 декабря 2017 в 13:41

0

Скорее JS и C#.



kahi4 12 декабря 2017 в 16:32

+4

Отвратительный стиль подачи и выбор порядка материала.

Необязательные параметры в конструкторе

Необязательные параметры в любой функции, не только в конструкторе, помечаются таким образом. Помимо того, они всегда должны быть после обязательных, а еще могут иметь значение по-умолчанию.

Типов данных в TS не так уж и много, можно было бы и все перечислить. В данном конкретно случае пропущен, как минимум, функциональный тип, который является достаточно важным. Но что еще хуже — ни слова об `type` `SOME_TYPE = 'a' | 'b'` и вообще об, вроде это называется "алгебраическая система типов", но это не точно, что позволяет писать `number | string` или `someInterface & anotherInterface`.

И что классы с конструкторами и гетерами, что стрелочные функции, есть в js. Пишите про отличия? Пишите про отличия. А так это просто набор каких-то случайных выдержек из документации.

Про модули просто невероятно выразительно расписано. Даже про возможность написать `import * as MyPoint from './myPoint';`
Ни слова.



spmbt 13 декабря 2017 в 02:26

0

Ми качественно подаём только за деньги. Неужели ви думаете, что мы будем для вас стараться? У нас все советы такие.

JSmitty 13 декабря 2017 в 07:35

0

Модули как в ES6, при чем тут TS?



kahi4 13 декабря 2017 в 10:06

+1

Я тоже не знаю зачем это есть в этой статье. Но оно есть, только в том виде, в котором все равно пойдешь читать документацию чтобы понять о чем речь.



pletinsky 12 декабря 2017 в 18:43

0

Было бы интересно почитать про особенности TypeScript по сравнению с современным JavaScript (ES6 etc.), а не со старой версией языка.



Drag13 12 декабря 2017 в 19:16

0

А какие особенности интересуют? Интерфейсы, generics, private fields (которые еще не приватные)? Не то что бы там было сильно много и наверняка уже все написано...



pletinsky 12 декабря 2017 в 21:12

0

Да, не то, чтобы что-то конкретное интересует. Просто рассматривать ts на базе старого js как то нерелевантно (имхо) чтоли. Выбирают то между ts и es (ну, не в случае ангуляра конечно). Для рекламы это хорошо. А практическая пользы намного меньше.



Drag13 13 декабря 2017 в 00:17

0

Мне ts удобнее. Коллега плюется и требует ES6. Каждому свое.



pletinsky 13 декабря 2017 в 00:58

0

Ну я не думаю, что дело тут только в предпочтениях. У TS есть конкретные фишки, которых в ES6 нет и скорее всего не будет. Интерфейсы прежде всего. Которые делают актуальным TS во вполне конкретных ситуациях (поставка интерфейсов в условиях низкого покрытия кода модульными тестами и т.п.).

А так как в целом синтаксис у них не очень то и отличается, то между ними можно выбирать на базе потребностей проекта.

И, конечно, фишки именно TS гораздо важнее, чем описанные, тут, например, стрелочные функции. Которые, вроде, уже во всех языках есть.

MisterPurple 12 декабря 2017 в 22:06

0

Имя метода геттера и сеттера принято/необходимо называть одинаково с именем поля, к которому осуществляется доступ, а у Вас почему-то имя получилось в аппер-кейсе

```
get X() { //обратите внимание на пробел перед X
  return this.x;
}
set X(value) { //обратите внимание на пробел перед Y
  this.x = value;
}
```

Druu 14 декабря 2017 в 11:04

0

> Имя метода геттера и сеттера принято/необходимо называть одинаково с именем поля, к которому осуществляется доступ

Его нельзя называть одинаково.

MisterPurple 14 декабря 2017 в 11:25

0

Действительно, спасибо, я ошибся, перепутал с [Object.defineProperty](#)

TourDeFrants 12 декабря 2017 в 22:06

0

привет всем, вот интересует вопрос, может не совсем по теме.

Нашей команде заказчик дал задание: написать ОДНО приложение (типа Megogo) для просмотров фильмов и ТВ каналов на платформах Tizen, WebOS, Android, iOS. Мы думаем использовать для этого Cordova + Angular. интересует как это соединить, стоит ли вообще соединять или лучше писать на нативном ES6, и вообще кто сталкивался с подобными заданиями подскажите направление решения) ответы типа «пишите для каждой платформы свое приложение» не интересуют, заранее спасибо)

JSmitty 13 декабря 2017 в 07:45

0

За тизен скажу — пишите на ванильном JS или край на AngularJS (который 1). Мы писали на реакте, кордова там вообще лишнее — всё платформо-специфичное API выведено в JS. Задолбаться — движок встроенный глючный, API телика документировано, но реалиям не соответствует. У ТВ разных модельных годов есть мягко говоря «особенности» воспроизведения видеопотоков. Самсунг регулярно выкатывает «новые» версии тизен, но косяки всё те же. Отладчика нормального нет.

По ощущениям чем проще фреймворк, тем легче будет отлаживаться. Смотреть в километровые трейсы (на любом современном фреймворке) то ещё удовольствие — и вы не сможете сказать сразу, то ли вы облажались, то ли платформа глючит.

Очень похожая ситуация, судя по заметкам в инете — и на WebOS (LG).

Про производительность кордовы на мобильных платформах не писал только ленивый — смотрите сами, хватит ли вам её.

TourDeFrants 13 декабря 2017 в 08:52

0

Спасибо JSmitty, обсудим командой.

Если у кого еще есть опыт / советы, с радостью слушаем.

unsafePtr 12 декабря 2017 в 22:47

+1

Всем советую, у кого есть готовая библиотека или какие-нибудь компоненты написать файл дефиниции типов(d.ts). Это позволит другим людям использовать вашу библиотеку в тайпскрипте, ну а если вы пользуетесь VisualStudio Code, так вы в придачу получаете intellisense, да же из javascript кода.



Fragster 13 декабря 2017 в 12:21

0

А статические анализаторы по типу PVS для js есть? Пора бы уже им появляться, ведь копипаста вредна независимо от языка программирования...

```
draw() {  
    //обратите внимание на то, что со свойством x мы работаем через this  
    console.log("X is: " + this.x);  
    console.log("X is: " + this.y);  
}
```



MNB 13 декабря 2017 в 12:53

0

ESLint/TSLint + очень неплохой встроенный анализатор в WebStorm



Fragster 13 декабря 2017 в 18:51

0

Они помогут найти «ошибку» в приведенном мной фрагменте?



MNB 13 декабря 2017 в 22:02

0

Вы спросили про анализаторы. Я ответил.
WebStorm умеет находить копипасту.



paratagas 13 декабря 2017 в 13:43

+1

В статье не все примеры рабочие. Думаю, если читателю (под которым подразумевается новичок в Typescript) предлагается запустить какой-либо код, то он должен быть приведен корректно. В файле main.ts внутри функции пропущен метод console.

```
console(message);
```

В таком виде код упадет с ошибкой компиляции. Также внутри myPoints.ts в метод getDistanceBtw передается неизвестный тип AnotherPoint.

```
getDistanceBtw(another: AnotherPoint) {  
    //посчитать и вернуть расстояние  
}
```

Это вызовет ошибку при компиляции Typescript. Что-то вроде «Cannot find name 'AnotherPoint'». И зачем вообще приводить код этого метода, если он нигде не вызывается?



taujavarob 13 декабря 2017 в 17:15

+1

В статье не все примеры рабочие.

Хвалить Typescript за то что он позволяет на стадии компиляции отлавливать ошибки и приводить примеры которые падают на стадии компиляции как рабочие примеры? - **Это удивляет.**



paratagas 14 декабря 2017 в 10:29

0

В примерах автора участки кода, отмеченные мной, позиционируются как рабочие. Автор не указывает, что в этих местах программа должна упасть. Напротив, он пишет, что результатом должно стать создание файла и класса соответственно. Если бы автор указал, что в приведенных двух случаях Typescript «подстрахует», я бы не писал свой первый комментарий.

Только [полноправные пользователи](#) могут оставлять комментарии. [Войдите](#), пожалуйста.

САМОЕ ЧИТАЕМОЕ










Сутки

Неделя

Месяц

Гаджеты с барахолки: зачем покупать 20-летний ноутбук Packard Bell за 10 евро

+58	52,4k	42	126
Для чего и как мы скрываем госномера автомобилей в объявлениях Авито			
+62	33,5k	86	116
Принципы построения REST JSON API			
+30	24,5k	451	186
Интервью с Владимиром Лихачевым, отцом Николая Лихачева, более известного как Крис Касперски			
+67	19,4k	56	81
Автоматизация бизнес-процессов в Excel или как спасти девушку от переработок			
+37	18,6k	53	42

Аккаунт	Разделы	Информация	Услуги	Приложения
Войти	Публикации	Правила	Реклама	<div>Загрузите в  App Store</div> <div>доступно в  Google Play</div>
Регистрация	Новости	Помощь	Тарифы	
	Хабы	Документация	Контент	
	Компании	Соглашение	Семинары	
	Пользователи	Конфиденциальность		
	Песочница			
<div><div> © 2006 – 2019 «ТМ»</div><div>Настройка языка</div><div>О сайте</div><div>Служба поддержки</div><div>Мобильная версия</div><div></div></div>				