Конспект лекции

Основы языка Typescript

Перед началом лекции наставнику требуется:

В папке source в каждой папке где присутствует файл package.json запустить команду “npm install”. Необходимый результат - команда должна выполниться без ошибок везде.

[Слайд 2]

Цель и задачи лекции

Цель – ознакомиться с языком Typescript и научиться его применять.

Задачи:

1. Изучить условия и цель появления языка.

2. Познакомиться с типизацией по TS.

3. Настроить конфиг и научиться применять команды tsc.

[Слайд 3]

План занятия

* Условия появления
* Отличительные особенности
* Определение
* Основы типизации по TS
* Практика
  + базовые типы
  + tsconfig
  + tslint
  + counter

1. Условия появления [Слайд 4]

Язык Typescript был разработан компанией Microsoft.Первый релиз состоялся в 2012 году. На тот момент актуальной спецификацией js является ES5. Предыдущее обновление было давно, да и ‘use strict’ не привнес всего, чего на тот момент уже хотели от javascript.

В это время модно писать собственные надмножества над js. Более того каждая уважающая себя крупная компания считала обязательным писать и использовать в разработке собственный язык на основе js.

На сегодняшний день известными языками основанными на js, являются coffeescript, dart, typescript.

Все чувствовали необходимость привнести в js особенности других языков.

Зачем ? Почему?

[Слайд 5]

Дело в том что javascript стали использовать для более сложных задач, чем те, для которых его создавали.

Все рано или поздно стали приходить к выводу, что строгая типизация нужна, так как проекты разрастаются, усложняются обрастают if-ами, его становится сложно поддерживать и модифицировать. Кроме того работа больших команд без типизации затрудняется.

2. Отличительные возможности

2.1 Что привносит TS

1 - В момент появления ts добавлял классы и модель наследования основанную на классах, которых на тот момент так не хватало разработчикам js.

2 - Типизация (опциональная). То есть уровень строгости типизации в typescript регулируются настройками. При этом тип типизация настраивается до такой степени что ее можно и вовсе отключить.

3 - stage 0 по умолчанию. Typescript превращается в javascript перед исполнением. Это означает что нет никаких преград для быстрого внедрения самых последних фич, которые на js еще долго будут оставаться экспериментальными из-за нерасторопности экосистемы javascript.

2.1 Чем отличается на фоне собратьев.

[Слайд 6]

Ведь разработчики typescript далеко не первыми пытались привнести все это.

1 - Все проверки с типизацией происходят до runtime в момент транспиляции кода в javascript. При этом в конечном javascript все проверки отсутствуют, что делает конечный бандл легче. Кроме того после транспиляции javascript остается читабельным. В Microsoft хотеля, чтобы их язык применяли не только в новых проектах, но и в уже существующих. По этому компилятор ts превращая typescript код в javascript код, и по сей день генерирует код максимально похожим на код, который пишут реальные программисты. Тем самым предполагалось, что в команде javascript разработчиков может работать typescript разработчик, который будет превращать свой код в js и оставаться незамеченным)

2 - Все современные IDE и редакторы кода знают typescript и подскажут вам об ошибках еще на этапе написания кода. Кроме того благодаря типизации подсказки будут содержать подробную информацию о любой переменной или способе применения функции или метода. Что существенно упрощает, а значит и ускоряет работу, особенно в большой команде и на больших проектах.

3 - Очень популярен и развивается. Популярность языка это очень важный показатель, так как отображает перспективность изучения и использования языка на ваших проектах.

4 - Можно типизировать уже существующие js библиотеки от сторонних разработчиков не трогая код самой библиотеки. Для таких целей создается специальный файл с расширением d.ts, в котором декларируются все типы переменных, функций классов и т.д. содержащихся в библиотеке.

Из минусов отмечают отсутствие проверки на runtime.

Обратите внимание на график популярности typescript за последние 5 лет.

[Слайд 7]

Следующий график отображает популярность запросов typescript в сравнении с babel

[Слайд 8]

3. Определение

[Слайд 9]

И так что же такое typescript?

В общем если собрать все что я рассказал о typescript в одно предложение, то

typescript - это компилируемое надмножество javascript, приносящее опциональную статическую типизацию и некоторые возможности современного стандарта ECMAScript.

4. Практика

[Слайд 10]

Первым делом нужно установить компилятор typescript. Сделать это можно командой “npm install -g typescript”.

4.1 Базовые типы

* логический тип. Boolean
* цифровой. number. В нем объединены все типы цифр.
* строковый. string. Любая обычная или шаблонная строка.
* массивы. array<T>. Массивам указывается тип данных содержащихся в массиве. После чего в массив можно поместить данные только соответствующего типа.
* enum. Энам это перечисление из ограниченного числа вариантов
* Ничего (пустота). void. Тип данных указываемый для функций, которые ничего не возвращают.
* Так же существуют отдельные типы для undefined и null.\

Рассмотрим пример. [папка 01\_simple, файл index.js]

перед нами обычный javascript файл. Давайте перепишем его на typescript. Первое что нужно сделать это изменить расширение файла с .js на .ts (переименовать файл index.js в index.ts)

Все, поздравляю вас. Мы завершили процедуру перевода на typescript. Это утверждение верно благодаря опциональности типизации typescript. Теперь чтобы скомпилировать файл в javascript нужно набрать команду “tsc index.ts”.

Рядом появился файл index.js в который был странспилирован наш код. Обратите внимание, что он был переведен на спецификацию es5. Это настройка по умолчанию. Что бы транспилировать в новый javascript нужно указать параметр target.

Команда будет выглядеть так “tsc --target es6 .\index.ts” (запустить команду в папке 01\_simple).

Приглядитесь внимательно к функции add. она принимает 2 параметра и складывает их, но при этом функция не различает сложение цифр и строк. то-есть во втором примере на строке 7 мы получим не 3 а 12. Давайте представим что это не то чего мы хотим. Допустим мы хотим получить предупреждение что мы делаем что-то для чего функция не предназначена. Давайте добавим сюда типизацию и посмотрим как typescript поможет нам избежать ошибки.

переписать 4 строку на

**const *add* =(*a: number*, *b: number*): number=>a+b;**

Обратите внимание. что на 7ой строке строка ‘2’ подчеркнута. Так же если перезапустить транспиляцию можно увидеть что в консоли появляются сообщения как раз о том, что мы прокидываем строку во второй параметр функции, ожидающей на этом месте цифру.

При это javascript файл все равно был сгенерирован. Это можно исправить с помощью опции “noEmitOnError”.

“tsc --target es6 --noEmitOnError .\index.ts”.

4.2 Конфиг

[Слайд 11]

Для настройки процесса транспиляции typescript в javascript в проектк создается файл tsconfig.json.

Внутри файла создаем объект со свойством “compilerOptions”.

“compilerOptions” - тоже объект в который прописываются настройки компилятора.

[Слайд 12]

вот некоторые из них:

* target - *Указывает на версию ecmaScript. [es5, es6, ... ]*
* *declaration - указывает создавать ли декларационные файлы [boolean]*
* *sourceMap - указывает создавать ли sourcemap [boolean]*
* *outDir - путь для генерации js файлов [string]*
* *noEmit - указывает сохранять ли результат в файл [boolean]*
* *strict - указывает включена ли строгая проверка типов [boolean]*
* *noImplicitAny - указывает разрешена ли автоматическая проставленовка типа any [boolean]*
* *strictNullChecks - указывает включена ли проверка на отсутствие не обязательных свойств [boolean]*
* *noUnusedLocals - указывает разрешены ли не используемые локальные переменные [boolean]*
* *noUnusedParameters - указывает разрешены ли не используемые параметры [boolean]*
* *noImplicitReturns - указывает на возможность не указывать тип возвращаемого значения [boolean]*
* *inlineSourceMap - указывает создаватьли js sourcemap прямо в файле js [boolean]*

рассмотрим примеры настройки конфига.

[папка 02\_basic\_config файл tsconfig.json]

Если в проекте присутствует tsconfig, то настройки для tsc есть там, и прописывать их в команде почти нет смысла.

для транспиляции прописываем команду “tsc”. Что получилось:

1 Благодаря свойству outDir весь результат был помещен в папку dist.

2 Благодаря свойству target со значением es6 в итоговом javascript соблюдена спецификация es6

3 Благодаря свойству module со значением es2015 сохранен синтаксис модулей import/export

4 Свойства files и exclude изменили область видмости компилятора таким образом что преобразовываться будут только импортированные файлы, а не все находящиеся в папке src.

(Продемонстрировать что при закомментированный строках 1 и 6 файлы src/index.ts в папке dist не появляется файл utils.js)

Свойство module. Начиная с ECMAScript 2015, JavaScript имеет концепцию модулей. TypeScript разделяет эту концепцию.

Модули выполняются в своей собственной области, а не в глобальной области; это означает, что переменные, функции, классы и т. д., объявленные в модуле, не видны вне модуля, если они явно не экспортированы с использованием одной из форм экспорта. И наоборот, чтобы использовать переменную, функцию, класс, интерфейс и т. Д., Экспортированные из другого модуля, его необходимо импортировать с помощью одной из форм импорта.

Модули декларативные; отношения между модулями определяются в терминах импорта и экспорта на уровне файлов.

Модули импортируют друг друга, используя загрузчик модулей. Во время выполнения загрузчик модулей отвечает за поиск и выполнение всех зависимостей модуля перед его выполнением. Хорошо известные загрузчики модулей, используемые в JavaScript, - это загрузчик модулей CommonJS для Node.js и require.js для веб-приложений.

В TypeScript, как и в ECMAScript 2015, любой файл, содержащий импорт или экспорт верхнего уровня, считается модулем. И наоборот, файл без каких-либо объявлений импорта или экспорта верхнего уровня рассматривается как скрипт, содержимое которого доступно в глобальной области (и, следовательно, также для модулей).

Рассмотрим еще несколько свойств

[папка 03\_more\_options]

На данный момент конфиг ничем не отличается от предыдущего и команда tsc приведет к такому же результату.

Раскомментируем свойство declaration

Рядом с .js файлами появились файлы с декларациями типов .d.ts файлы. Рассмотрим utils.d.ts. В нем описана типизация для функции add. typescript и многие редакторы кода и IDE умеют читать такие файлы. Благодаря этому можно импортировать в typescript проект - js библиотеку. В таком случае такими библиотеками можно пользоваться и в ts и в js проектах.

Думаю не для кого не секрет что случится если раскомментировать строку со свойством sourceMap - появятся файлы .js.map

При этом если заменить это свойство на inlineSourceMap (23 строка), то эта же информация появится прямо в файлах.

Свойство noEmit, как говорилось ранее, указывает создавать ли файлы по результату транспиляции. Такой параметр добавляютдля того, что бы просто проверить собирается ли проект без ошибок например перед отправкой комита.

Свойство strict включает все проверки на строгую тиизацию, при этом вместо общего правила strict = true можно перечислить свойства, уточняющие конкретные правила. Например noImplicitAny и strictNullCkeck.

4.3 tslint

TSLint - это расширяемый инструмент статического анализа, который проверяет код TypeScript на наличие ошибок читаемости, удобства обслуживания и функциональности. Он широко поддерживается в современных редакторах и системах сборки и может быть настроен с использованием ваших собственных правил, конфигураций и средств форматирования.

Для работы tslint нужно установить командой “npm install -g tslint”.

Далее в каталоге с проектом можно создать файл “tslint.json” либо вручную, либо командой “tslint --init”. Tslint понимает правила jslint и дополнен правилами, охватывающими особенности typescript.

Рассмотрим основные:

# ***no-angle-bracket-type-assertion*** - запрет на исользование приведения типа через угловые скобки. Дело в том, что в jsx можно приводить тип только вариантом через “as”.

* ***no-any*** - запрет на тип any.

# ***no-empty-interface*** - запрет на “пустые” интерфейсы.

# ***await-promise*** *-* запрет на использование await перед методами не возвращающими обещание.

Все рекомендуемые параметры находятся в сборке tslint:recommended,которую можно брать за основу расширяя собственными правилами добавляя в начало объекта настроек свойство “extends” со значением [“tslint:recommended”].

"extends": ["tslint:recommended"]

4.4 Counter.

Теперь посмотрим на пример небольшого приложения написанного на js, основанного на create-react-app.

[папка 05\_counter. запустить приложение командой “npm start”]

Перед нами простой счетчик то 0 до 99. можно прибавлять, отнимать и сбрасывать значение. В данном приложении присутствует баг, который мы постараемся отловить, переписав приложение на typescript.

1 шаг - переименовать файлы в ts.

(переименовать 05\_counter/src/App.jsx => 05\_counter/src/App.tsx,

05\_counter/src/data.js => 05\_counter/src/data.ts,

05\_counter/src/index.jsx => 05\_counter/src/index.tsx, после чего перезапустить приложение командой “npm start”, возможно потребуется перезапустить редактор кода)

После первого шага приложение может перестать собираться из-за отсутствия типизации компонента App. Чтобы это исправить допишем “<any, any>” в файле App.tsx на 9 строке перед символом “{” (должно получиться так “class App extends React.Component<any, any> {“).

Теперь когда приложение снова запускается, только тепеь через typescript, можно добавить типизацию и посмотреть как typescript подскажет нам где проблема.

2 шаг Добавляем типизацию.

Добавляя 2 any в угловых скобках в месте объявления о расширении класса Component, мы указываем typescript какие два типа мы будем использовать с данном компоненте в качестве props и state.

Никаких props приложение App не принемает, так что в качестве props укажем пустой объект.(9 строка, в конце Component <{}, any>)

В качестве state у приложения есть объект со свойством count. Напишем для него интерфейс.

interface IAppState {

count: number;

}

И укажем его вместо второго any

(9 строка, в конце Component <{}, IAppState>)

В консоле появилась ошибка



Argument of type '{ count: string; }' is not assignable to parameter of type 'IAppState | ((prevState: Readonly<IAppState>, props: Readonly<{}>) => IAppState | Pick<IAppState, "count">) | Pick<IAppState, "count">'.

Type '{ count: string; }' is not assignable to type 'Pick<IAppState, "count">'.

Types of property 'count' are incompatible.

Type 'string' is not assignable to type 'number'

Для нас самым главным тут являются две последние строчки. В них говорится что мы пытаемся присвоить строку в качестве значения для свойства count, тогда как в интерфейсе мы обязались присваивать только цифры. меняем строковый ноль на цифровой и видим вторую такую же ошибку, но про другое место. Исправляем проблему и там и ошибки пропадают. После чего, если проверить приложение, можно убедиться что баг пропал. И хотя проблема решена, в приложении еще есть места, которые нужно типизировать для недопущения новых багов при последующем развитии приложения.

Давайте включим свойство strictMode и noImplicitAny в tsconfig (заменить false на true напротив перечисленных свойств в файле 05\_counter/tsconfig.json)

Теперь в консоле появляется новые ошибки, в этих сообщениях указано что методы компонента App принимают параметры, к которым не указаны типы. Исправим это.

18 стр getBlockState = (*number*: number) => (*\_*: undefined, *index*: number) => {

22 стр getNumberBlocks = (*index*: number, *number*: number) => {

38 стр renderBlock = (*blockState*: boolean, *index*: number) => {

Мы закончили. Приложение теперь написано на typescript. Обратите внимание, что в приложении осталось много переменных и методов для которых не вся типизация указана. Например для методов мы не указывали возвращаемое значение, но при этом в консоле нет ошибок. Это происходит потому что typescript сам смотрим на то что мы возвращаем в функции и сам принимает решение в тех ситуациях когда это очевидно. При этом предположение делается даже по типам массивов, находящихся в файле data.ts.

Тип для этих массивов выглядит так:

Array<number> или по другому number[]. Эти записи равноправны.

С более сложными ситуациями, связанными с типизацией мы познакомимся в дальнейшем.

Литература и ссылки

1. Официальная документация по typescript <https://www.typescriptlang.org/docs/home.html>.
2. Опции для компилятора typescript <https://www.typescriptlang.org/docs/handbook/compiler-options.html>

Вопросы для самоконтроля

1. Что такое typescript.
2. В чем различия правил в файле tsconfig.json и tslint.json.
3. Для чего нужны файлы d.ts.
4. Каковы плюсы и минусы typescript, по сравнению с javascript.