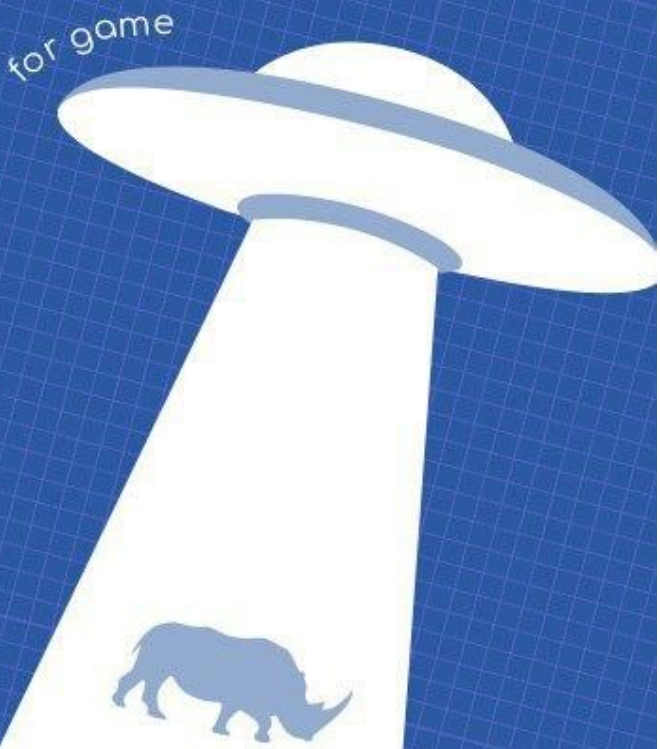
TYPESCRIPT DEFINITIVE GUIDE 3.2



TypeScript
Подробное Руководство

Книга и документация в одном

Дата последнего обновления: 9/5/2020

Глава 11

Аннотация Типов

Чтобы избавится от страха возникающего от слова *типизация* необходимо в самом начале увидеть все преобразования которые проделал *TypeScript* над своим фундаментом коим для него является, никого не оставляющий равнодушным *JavaScript*.

Аннотация Типов - общее

Как уже было сказано ранее, *TypeScript* — это типизированная надстройка над *JavaScript*. Другими словами *TypeScript* не добавляет никаких новых языковых конструкций (за исключением `Enum`, которая будет рассмотрена чуть позже), а лишь расширяет синтаксис *JavaScript* за счет добавления в него типов. По этой причине в этой книге не будут затрагиваться темы относящиеся к *JavaScript*, так как она рассчитана на тех, кто уже знаком с его основами. Именно поэтому погружение в типизированный мир *TypeScript* необходимо начать с рассмотрения того как типизация преобразила *JavaScript* конструкции.

Аннотация типа

В *TypeScript* аннотация типа или указание типа осуществляется с помощью оператора двоеточия `:`, после которого следует идентификатор типа. *TypeScript* является статически типизированным языком, поэтому после того как идентификатор будет связан с типом, изменить тип будет невозможно.

Синтаксические конструкции var, let, const

При объявлении синтаксических конструкций объявляемых с помощью операторов `var`, `let` и `const`, тип данных указывается сразу после идентификатора.

```
var identifier: Type = value;
let identifier: Type = value;
const IDENTIFIER: Type = value;
```

Функции (function)

При объявлении функции тип возвращаемого ею значения указывается между её параметрами и телом. При наличии параметров, тип данных указывается и для них.

```
function identifier(param1: Type, param2: Type): ReturnedType {

}
```

Не будет лишним напомнить, что, в отличие от *JavaScript*, в *TypeScript* в сигнатуру функции помимо её имени и параметров также входит и возвращаемое значение.
Помимо этого, в *TypeScript* можно объявлять параметризированные функции. Функции, имеющие параметры типа, называются обобщенными (подробнее о них речь пойдет в главе "Типы - Обобщения (Generics)". Параметры типа заключаются в угловые скобки `<>` и располагаются перед круглыми скобками `()`, в которые заключены параметры функции.

```
function identifier <T, U>(): ReturnedType {

}
```

Кроме того *TypeScript* расширяет границы типизирования функций и методов с помощью незнакомого *JavaScript* разработчикам механизма *перегрузки функций*. С помощью перегрузки функций можно аннотировать функции с одинаковыми идентификаторами, но с различными сигнатурами.
Для этого перед определением функции, метода или функции-конструктора перечисляются совместимые объявления одних только сигнатур. Более подробно эта тема будет освещена позднее.

```
function identifier(p1: T1, p2: T2): T3;
function identifier(p1: T4, p2: T5): T6;
function identifier(p1: T, p2: T): T {
    return 'value';
}

const a: T1 = 'value';
const b: T2 = 'value';
const c: T4 = 'value';
const d: T5 = 'value';

identifier(a, b); // валидно
identifier(c, d); // валидно

class Identifier {
    constructor(p1: T1, p2: T2);
    constructor(p1: T3, p2: T4);
    constructor(p1: T, p2: T) {

    }

    identifier(p1: T1, p2: T2): T3;
    identifier(p1: T4, p2: T5): T6;
    identifier(p1: T, p2: T): T {
        return 'value';
```

```
    }
}
```

Стрелочные Функции (arrow function)

К стрелочным функциям применимы те же правила указания типов данных, что и для обычных функций, за исключением того, что возвращаемый ими тип указывается между параметрами и стрелкой.

```
<T, U>(param: Type, param2: Type): Type => value;
```

Классы (class)

Прежде чем продолжить рассмотрение изменений которые привнёс *TypeScript* в нетипизированный мир *JavaScript*, хотелось бы предупредить о том, что относительно классов будет использоваться терминология заимствованная из таких языков, как *Java* или *C#*, так как она способствует большей ясности (тем более, что в спецификации *TypeScript* встречается аналогичная терминология). Так, *переменные экземпляра* и *переменные класса* (статические переменные) в этой книге обозначаются как *поля* (*field*). *Аксессоры* (*get*_set*) обозначаются как* свойства (*_property*). А кроме того, поля, свойства, методы, *вычисляемые свойства* (*computed property*) и *индексируемые сигнатуры* (*index signature*) обозначаются как *члены* класса (*member*).
При объявлении поля класса, как и в случае с переменными, тип данных указывается сразу после идентификатора (имени класса). Для методов класса действуют те же правила указания типов что и для обычных функций.
Для свойств, в частности для `get`, указывается тип данных возвращаемого значения. Для `set` указывается лишь тип единственного параметра, а возвращаемый им тип и вовсе запрещается указывать явно.

Кроме того, классы в *TypeScript* также могут быть обобщенными. В случае объявления обобщенного класса, параметры типа, заключенные в треугольные скобки, указываются сразу после идентификатора класса.

```
class Identifier<T> {
    static staticField: Type = value; // член класса

    static get staticProperty(): Type { // член класса
        return value;
    }

    static set staticProperty(value: Type) { // член класса

    }

    static staticMethod <T, U>(param0: Type, param1: Type): Type { //
член класса

    }

    [indexSignature: Type]: Type; // член класса

    [computedProp]: Type = value; // член класса

    field: Type = value; // член класса

    get property(): Type { // член класса
        return value;
    }

    set property(value: Type) { // член класса

    }

    constructor(param0: Type, param1: Type) {

    }

    method <T, U>(param0: Type, param1: Type): Type { // член класса
```

```
    }
}
```

Сравнение Синтаксиса TypeScript и JavaScript

Перед тем, как подвести итоги этой главы, не будет лишним собрать все рассмотренные *TypeScript* конструкции и наглядно сравнить их со своими нетипизированными *JavaScript* аналогами.

```
// .ts
var identifier: Type = value;
let identifier: Type = value;
const IDENTIFIER: Type = value;
//  .js
var identifier = value;
let identifier = value;
const IDENTIFIER = value;

// .ts
function identifier(param1: Type, param2: Type): ReturnedType {

}

// .js
function identifier(param1, param2) {

}

// .ts
class Identifier<T> {
    static staticField: Type = value;

    static get staticProperty(): Type {
        return value;
    }

    static set staticProperty(value: Type) {

    }

    static staticMethod <T, U>(param0: Type, param1: Type): Type {
```

```
    }

    [indexSignature: Type]: Type;

    [computedProp]: Type = value;

    field: Type = value;

    get property(): Type {
        return value;
    }

    set property(value: Type) {

    }

    constructor(param0: Type, param1: Type) {

    }

    method <T, U>(param0: Type, param1: Type): Type {

    }
}

// .js
class Identifier {
    static staticField = value;

    static get staticProperty() {
        return value;
    }

    static set staticProperty(value) {

    }

    static staticMethod (param, param) {

    }

    [computedProp] = value;

    field = value;

    get property() {
        return value;
    }

    set property(value) {

    }

    constructor(param0, param1) {

    }
```

```
    method (param0, param1) {

    }
}
```

Итог

- Аннотация типа устанавливается оператором двоеточия `:`, после которого следует указание типа данных.
- При объявлении переменных тип данных указывается сразу после идентификатора.
- У функций и методов класса возвращаемый тип данных указывается между параметрами и телом.
- У стрелочных функций возвращаемый тип данных указывается между параметрами и стрелкой.
- У функций, стрелочных функций и методов класса, параметрам также указывается тип данных.
- При необходимости функциям, стрелочным функциям и методам класса можно указать параметры типа, которые заключаются в угловые скобки и указываются перед круглыми скобками, в которых размещаются параметры функции.
- В *TypeScript* аннотирование типов у функций, методов и конструкторов расширено при помощи перегрузки функций.
- Для полей класса тип данных указывается сразу после идентификатора-имени.
- Для геттеров (getters) указывается возвращаемый тип данных.
- Для сеттеров (setters) указывается тип единственного параметра и вовсе не указывается возвращаемый тип.

Глава 12

Базовый Тип Any

Базовый Тип Any

То что *TypeScript* является типизированной надстройкой над *JavaScript*, от которой после компиляции не остаётся и следа, означает, что первый перенял от второго всю его идеологию. Одним из таких моментов является разделение типов данных на типы значения (примитивные) и ссылочные типы.
В *JavaScript* все не объектные типы по своей природе иммутабельные. Иммутабельность означает, что значения нельзя модифицировать, а можно лишь полностью перезаписать новым значением. Из-за этого типы значения были прозваны примитивными типами.

Any (any) произвольный тип

Все типы в *TypeScript* являются подтипами `any`, который указывается с помощью ключевого слова `any`. Это означает, что он совместим в обе стороны с любым типом данных и с точки зрения системы типов является *высшим типом* (top type).

```
let apple: any = 0;
apple = "";
apple = true;
```

Тип `any` в аннотации рекомендуется применять только в самых крайних случаях. К примеру объявление неинициализированной переменной, с заранее неизвестным типом,

что является допустимым сценарием на ранних стадиях применения техники `TDD` при создании программы. В остальных случаях настоятельно рекомендуется прежде всего рассматривать кандидатуру обобщенных типов, которые будут рассматриваться позднее.
Тип `any` позволяет работать со значением динамически, что не вызывает ошибок при обращении к членам не описанных в типе данных (незадекларированных), о которых анализатор *TypeScript* ничего не знает.
Примером этого может служить сервис, который работает с сервером посредством *API*. Полученные и сериализованные данные могут храниться как тип `any` прежде чем они будут и преобразованы к конкретному типу.

```
let data: any = JSON.parse('{"id": "abc"}');
let id = data.id; // ок
```

Если при объявлении переменных `var` и `let` и полей, объявленных в теле класса, не было присвоено значение, компилятором будет выведен тип данных `any` .

```
var apple; // apple: any
let lemon; // lemon: any

class Fruit {
    name; // name: any
}
```

То же правило касается и параметров функций.

```
function weight(fruit) { // fruit: any

}
```

Кроме того, если функция возвращает значение, принадлежащее к типу, который компилятор не в состоянии вывести, то возвращаемый этой функцией тип данных будет также будет выведен как тип `any` .

```
function sum(a, b) { // function sum(a: any, b: any): any
    return a + b;
}
```

Тип `any` является уникальным для *TypeScript*, в *JavaScript* подобного типа не существует.

Итог

- Тип данных `any` является супертипом для всех типов в *TypeScript* и указывается с помощью ключевого слова `any` .
- Тип данных `any` указывается тогда, когда тип данных заранее неизвестен.
- Тип данных `any` указывается только в самых крайних случаях.
- Тип данных `any` позволяет обращаться к незадекларированным членам объектов, имитируя динамическое поведение, тем самым перекладывая ответственность за проверку реализации интерфейса на разработчика.