TypeScript 的注释指令

TypeScript 接受一些注释指令。

所谓“注释指令”，指的是采用 JS 双斜杠注释的形式，向编译器发出的命令。

// @ts-nocheck

`// @ts-nocheck` 告诉编译器不对当前脚本进行类型检查，可以用于 TypeScript 脚本，也可以用于 JavaScript 脚本。

```javascript
// @ts-nocheck

const element = document.getElementById(123);
```

上面示例中， `document.getElementById(123)` 存在类型错误，但是编译器不对该脚本进行类型检查，所以不会报错。

// @ts-check

如果一个 JavaScript 脚本顶部添加了 `// @ts-check` ，那么编译器将对该脚本进行类型检查，不论是否启用了 `checkJs` 编译选项。

```javascript
// @ts-check
let isChecked = true;

console.log(isChceked); // 报错
```

上面示例是一个 JavaScript 脚本，`// @ts-check` 告诉 TypeScript 编译器对其进行类型检查，所以最后一行会报错，提示拼写错误。

// @ts-ignore

`// @ts-ignore` 或 `// @ts-expect-error`，告诉编译器不对下一行代码进行类型检查，可以用于 TypeScript 脚本，也可以用于 JavaScript 脚本。

typescript

```
let x: number;

x = 0;

// @ts-expect-error
x = false; // 不报错
```

上面示例中，最后一行是类型错误，变量 `x` 的类型是 `number`，不能等于布尔值。但是因为前面加上了 `// @ts-expect-error`，编译器会跳过这一行的类型检查，所以不会报错。

JSDoc

TypeScript 直接处理 JS 文件时，如果无法推断出类型，会使用 JS 脚本里面的 JSDoc 注释。

使用 JSDoc 时，有两个基本要求。

（1）JSDoc 注释必须以 `/**` 开始，其中星号（`*`）的数量必须为两个。若使用其他形式的多行注释，则 JSDoc 会忽略该条注释。

（2）JSDoc 注释必须与它描述的代码处于相邻的位置，并且注释在上，代码在下。

下面是 JSDoc 的一个简单例子。

javascript

```
/**
 * @param {string} somebody
 */
function sayHello(somebody) {
```

```
  console.log("Hello " + somebody);
}
```

上面示例中，注释里面的 `@param` 是一个 JSDoc 声明，表示下面的函数 `sayHello()` 的参数 `somebody` 类型为 `string` 。

TypeScript 编译器支持大部分的 JSDoc 声明，下面介绍其中的一些。

@typedef

`@typedef` 命令创建自定义类型，等同于 TypeScript 里面的类型别名。

javascript

```
/**
 * @typedef {(number | string)} NumberLike
 */
```

上面示例中，定义了一个名为 `NumberLike` 的新类型，它是由 `number` 和 `string` 构成的联合类型，等同于 TypeScript 的如下语句。

typescript

```
type NumberLike = string | number;
```

@type

`@type` 命令定义变量的类型。

javascript

```
/**
 * @type {string}
 */
let a;
```

上面示例中， `@type` 定义了变量 `a` 的类型为 `string` 。

在 `@type` 命令中可以使用由 `@typedef` 命令创建的类型。

javascript

```
/**
 * @typedef {(number | string)} NumberLike
 */
```

```javascript
/**
 * @type {NumberLike}
 */
let a = 0;
```

在 `@type` 命令中允许使用 TypeScript 类型及其语法。

```javascript
/**@type {true | false} */
let a;

/** @type {number[]} */
let b;

/** @type {Array<number>} */
let c;

/** @type {{ readonly x: number, y?: string }} */
let d;

/** @type {(s: string, b: boolean) => number} */
let e;
```

@param

`@param` 命令用于定义函数参数的类型。

```javascript
/**
 * @param {string}  x
 */
function foo(x) {}
```

如果是可选参数，需要将参数名放在方括号 `[]` 里面。

```javascript
/**
 * @param {string}  [x]
 */
function foo(x) {}
```

方括号里面，还可以指定参数默认值。

```javascript
/**
 * @param {string} [x="bar"]
 */
function foo(x) {}
```

上面示例中，参数 `x` 的默认值是字符串 `bar` 。

@return，@returns

`@return` 和 `@returns` 命令的作用相同，指定函数返回值的类型。

```javascript
/**
 * @return {boolean}
 */
function foo() {
  return true;
}

/**
 * @returns {number}
 */
function bar() {
  return 0;
}
```

@extends 和类型修饰符

`@extends` 命令用于定义继承的基类。

```javascript
/**
 * @extends {Base}
 */
class Derived extends Base {}
```

`@public` 、 `@protected` 、 `@private` 分别指定类的公开成员、保护成员和私有成员。

`@readonly` 指定只读成员。

javascript

```
class Base {
  /**
   * @public
   * @readonly
   */
  x = 0;

  /**
   *  @protected
   */
  y = 0;
}
```


最后更新: 2023/8/13 15:25

Previous page
类型工具

Next page
tsconfig.json 文件