declare 关键字

简介

declare 关键字用来告诉编译器，某个类型是存在的，可以在当前文件中使用。

它的主要作用，就是让当前文件可以使用其他文件声明的类型。举例来说，自己的脚本使用外部库定义的函数，编译器会因为不知道外部函数的类型定义而报错，这时就可以在自己的脚本里面使用 `declare` 关键字，告诉编译器外部函数的类型。这样的话，编译单个脚本就不会因为使用了外部类型而报错。

declare 关键字可以描述以下类型。

- 变量（const、let、var 命令声明）
- type 或者 interface 命令声明的类型
- class
- enum
- 函数（function）
- 模块（module）
- 命名空间（namespace）

declare 关键字的重要特点是，它只是通知编译器某个类型是存在的，不用给出具体实现。比如，只描述函数的类型，不给出函数的实现，如果不使用 `declare` ，这是做不到的。

declare 只能用来描述已经存在的变量和数据结构，不能用来声明新的变量和数据结构。另外，所有 declare 语句都不会出现在编译后的文件里面。

declare variable

declare 关键字可以给出外部变量的类型描述。

举例来说，当前脚本使用了其他脚本定义的全局变量 `x` 。

typescript

```
x = 123; // 报错
```

上面示例中，变量 `x` 是其他脚本定义的，当前脚本不知道它的类型，编译器就会报错。

这时使用 declare 命令给出它的类型，就不会报错了。

typescript

```
declare let x: number;
x = 1;
```

如果 declare 关键字没有给出变量的具体类型，那么变量类型就是 `any` 。

typescript

```
declare let x;
x = 1;
```

上面示例中，变量 `x` 的类型为 `any` 。

下面的例子是脚本使用浏览器全局对象 `document` 。

typescript

```
declare var document;
document.title = "Hello";
```

上面示例中，declare 告诉编译器，变量 `document` 的类型是外部定义的（具体定义在 TypeScript 内置文件 `lib.d.ts` ）。

如果 TypeScript 没有找到 `document` 的外部定义，这里就会假定它的类型是 `any` 。

注意，declare 关键字只用来给出类型描述，是纯的类型代码，不允许设置变量的初始值，即不能涉及值。

typescript

```
// 报错
declare let x: number = 1;
```

上面示例中，declare 设置了变量的初始值，结果就报错了。

declare function

declare 关键字可以给出外部函数的类型描述。

下面是一个例子。

```typescript
declare function sayHello(name: string): void;

sayHello("张三");
```

上面示例中，declare 命令给出了 `sayHello()` 的类型描述，因此可以直接使用它。

注意，这种单独的函数类型声明语句，只能用于 `declare` 命令后面。一方面，TypeScript 不支持单独的函数类型声明语句；另一方面，declare 关键字后面也不能带有函数的具体实现。

```typescript
// 报错
function sayHello(name: string): void;
function sayHello(name) {
  return "你好，" + name;
}
```

上面示例中，单独写函数的类型声明就会报错。

declare class

declare 给出 class 的描述描述写法如下。

```typescript
declare class Animal {
  constructor(name: string);
  eat(): void;
  sleep(): void;
}
```

下面是一个复杂一点的例子。

typescript

```typescript
declare class C {
  // 静态成员
  public static s0(): string;
  private static s1: string;

  // 属性
  public a: number;
  private b: number;

  // 构造函数
  constructor(arg: number);

  // 方法
  m(x: number, y: number): number;

  // 存取器
  get c(): number;
  set c(value: number);

  // 索引签名
  [index: string]: any;
}
```

同样的，declare 后面不能给出 Class 的具体实现或初始值。

---

declare module，declare namespace

如果想把变量、函数、类组织在一起，可以将 declare 与 module 或 namespace 一起使用。

typescript

```typescript
declare namespace AnimalLib {
  class Animal {
    constructor(name: string);
    eat(): void;
    sleep(): void;
  }

  type Animals = "Fish" | "Dog";
}
```

```typescript
// 或者
declare module AnimalLib {
  class Animal {
    constructor(name: string);
    eat(): void;
    sleep(): void;
  }

  type Animals = "Fish" | "Dog";
}
```

上面示例中，declare 关键字给出了 module 或 namespace 的类型描述。

declare module 和 declare namespace 里面，加不加 export 关键字都可以。

typescript

```typescript
declare namespace Foo {
  export var a: boolean;
}

declare module "io" {
  export function readFile(filename: string): string;
}
```

上面示例中，namespace 和 module 里面使用了 export 关键字。

下面的例子是当前脚本使用了 `myLib` 这个外部库，它有方法 `makeGreeting()` 和属性 `numberOfGreetings` 。

typescript

```typescript
let result = myLib.makeGreeting("你好");
console.log("欢迎词: " + result);
let count = myLib.numberOfGreetings;
```

`myLib` 的类型描述就可以这样写。

typescript

```typescript
declare namespace myLib {
  function makeGreeting(s: string): string;
  let numberOfGreetings: number;
}
```

declare 关键字的另一个用途，是为外部模块添加属性和方法时，给出新增部分的类型描述。

```typescript
import { Foo as Bar } from "moduleA";

declare module "moduleA" {
  interface Bar extends Foo {
    custom: {
      prop1: string;
    };
  }
}
```

上面示例中，从模块 `moduleA` 导入了 `Foo` 接口，将其重命名为 `Bar` ，并用 declare 关键字为 `Bar` 增加一个属性 `custom` 。

下面是另一个例子。一个项目有多个模块，可以在一个模型中，对另一个模块的接口进行类型扩展。

```typescript
// a.ts
export interface A {
  x: number;
}

// b.ts
import { A } from "./a";

declare module "./a" {
  interface A {
    y: number;
  }
}

const a: A = { x: 0, y: 0 };
```

上面示例中，脚本 `a.ts` 定义了一个接口 `A` ，脚本 `b.ts` 为这个接口添加了属性 `y` 。 `declare module './a' {}` 表示对 `a.ts` 里面的模块，进行类型声明，而同名 interface 会自动合并，所以等同于扩展类型。

使用这种语法进行模块的类型扩展时，有两点需要注意：

（1） `declare module NAME` 语法里面的模块名 `NAME` ，跟 import 和 export 的模块名规则是一样的，且必须跟当前文件加载该模块的语句写法（上例 `import { A } from './a'` ）保持一致。

（2）不能创建新的顶层类型。也就是说，只能对 `a.ts` 模块中已经存在的类型进行扩展，不允许增加新的顶层类型，比如新定义一个接口 `B` 。

（3）不能对默认的 `default` 接口进行扩展，只能对 export 命令输出的命名接口进行扩充。这是因为在进行类型扩展时，需要依赖输出的接口名。

某些第三方模块，原始作者没有提供接口类型，这时可以在自己的脚本顶部加上下面一行命令。

typescript

```
declare module "模块名";

// 例子
declare module "hot-new-module";
```

加上上面的命令以后，外部模块即使没有类型，也可以通过编译。但是，从该模块输入的所有接口都将为 `any` 类型。

declare module 描述的模块名可以使用通配符。

typescript

```
declare module "my-plugin-*" {
  interface PluginOptions {
    enabled: boolean;
    priority: number;
  }

  function initialize(options: PluginOptions): void;
  export = initialize;
}
```

上面示例中，模块名 `my-plugin-*` 表示适配所有以 `my-plugin-` 开头的模块名（比如 `my-plugin-logger` ）。

---

declare global

如果要为 JavaScript 引擎的原生对象添加属性和方法，可以使用 `declare global {}` 语法。

typescript

```typescript
export {};

declare global {
  interface String {
    toSmallString(): string;
  }
}

String.prototype.toSmallString = (): string => {
  // 具体实现
  return "";
};
```

上面示例中，为 JavaScript 原生的 `String` 对象添加了 `toSmallString()` 方法。declare global 给出这个新增方法的类型描述。

这个示例第一行的空导出语句 `export {}` ，作用是强制编译器将这个脚本当作模块处理。这是因为 `declare global` 必须用在模块里面。

下面的示例是为 window 对象添加一个属性 `myAppConfig` 。

typescript

```typescript
export {};

declare global {
  interface window {
    myAppConfig: object;
  }
}

const config = window.myAppConfig;
```

declare global 只能扩充现有对象的类型描述，不能增加新的顶层类型。

---

declare enum

declare 关键字给出 enum 类型描述的例子如下，下面的写法都是允许的。

typescript

```
declare enum E1 {
  A,
  B,
}

declare enum E2 {
  A = 0,
  B = 1,
}

declare const enum E3 {
  A,
  B,
}

declare const enum E4 {
  A = 0,
  B = 1,
}
```

---

declare module 用于类型声明文件

我们可以为每个模块脚本，定义一个 `.d.ts` 文件，把该脚本用到的类型定义都放在这个文件里面。但是，更方便的做法是为整个项目，定义一个大的 `.d.ts` 文件，在这个文件里面使用 `declare module` 定义每个模块脚本的类型。

下面的示例是 `node.d.ts` 文件的一部分。

typescript

```
declare module "url" {
  export interface Url {
    protocol?: string;
    hostname?: string;
    pathname?: string;
  }

  export function parse(
    urlStr: string,
    parseQueryString?,
    slashesDenoteHost?
```

```
  ): Url;
}

declare module "path" {
  export function normalize(p: string): string;
  export function join(...paths: any[]): string;
  export var sep: string;
}
```

上面示例中， `url` 和 `path` 都是单独的模块脚本，但是它们的类型都定义在 `node.d.ts` 这个文件里面。

使用时，自己的脚本使用三斜杠命令，加载这个类型声明文件。

typescript

```
/// <reference path="node.d.ts"/>
```

如果没有上面这一行命令，自己的脚本使用外部模块时，就需要在脚本里面使用 declare 命令单独给出外部模块的类型。

参考链接

- How Does The Declare Keyword Work In TypeScript?, Tim Mouskhelichvili

最后更新: 2023/8/13 15:25

Previous page
装饰器（旧语法）

Next page
d.ts 类型声明文件