简介

这篇指南的目的是教你如何书写高质量的TypeScript声明文件。
我们在这里会展示一些API的文档，还有它们的使用示例，
并且阐述了如何为它们书写声明文件。

这些例子是按复杂度递增的顺序组织的。

例子

全局变量

文档

全局变量`foo`包含了存在组件总数。

代码

```
console.log("Half the number of widgets is " + (foo / 2));
```

声明

使用`declare var`声明变量。
如果变量是只读的，那么可以使用`declare const`。
你还可以使用`declare let`如果变量拥有块级作用域。

```
/** 组件总数 */
declare var foo: number;
```

全局函数

文档

用一个字符串参数调用`greet`函数向用户显示一条欢迎信息。

代码

```
greet("hello, world");
```

声明

使用`declare function`声明函数。

```
declare function greet(greeting: string): void;
```

带属性的对象

文档

> 全局变量`myLib`包含一个`makeGreeting`函数，
> 还有一个属性`numberOfGreetings`指示目前为止欢迎数量。

代码

```
let result = myLib.makeGreeting("hello, world");
console.log("The computed greeting is:" + result);

let count = myLib.numberOfGreetings;
```

声明

使用`declare namespace`描述用点表示法访问的类型或值。

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

函数重载

文档

> `getWidget`函数接收一个数字，返回一个组件，或接收一个字符串并返回一个组件数组。

代码

```
let x: Widget = getWidget(43);

let arr: Widget[] = getWidget("all of them");
```

声明

```
declare function getWidget(n: number): Widget;
declare function getWidget(s: string): Widget[];
```

可重用类型（接口）

文档

> 当指定一个欢迎词时，你必须传入一个`GreetingSettings`对象。
> 这个对象具有以下几个属性：

1- greeting：必需的字符串
2- duration: 可靠的时长（毫秒表示）
3- color: 可选字符串，比如‘#ff00ff’

代码

```
greet({
  greeting: "hello world",
  duration: 4000
});
```

声明

使用`interface`定义一个带有属性的类型。

```
interface GreetingSettings {
  greeting: string;
  duration?: number;
  color?: string;
}

declare function greet(setting: GreetingSettings): void;
```

可重用类型（类型别名）

文档

在任何需要欢迎词的地方，你可以提供一个`string`，一个返回`string`的函数或一个`Greeter`实例。

代码

```
function getGreeting() {
    return "howdy";
}
class MyGreeter extends Greeter { }

greet("hello");
greet(getGreeting);
greet(new MyGreeter());
```

声明

你可以使用类型别名来定义类型的短名：

```
type GreetingLike = string | (() => string) | MyGreeter;

declare function greet(g: GreetingLike): void;
```

组织类型

文档

`greeter`对象能够记录到文件或显示一个警告。

你可以为`.log(...)`提供LogOptions和为`.alert(...)`提供选项。

代码

```
const g = new Greeter("Hello");
g.log({ verbose: true });
g.alert({ modal: false, title: "Current Greeting" });
```

声明

使用命名空间组织类型。

```
declare namespace GreetingLib {
    interface LogOptions {
        verbose?: boolean;
    }
    interface AlertOptions {
        modal: boolean;
        title?: string;
        color?: string;
    }
}
```

你也可以在一个声明中创建嵌套的命名空间：

```
declare namespace GreetingLib.Options {
    // Refer to via GreetingLib.Options.Log
    interface Log {
        verbose?: boolean;
    }
    interface Alert {
        modal: boolean;
        title?: string;
        color?: string;
    }
}
```

类

文档

你可以通过实例化`Greeter`对象来创建欢迎词，或者继承`Greeter`对象来自定义欢迎词。

代码

```
const myGreeter = new Greeter("hello, world");
myGreeter.greeting = "howdy";
myGreeter.showGreeting();

class SpecialGreeter extends Greeter {
    constructor() {
        super("Very special greetings");
    }
}
```

声明

使用`declare class`描述一个类或像类一样的对象。
类可以有属性和方法，就和构造函数一样。

```
declare class Greeter {
    constructor(greeting: string);

    greeting: string;
    showGreeting(): void;
}
```