书写声明文件之砍柴：为不同类型库书写声明文件

更新时间：2019-07-22 10:24:18

> 与有肝胆人共事，从无字句处读书。
>
> ——周恩来

5.3.1 模块插件或 UMD 插件

一些模块和插件是支持插件机制的，比如我们常见的 jQuery，它的插件有非常多。我们可以为库书写声明文件的同时，为库的插件定义声明文件，可以参考官方模板 module-plugin.d.ts。

5.3.2 全局插件

全局插件往往会修改全局中一些对象，在这些对象上添加或修改属性方法，比如下面的示例：

```
// add-methods-to-string.js
String.prototype.getFirstLetter = function() {
  return this[0];
};
```

这段代码在 String 构造函数的原型对象上添加一个 getFirstLetter 方法，这个方法可以返回字符串的第一个字符。我们创建一个字符串，就可以调用这个方法。来讲下这个原理，我们在 String 构造函数原型对象上添加一个方法，这个方法就会被 String 创建的实例继承，如果我们使用`new String('Lison')`的方式创建一个实例 name，那么这个 name 将是一个对象类型的值，它的属性是 0 开始到 n 的数字，属性值对应字符串的第 1 个、第 n 个字符；但是像例子中这样使用`const name = 'Lison'`字面量的形式定义的 name，其实是个字符串类型的值，字符串就不会继承构造函数的方法了，以为它不是对象，但事实是它可以调用`getFirstLetter`方法。这是因为它在调用方法的时候，会先将这个字符串包装成一个封装对象，在内部即使用 String 构造函数，所以它依然可以调用原型对象上的方法。

我们在 html 文件里引入这个 js 文件后创建一个字符串，这个字符串就可以调用 getFirstLetter 方法：

```
<script type="text/javascript" src="./add-methods-to-string.js"></script>
<script type="text/javascript">
  var str = "Lison";
  console.log(str.getFirstLetter()); // "L"
</script>
```

如果我们在 TS 中使用，就需要为这个创建声明文件，我们创建一个声明文件 global-plugin.d.ts：

```
// global-plugin.d.ts
interface String {
  getFirstLetter(): number;
}
// index.ts
var str = "Lison";
console.log(str.getFirstLetter()); // "L"
```

遇到这类情景，你可以参考官方的 global-plugin.d.ts 模板来书写声明文件。

5.3.3 修改全局的模块

还有一些影响全局的全局模块，这些模块除了导出一些东西，还会直接修改全局的一些对象，我们还是使用上面例子的代码，只不过这次我们使用引入模块的形式来引入，：

```
// add-methods-to-string模块
String.prototype.getFirstLetter = function() {
  return this[0];
};
// index.js
require("add-methods-to-string");
const name = "Lison";
console.log(name.getFirstLetter()); // "L"
```

通过这个例子我们知道一些全局模块是做什么事了，你大概知道怎么为它们定义声明文件了。我们新建一个声明文件 global-modifying-module.d.ts，在声明文件中如下声明：

```
declare global {
  interface String {
    getFirstLetter(): number;
  }
}
export {};
```

注意如果我们这个声明文件没有需要导出的东西，这里必须在末尾加上 `export {}`，这样才能让 TS 编译器把这个声明文件当做一个模块声明。我们加了这个声明文件后，就可以在 TS 中引入这个模块，再在 TS 中调用字符串的 getFirstLetter 方法就不会报错了。这类全局模块，你可以参考官方的 global-modifying-module.d.ts 模板。

5.3.4 使用依赖

库多数会依赖其它库，所以可以在定义库声明文件的时候，声明对其它库的依赖，从而加载其它库的内容。如果是依赖全局库，可以使用 `///<reference types="UMDModuleName" />` 三斜线指令来指定加载了某个全局库：

```
/// <reference types="globalLib" />
function func(): globalLib.someName;
```

如果依赖的是模块库，可以使用 import 语句：

```
import * as moment from "moment";
function func(): moment;
```

因为有些库是没有 default 默认输出的，所以如果你在使用 `import xxx from 'xxx'` 语句引入一个库报错时，可以使用 `import * as xxx from 'xxx'` 的形式引入。

如果是全局库依赖于某个 UMD 模块，也可以使用 `///<reference types="UMDModuleName" />` 三斜线指令来指定对某个 UMD 模块的依赖：

```
// globals.d.ts
/// <reference types="moment"/>
function getMoment(): moment;
```

如果模块或一个 UMD 库依赖于一个 UMD 库，使用 `import * as` 语句引入模块：

```
// module.d.ts
import * as moment from "moment";
export default function(m: typeof moment): void;
```

最后有三点要注意的：

- 防止命名冲突

我们在写全局声明时，在全局范围定义大量类型，有时会导致命名冲突。所以建议相关的定义放到命名空间内。

- ES6 模块插件影响

一些开发者为一些库开发了插件，用在原有库的基础上添加更多功能，这往往需要修改原有库导出的模块。我们在讲模块的时候说过，ES6 模块标准中，导出的模块是不允许修改的；但是在 CommonJS 和其它一些加载器里是允许的，所以要使用 ES6 模块的话要注意这一点。

- ES6 模块调用

我们在使用一些库的时候，引入的模块可以作为函数直接调用。ES6 的模块顶层对象是一个对象，它不能作为函数调用，比如我们直接用 export 导出几个内容：

```
// moduleB.js
export const age = 10;
export let name = "lison";
// main.js
import info from "./moduleB.js";
console.log(info.name); // 'lison'
// index.js
import { name, age } from "./moduleB.js";
console.log(name); // 'lison'
```

如果我们想导出一个直接可以调用的函数，又要使用 ES6 模块，则可以用 export default 来导出一个函数，这个我们在讲模块的时候也讲过了。

5.3.5 快捷外部模块声明

如果我们使用一个新模块不想花时间精力为这个模块写声明，TS 在 2.0 版本支持了快捷外部模块声明，比如我们使用 moment 模块，就可以在 typings 创建一个 moment 文件夹，并在这个 moment 文件夹创建一个 index.d.ts 文件，写如下内容：

```
// index.d.ts
declare module "moment";
```

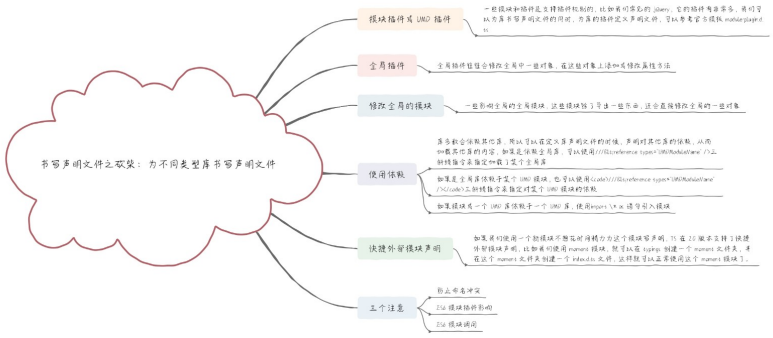
这样就可以正常使用这个 moment 模块了。

本节小结

本小节我们学习了如何为各种类型的库书写声明文件，官方提供了各种类型库的声明文件的模板，总结如下：

- global-modifying-module.d.ts：适合修改全局的模块。
- global-plugin.d.ts：适合全局插件。
- global.d.ts：适合全局库。
- module-class.d.ts：适合引入后可以直接当做类使用new关键字创建实例的模块。
- module-function.d.ts：适合引入后可以直接当做函数的模块，
- module-plugin.d.ts：适合模块插件或UMD插件。
- module.d.ts：适合引入后既不能当做类直接使用，也不能直接当做函数调用的模块。

后面我们的实战章节还会运用到书写声明文件的知识，我们可以再在实际开发中体验一下。到这里本章就学习完了，我们的语法和基础知识部分就学习完了，接下来的一章内容将是实战章节，我们将结合使用Vue+TypeScript开发一个实际的项目，来更好地学习如何运用TypeScript进行项目开发。



书写声明文件之磨刀：识别库类型

搭建基础项目

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论