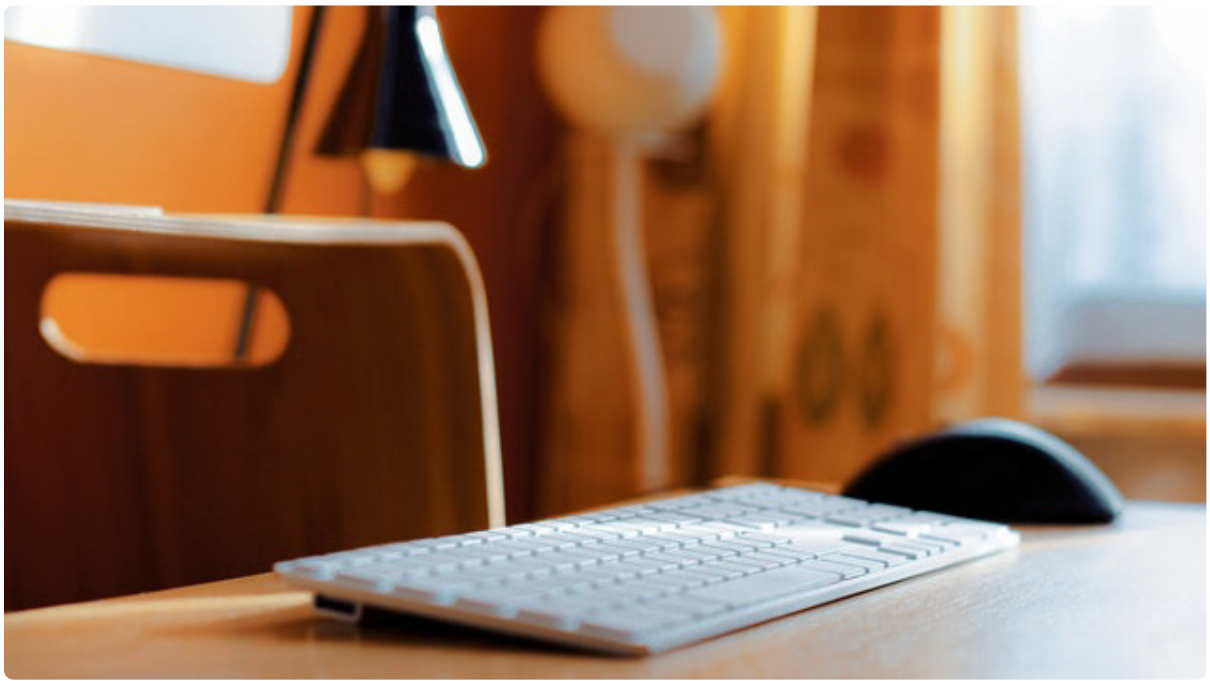
书写声明文件之磨刀：识别库类型

更新时间：2019-07-19 14:03:18



> 每个人的生命都是一只小船，理想是小船的风帆。
>
> ——张海迪

前面我们提到几次 `.d.ts` 后缀的文件，这节课我们来完整学习下与声明文件相关的内容。

我们之前讲模块的时候讲到过两种常见模块标准，即 CommonJS 和 RequireJS。不同的模块在实现方式上是不一样的。我们要为已有的第三方 JS 库编写声明文件，以便在 TS 中更好地使用类型系统，所以首先需要知道我们使用的 JS 库被编译成了什么类型。我们来分别看下几种类型的特征：

5.2.1. 全局库

在一开始，没有 webpack 这些编译工具的时候，我们都是在 html 文件里使用 script 标签引入 js 文件，然后就可以在引入的后面使用引入的库了。比如我们使用 jQuery，只需要在 `<body>` 标签里通过 `<script src="http://xxx.com/jQuery.min.js"></script>` 引入 jQuery 库，然后就可以在 `<script></script>` 标签内使用：

```
$(function() {
  // ...
});
```

这种不需要我们引入什么变量，只需要将库引入即可使用的库，就叫做**全局库**。后面讲到 UMD 模块的时候要注意，UMD 模块既可以作为模块使用，又可以作为全局库使用的模块，所以在判断一个库的时候，如果它可以像例子中那样全局使用，首先要确定它是不是 UMD 模块；如果不是，那它可能就是一个单纯的全局库。

另外，你还可以通过看库的源码，来判断它是什么类型，一个全局库，通常会包含下面内容中的一个或多个：

- 顶级的 var 语句或 function 声明；
- 一个或多个赋值给 window.someName 的赋值语句；
- 判断 document 或 window 是否存在的判断逻辑。

因为顶级的 var 或 function 是直接在全局环境声明变量或函数，不使用立即执行函数包裹会影响到全局，所以有这种一般会是全局库；当出现给 window 设置某个属性名 someName ，然后给这个属性赋值的语句时，是在给全局对象 window 赋值。引入这个库后直接通过 window.someName 即可在全局任何地方访问到这个属性值；如果出现 if 语句或三元操作符这种判断 document 或 window 是否存在的语句，也有可能是要给这两个全局对象添加内容，所以也有可能是全局库。

但是由于把一个全局库转变成 UMD 库较为容易，所以现在全局库较少。

我们来着手为下面这个简单的示例全局库编写一个全局库声明文件，先来看这个全局库的代码：

```
// handle-title.js
function setTitle(title) {
  document && (document.title = title);
}

function getTitle() {
  return (document && document.title) || "";
}

let documentTitle = getTitle();
```

这个 handle-title.js 库非常简单，声明了一个 setTitle 函数，接收一个参数，在函数内首先通过&&符判断 document 是否不为 undefined，如果不为 undefined，才会执行后面的逻辑，将 title 赋值给 document.title，从而实现修改显示在浏览器标签的文字；一个 getTitle 函数，用于获取此时的 title 值，如果没有 document 对象，则返回空字符串；一个全局变量 documentTitle，用来在初始化时记录当前的 title。

我们要为这个 handle-title.js 全局库编写一个声明文件*handle-title.d.ts*，官方为每一种库类型都提供了响应的声明文件模板，全局库的模板是*global.d.ts*，我们首先来看一下这个模板中的内容：

```
// 如果这个库有一个全局暴露的函数，他可能可以传入不同类型的参数，返回不同的值，所以可以为它定义函数重载
declare function myLib(a: string): string;
declare function myLib(a: number): number;
// 如果你想让这个库名作为一种类型，可以定义一个接口
declare interface myLib {
  name: string;
  length: number;
  extras?: string[];
}
// 如果这个库有一些需要在全局暴露的属性，可以定义这个命名空间，将值、接口和类型别名等定义在这里
// 这样，在下面命名空间中没有列出的内容，通过myLib.xxx访问时在编译阶段会报错，但是运行时是可以访问的，只要这个JS库里定义了。
declare namespace myLib {
  let timeout: number; // 通过myLib.timeout访问，也可以修改: myLib.timeout = 123
  const version: string; // 可通过myLib.version访问，但不能修改，因为是const声明的
  class Cat {
    constructor(n: number);
    readonly age: number;
    purr(): void;
  }
  interface CatSettings {
    weight: number;
    name: string;
    tailLength?: number;
  }
  type VetID = string | number;
  function checkCat(c: Cat, s?: VetID);
}
```

这个*handle-title.js*文件我们可以直接在 index.html 文件里引入，如果不定义声明文件，我们直接在 index.ts 里使用，会报错：

```
console.log(getTitle()); // error 找不到名称“getTitle”
console.log(documentTitle); // error 找不到名称“documentTitle”
```

接下来我们为 handle-title.js 库编写一个声明文件：

```
// handle-title.d.ts
declare function setTitle(title: string | number): void;

declare function getTitle(): string;

declare let documentTitle: string;
```

我们在 tsconfig.json 里，通过设置 include 来让编译器自动引入`"./src/"`文件夹下的所有声明文件：

```
"include": [
    "./src/**/*.ts",
    "./src/**/*.d.ts"
]
```

这样我们定义在`src/types`文件夹下的所有声明文件就会起作用了，这下再看 index.ts 文件里使用 getTitle 和 documentTitle 就没有问题了。

5.2.2. 模块化库

模块化库即依赖模块解析器的库。之前讲模块的时候讲到过 CommonJS 和 ES6 模块，接下来我们看下如何判断一个库是模块化库。在模块库代码中，你一般会看到下面的情况之一：

- 无条件地调用 require 或 define 方法；
- 像 import * as a from 'b'或者 export c 这样的声明；
- 赋值给 exports.someName 或 module.exports。

因为模块化库依赖模块解析器环境，在使用这种库的时候，就已经引入模块解析器的 require 或 define 等方法了，所以模块化库会直接调用这些方法来加载代码；库中包括 import * as a from 'b'和 export c 这种模块中才有的引入和导出语句的话，基本就是模块库了；如果有赋值语句赋值给 exports.someName 或 module.exports，这种就是 CommonJS 模块的导出语句了。

你极少会在模块化库中看到*对 window*或*global*的赋值，当然这不是绝对的，比如有的库需要操作 window 的一些属性，这就难免了。

针对模块，官方有三个模板声明文件，分别是 module.d.ts、module-class.d.ts 和 module-function.d.ts：

- 如果这个模块引入后，可以直接当做函数调用，那可以参考 module-function.d.ts 文件；
- 如果模块引入后，可以直接当做类使用 new 关键字创建实例，可以参考 module-class.d.ts 文件；
- 如果模块不能被调用也不能当做类，参考 module.d.ts。

关于这几种模板，以及其它类型库声明文件的书写，我们会在后面实战课中通过几个实际的例子来进一步学习。

5.2.3. UMD 库

UMD 库将全局库和模块库的功能进行了结合，它会先判断环境中有没有模块加载器的一些特定方法。如果有，说明是模块加载器环境，UMD 库就会使用模块的方式导出；如果没有检测到这些方法，则会将内容添加到全局环境。一般你会在 UMD 库中看到这种逻辑判断：

```
(function(root, factory) {
  if (typeof define === "function" && define.amd) {
    define(["libName"], factory);
  } else if (typeof module === "object" && module.exports) {
    module.exports = factory(require("libName"));
  } else {
    root.returnExports = factory(root.libName);
  }
})(this, function(b) {
  // ...
});
```

现在很多库都是 UMD 库，比如 jQuery、moment 等，你既可以在 html 文件中直接通过<script>标签引入它，也可以通过模块的形式引入。

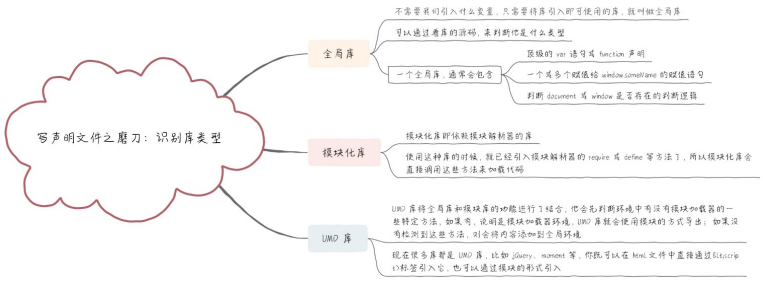
本节小结

本小节我们学习了如何识别库的类型，我们讲了三类库的识别：全局库、模块化库和UMD库，这三类都是较为常见的，基本上所有的库都是这三类中的一种。每种类型的库都有它们的特点：

- 全局库：顶级的var或function声明语句，给window添加属性，判断document或window是否存在的判断逻辑；
- 模块化库：无条件地调用require或define方法，使用import或export导入和导出内容，赋值给exports或module.exports；
- UMD库，判断`typeof define === “function” && define.amd`的逻辑，判断`typeof module === “object” && module.exports`的逻辑。

判断库的类型，可以首先看文档，看支持怎么使用，然后是看代码。使用这些方法，你足以判断出一个库的类型。

下个小节我们就要正式开始学习为第三方库写声明文件了。



逐条来看tsconfig.json配置

书写声明文件之砍柴：为不同类型库书写声明文件

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

慕数据7345090

老师，怎么引入handle-title.js呢，怎么引入本地文件？webpack怎么设置呢

0 回复 2019-07-24