24讲前端语言支持： JavaScript和Node.js

在专栏的前两部分基础篇和工作区，我们一起学习了 VS Code 的各种不同程度的语言支持，从**代码片段、自动补全、快速修复、重构**，到**代码调试、任务管理**，等等。在介绍这些功能时，我都是拿 JavaScript 来举例的，你可能会问，JavaScript 有什么特别之处吗？

VS Code 团队本身就是 JavaScript 的使用者，同时 VS Code 还是 TypeScript 项目非常早期的用户，可以说是和 TypeScript 一起成长起来的。无论是 VS Code 还是 TypeScript 团队，都极其重视 VS Code 上的 JavaScript 使用体验，因为这是他们工作的很大一部分。

而你应该记得我在前面的文章中也介绍过，VS Code 的各种语言功能，都是以 API 的形式开放给各个插件的，VS Code 的 JavaScript/TypeScript 也是一个插件。所以 VS Code 的各种新功能、新特性，JavaScript/TypeScript 往往就是试验田，而 JavaScript/TypeScript 插件的用户反馈也会第一时间被 VS Code 团队分析研究。

今天，我会先带领你快速回顾一下 VS Code 中 JavaScript 的各种基础语言支持，这些功能在前面的专栏文章里都作过介绍。然后，我们一起探讨下，VS Code 是如何为 JavaScript 提供语言支持的，以及如果你想更好地让 VS Code 理解和分析你的 JavaScript 项目，你可以怎么做。

同时，我还会介绍如何把自己的项目渐进地使用上 TypeScript 的语言服务，以及VS Code 里做了哪些关于 Node.js 的优化。

好了，废话不多说，让我们开始今天的内容。这里，我们先创建一个新的文件夹，然后在 VS Code 中打开。

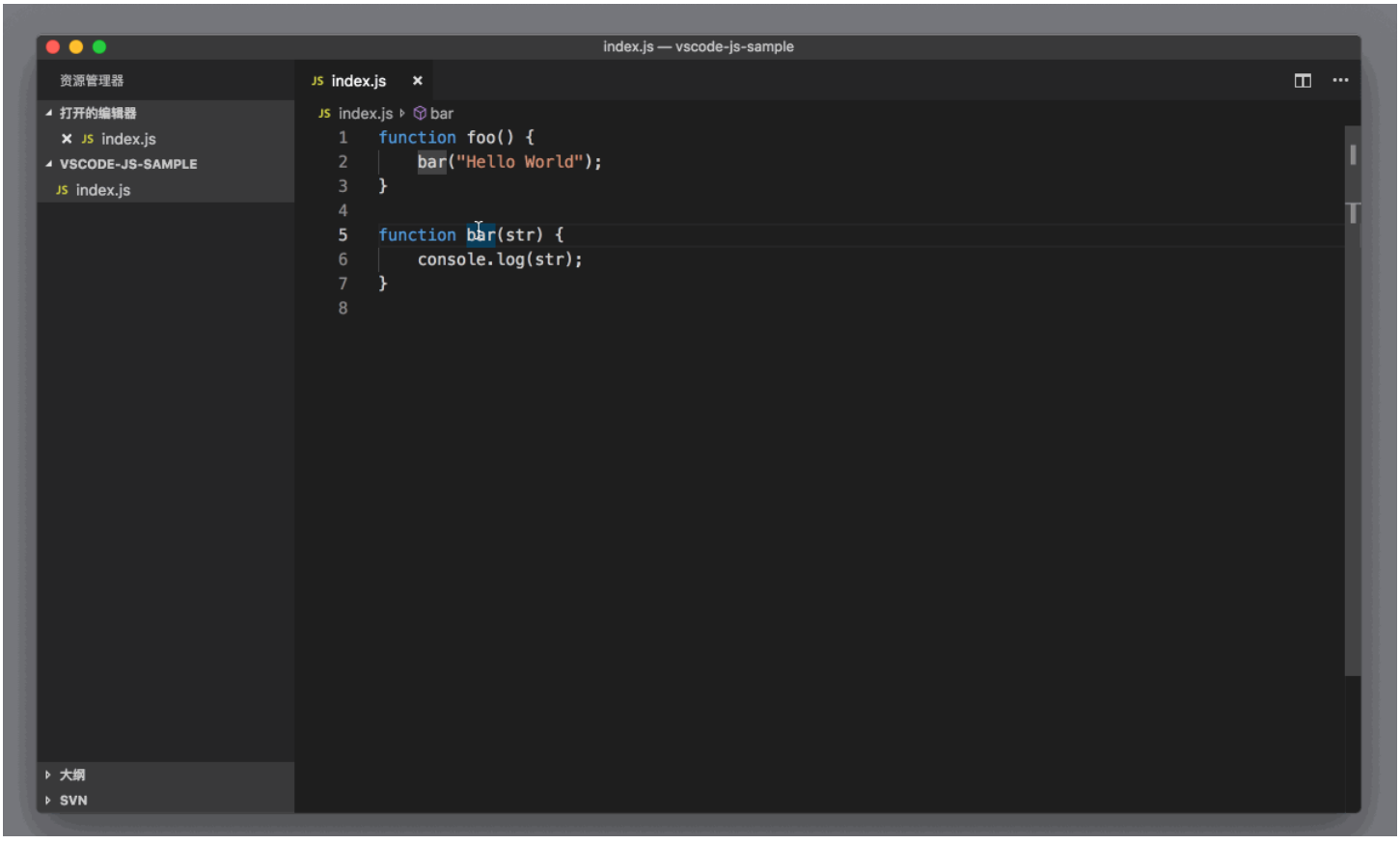
基础语言支持

接下来我们在这个文件夹下创建一个 JavaScript 文件 index.js，内容如下：

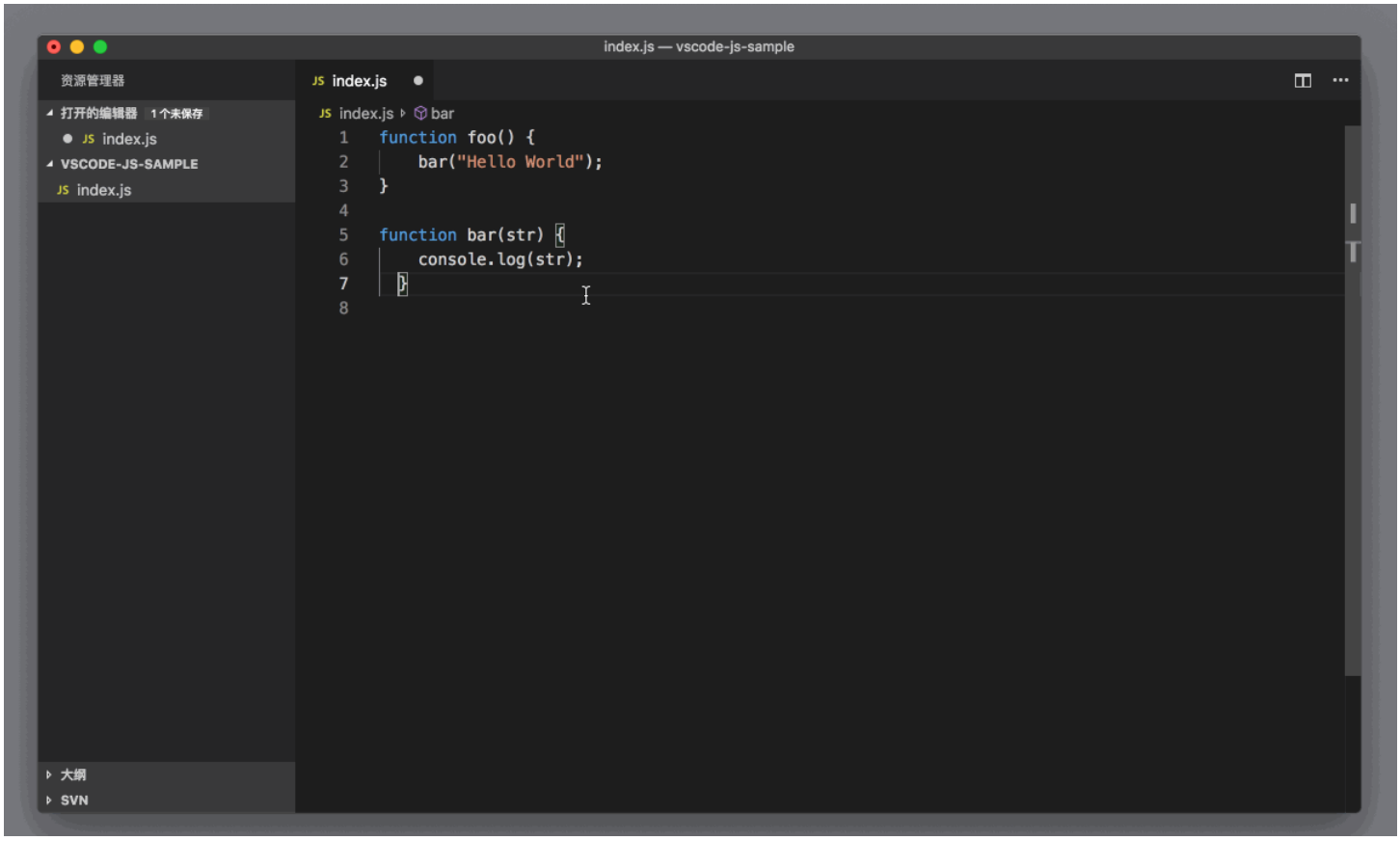
```
function foo() {
    bar("Hello World");
}


function bar($tr) {
    console.log($tr);
}
```

这段 JavaScript 代码中定义两个函数 foo 和 bar，其中 foo 函数内部调用了 bar 这个函数。根据我们之前学习的知识，我们可以使用下面这些命令：

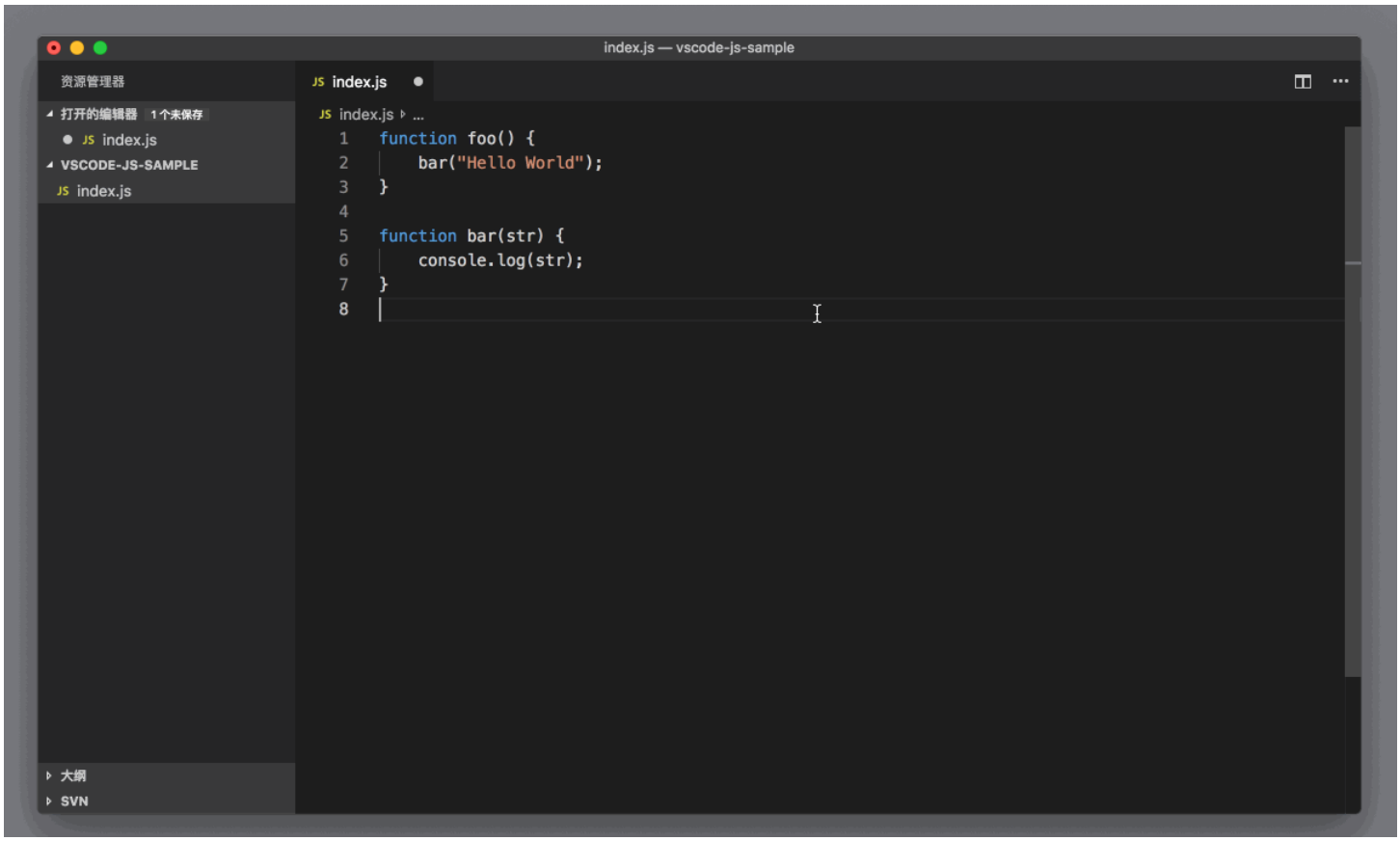
1、转到定义（F12）



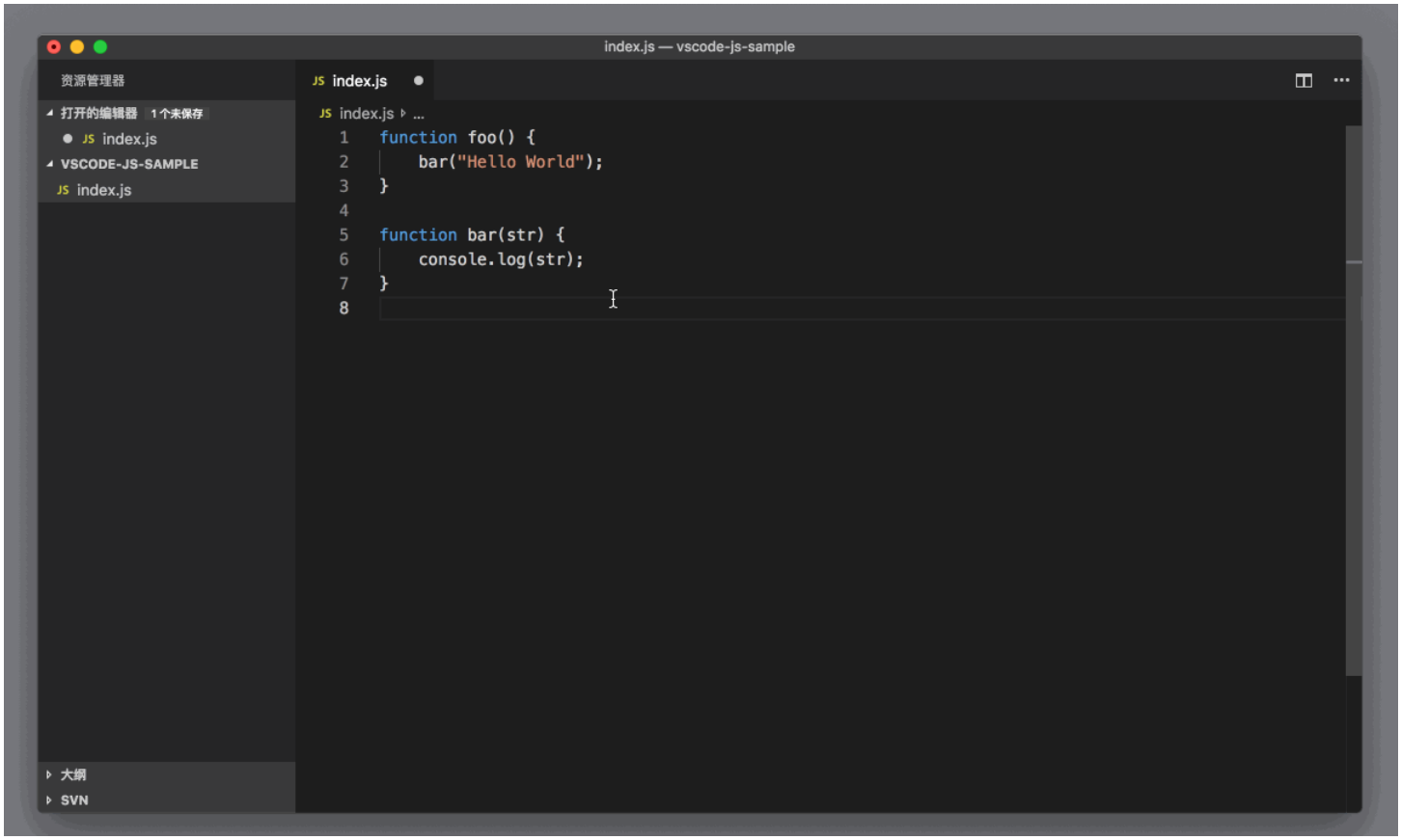
2、格式化文件（Format Document）



3、符号跳转（Cmd + Shift + O）



4、函数建议列表和参数建议



如果你是跟着专栏一章一章学习过来的话，相信你已经非常熟悉这些操作了。

类型提示

我们知道 JavaScript 是一门动态类型语言，一个对象的类型在最终运行的时候才会决定和进行检测。而 TypeScript 这门语言的出现，允许我们在书写和编译代码时，就对代码中对象的类型和使用进行规范和约束，以降低因类型错误而导致的 bug。尤其是当项目逐渐庞大后，拥有一个比较完善的类型体系，无论是对于代码质量，还是代码的维护难度，都有很多好处。

到这里你可能会问，“那是不是说，如果要使用上类型系统，我就一定得把项目全部用 TypeScript 重写呢？”答案当然是：不用。这里我教你两招。

第一招是使用工具JSDoc，第二招是typings/d.ts。下面我们来分别阐述其妙处。

JSDoc

JSDoc 跟 Javadoc 或者 phpDocumentor 很类似，它是一个文档规范工具，我们通过在代码里写上注释，记录好各个类、函数、对象的作用，然后就可以生成相应的 API 文档了。同时，我们还能够在注释里标记对象的 JavaScript 类型，这样我们在阅读和使用这段代码时就很方便了。

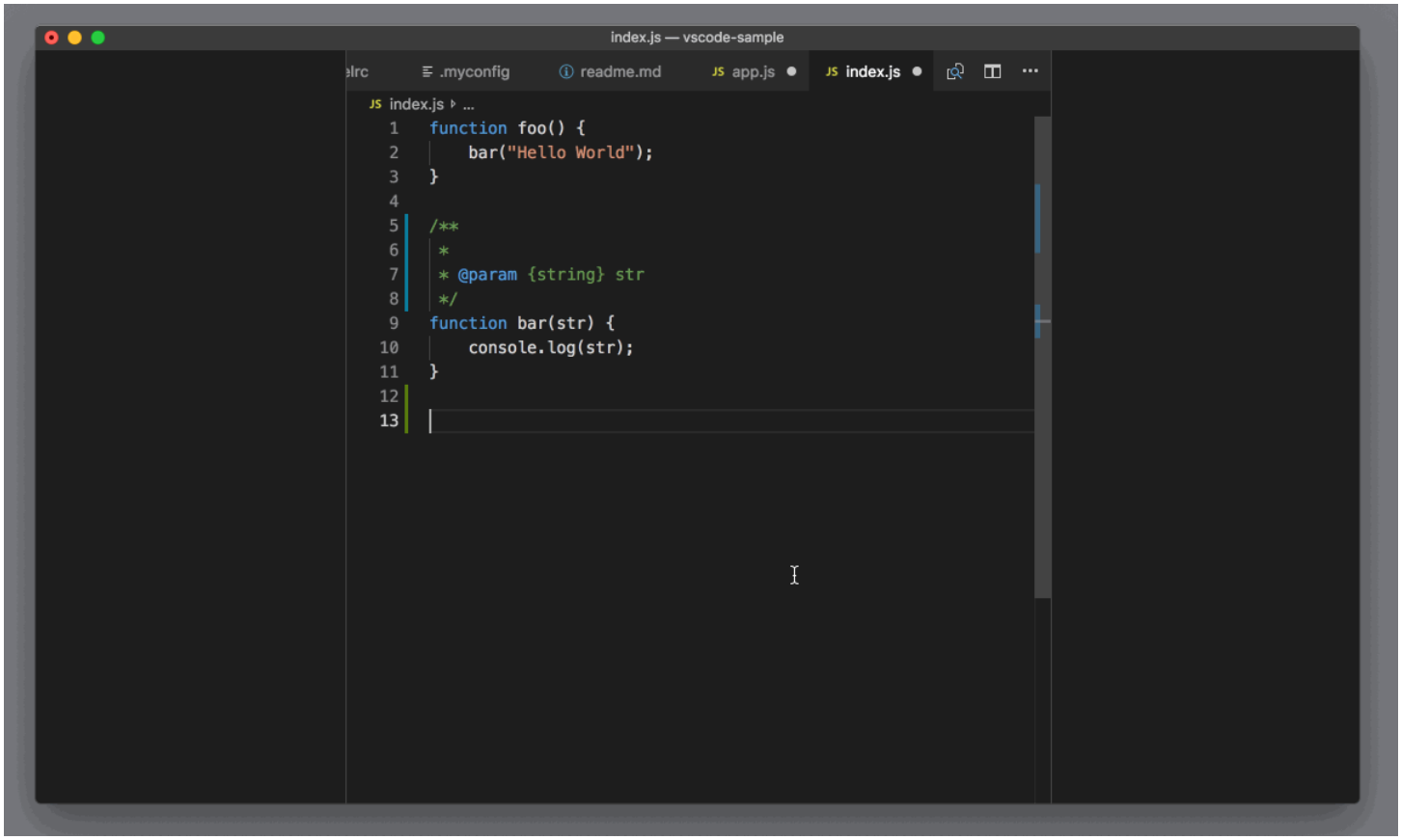
下面，让我们来给 index.js 里的 bar 函数添加上 JSDoc，新的代码如下：

```
function foo() {
    bar("Hello World");
```

```
}

/**
 * bar
 * @param {$tring} $tr
 */
function bar($tr) {
    console.log($tr);
}
```

上面的代码注释的意思是，bar 这个函数，它需要传入一个参数 str，同时这个参数的类型是 string。

此时当我们在调用 bar 函数时，参数建议就会告诉我们需要传入一个 string 类型的参数。



通过 JSDoc 提供函数参数建议

VS Code 里的 JavaScript 语言服务，会读取 JavaScript 文件里的 JSDoc 注释，然后根据注释里提供的类型信息，来对类型进行检查和建议。所以，如果你希望给你的 JavaScript 项目增加类型，并且有比较好的开发体验，JSDoc 就是一个不错的选择。关于更多 JSDoc 的知识，请参考文档。

typings / d.ts

看到这里，你的下一个问题可能是：如果我使用的函数是来自某个 npm 模块，也就是说这是别人写的

代码，VS Code 还能知道这个函数的参数类型吗？

这里就必须要再提一次 TypeScript 的类型系统了。TypeScript 的一大特点就是**静态类型**，一般一个 TypeScript 项目，发布的时候会编译成 JavaScript，同时也会发布一个**d.ts**文件，这个文件记录了发布的这个 JavaScript 文件里的对象类型。

接着 VS Code 则会通过读取这个 d.ts 文件，来为这个模块里的函数和对象提供类型建议。VS Code 是使用下面几种方式去寻找 d.ts 文件：

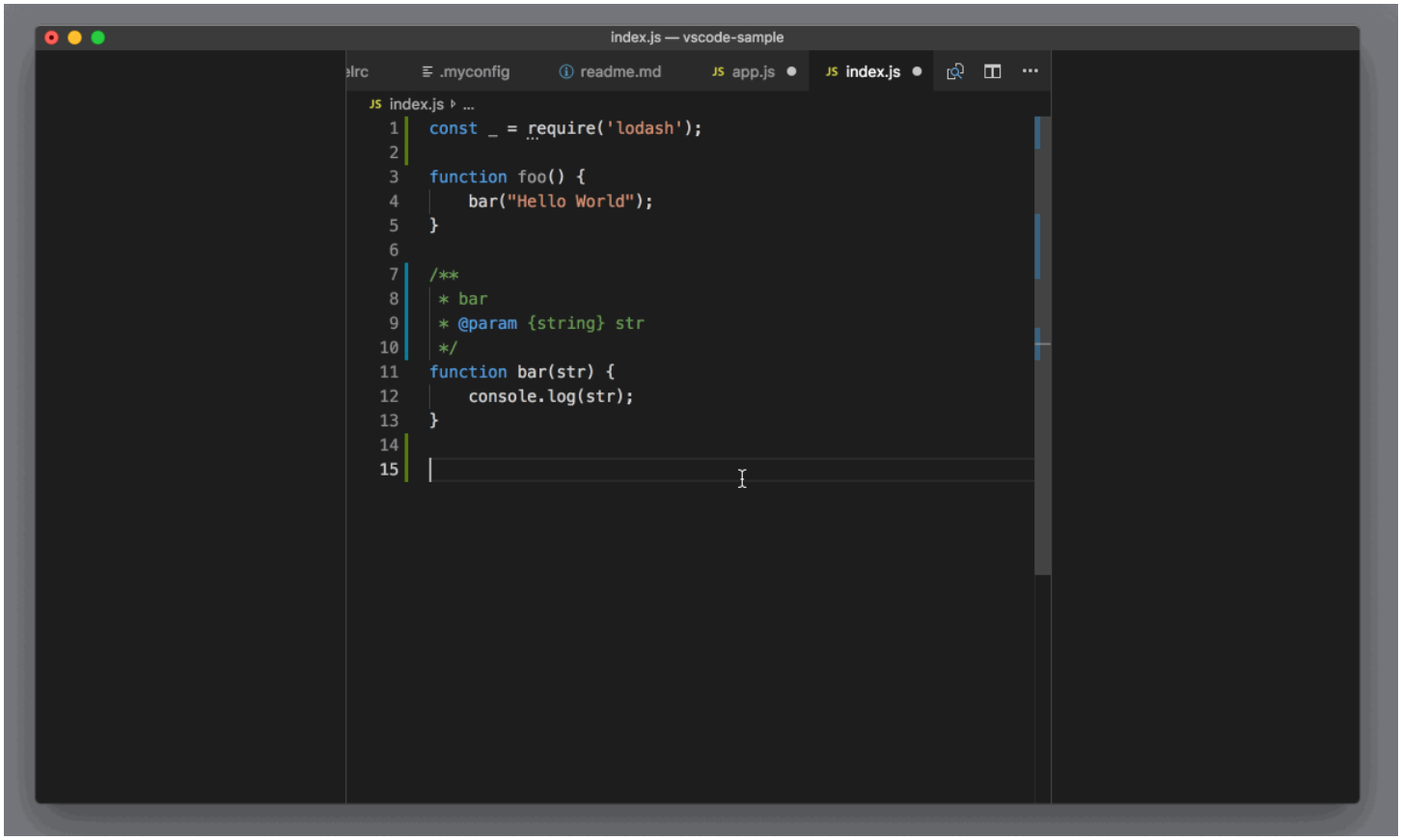
- 首先，就是看看这个 npm 包本身有没有自带 d.ts 文件，如果有的话就直接使用。使用 TypeScript 书写的项目一般都会有 d.ts 文件，而很多知名的 JavaScript 框架，虽然并不是使用 TyepScript 来维护的，也提供了 d.ts 文件。
- 其次，VS Code 还会看当前项目文件夹下是否有 d.ts 文件。如果你使用的某个 npm 包自己没有 d.ts 文件的话，你可以自行书写。关于如何书写 d.ts 文件，可以参考TypeScript 的文档。
- 最后，还有很多 JavaScript 项目，它们自己没有 d.ts 文件，但是社区为它们书写了d.ts文件，并且发布到 npm @types 下供大家使用。而VS Code 会自动到 npm @types 里进行搜索，看看是不是有合适的类型 d.ts 可以使用。

在我的日常开发工作中，上面的第三种方式是最常发生的，而且值得称道的是，第三种方式并不需要用户做任何的事情。VS Code 会自动搜索，找到合适的 d.ts 文件，然后下载下来，接着就能够提供智能提示了。这个功能又被叫做**自动类型采集**（Auto Type Acquisition）。

回到我们的示例代码中，假如说我们想在 index.js 里使用 lodash 这个 npm 包，我们会在代码的最上方，添加如下的模块引用：

```
const _ = require('lodash');
```

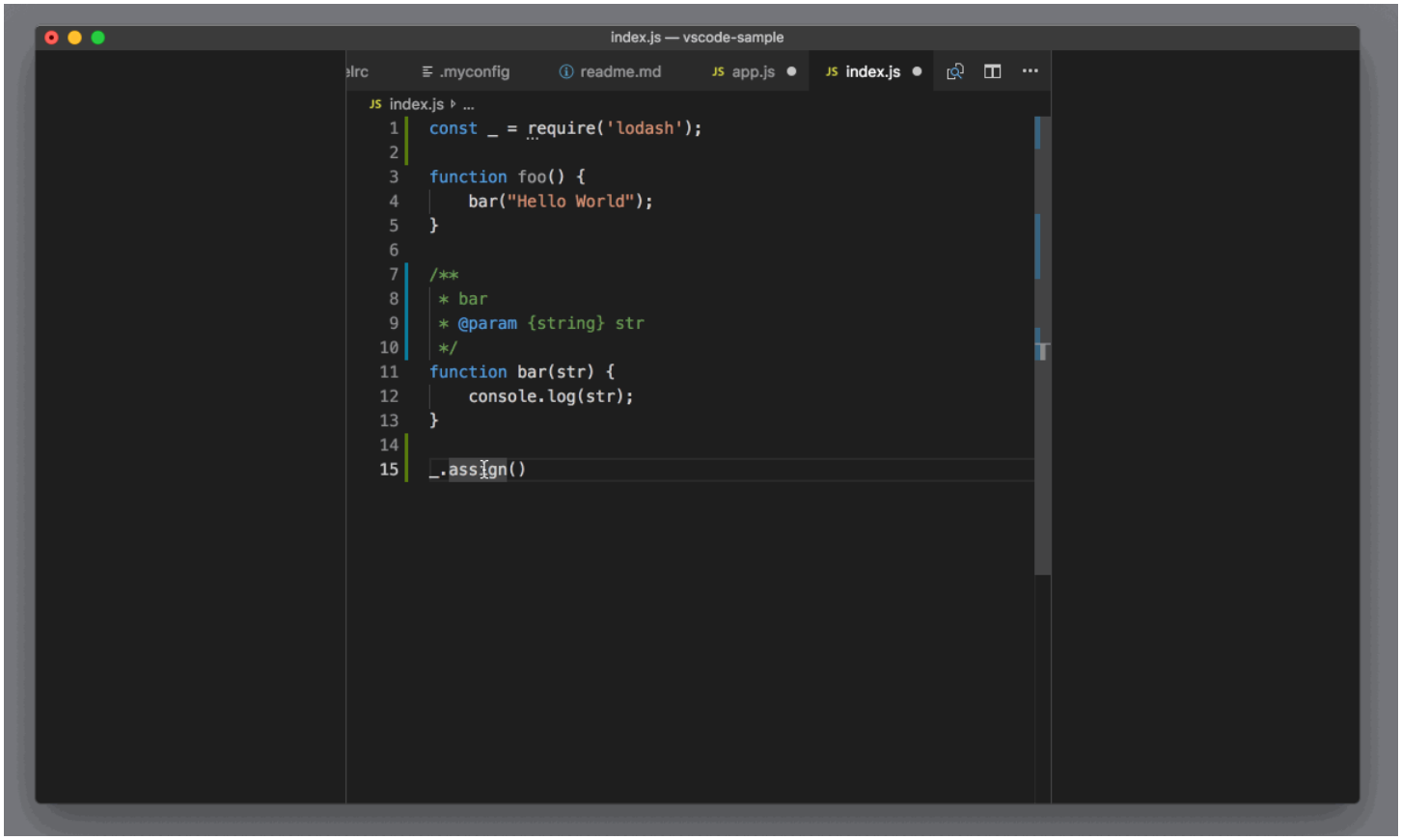
添加完这段引用后，当我们输入 _. 时，VS Code 就会立刻给我们建议 lodash 里提供的各种函数了。



通过 d.ts 文件提供建议

在上面的动图中，你可以看到，除了建议列表，在输入参数时，我们还可以看到参数类型建议和相关的文档信息。

而如果我们在 lodash 的这个函数上运行“跳转定义”命令，会发现 VS Code 跳转到了 lodash 对应的 d.ts 文件（object.d.ts）中。



跳转到 d.ts 文件中

通过上面的两个例子，相信你已经明白了，**虽然我们没有书写 TypeScript，但是 VS Code 会通过查找模块相关的 d.ts 文件**，来努力给我们提供类型相关的建议。而我们也可以通过书写 JSDoc 格式的注释，主动地给 VS Code 提供类型信息。

ts-check

很多同学很喜欢TypeScript 的类型系统和代码检查，但是另一方面，又觉得要想把整个项目迁移到 TypeScript 工作量太大了。TypeScript 团队也考虑到了这一点，为了能够让 JavaScript 社区渐进地使用 TypeScript 的工具链，他们为 JavaScript 提供了一个新的功能，叫做**ts-check**，也就是在 JavaScript 代码中，**手动地**申明开启更强的代码审核。

我们可以在上面的示例代码的第一行，添加下面这段注释：

```
// @ts-check
```

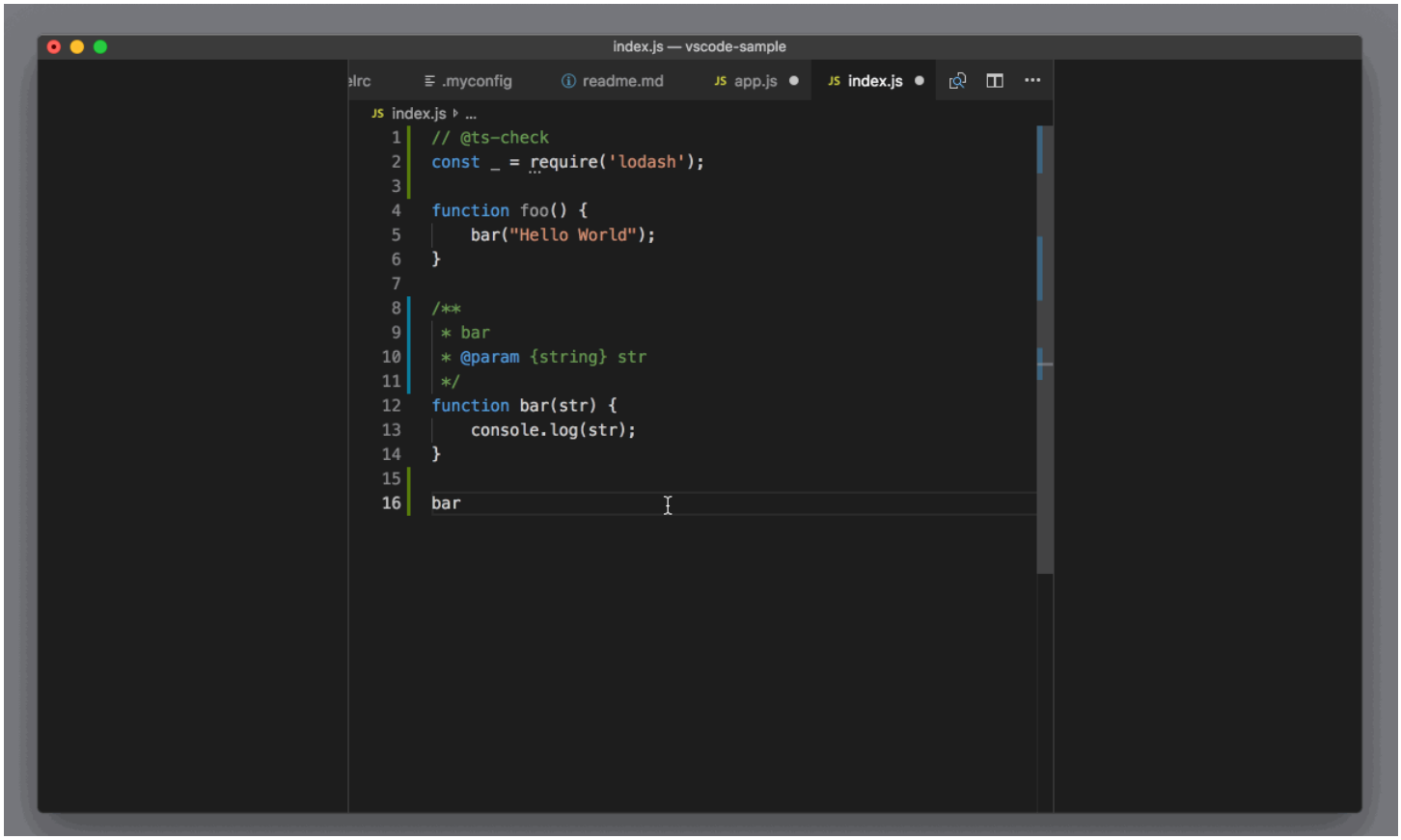
此时整段代码显示为如下：

```
// @ts-check


const _ = require('lodash');
```

```
function foo() {
    bar("Hello World");
}

/**
 * bar
 * @param {string} str
 */
function bar(str) {
    console.log(str);
}
```

在这段代码中，我们使用了 JSDoc 来申明 bar 这个函数，必须传入一个参数，而且是 string。如果我们在使用 bar 函数时没有遵守的话，TypeScript 就会提供错误信息。

比如，我们输入 bar() ，就能看到错误信息：[ts] 应有 1 个参数，但获得 0 个。而如果我们输入 bar(1)，则能看到： [ts] 类型"1"的参数不能赋给类型"string"的参数。



ts-check 检查错误信息

有了 ts-check后，我们就可以一个文件一个文件地给代码添加类型信息，然后为这个文件单独开启错误检查了。这样既不用担心全部迁移 TypeScript 带来的工作量和各种阻力，也能享受到类型系统所带来

的好处了。

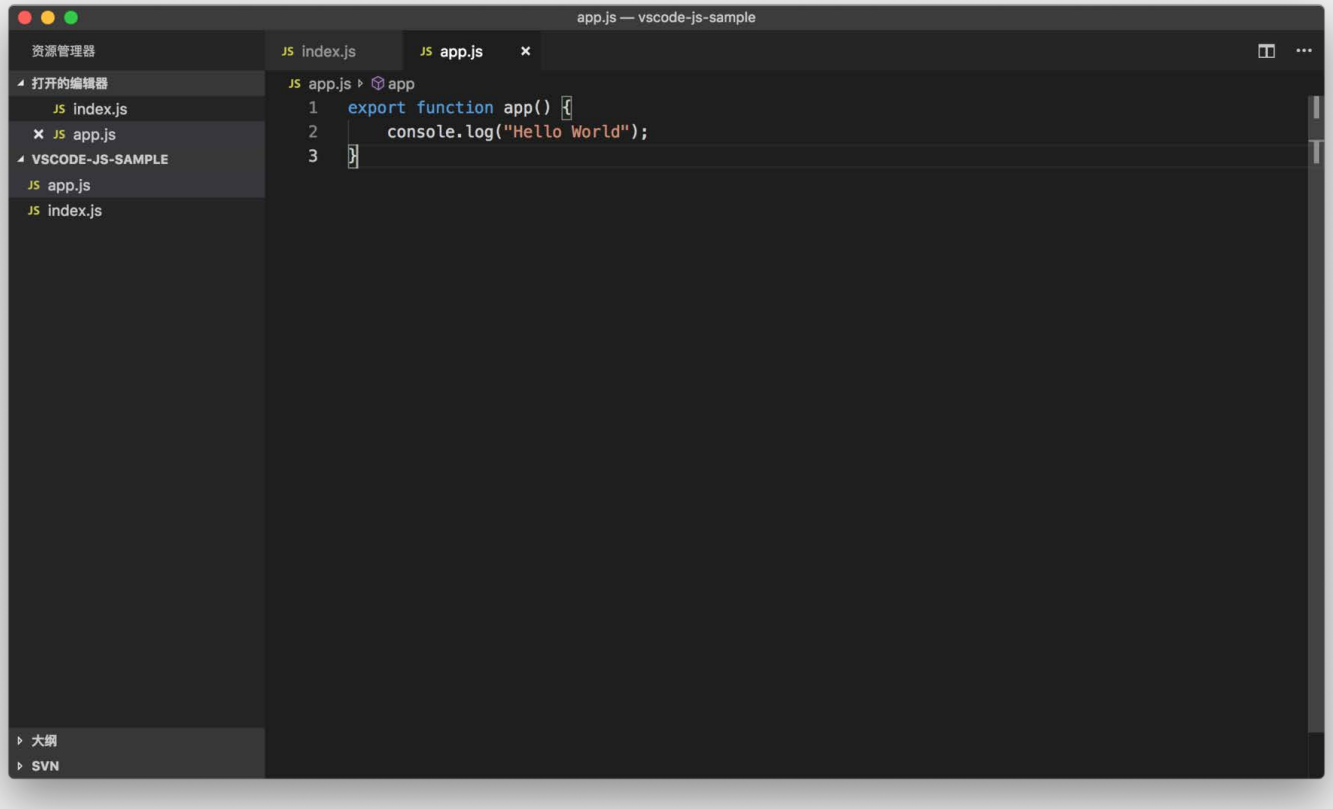
模块引用

上面我们介绍的代码示例，是一个单文件，我们能够在其中引用第三方 npm 模块。而如果我们的文件夹中有多个 JavaScript 代码，并且彼此之间互相引用的话，VS Code 还能够理解吗？我们不妨试试。

相对地址引用

我们先在项目下新创建一个 app.js 文件，内容如下：

```
export function app() {
    console.log("Hello World");
}
```

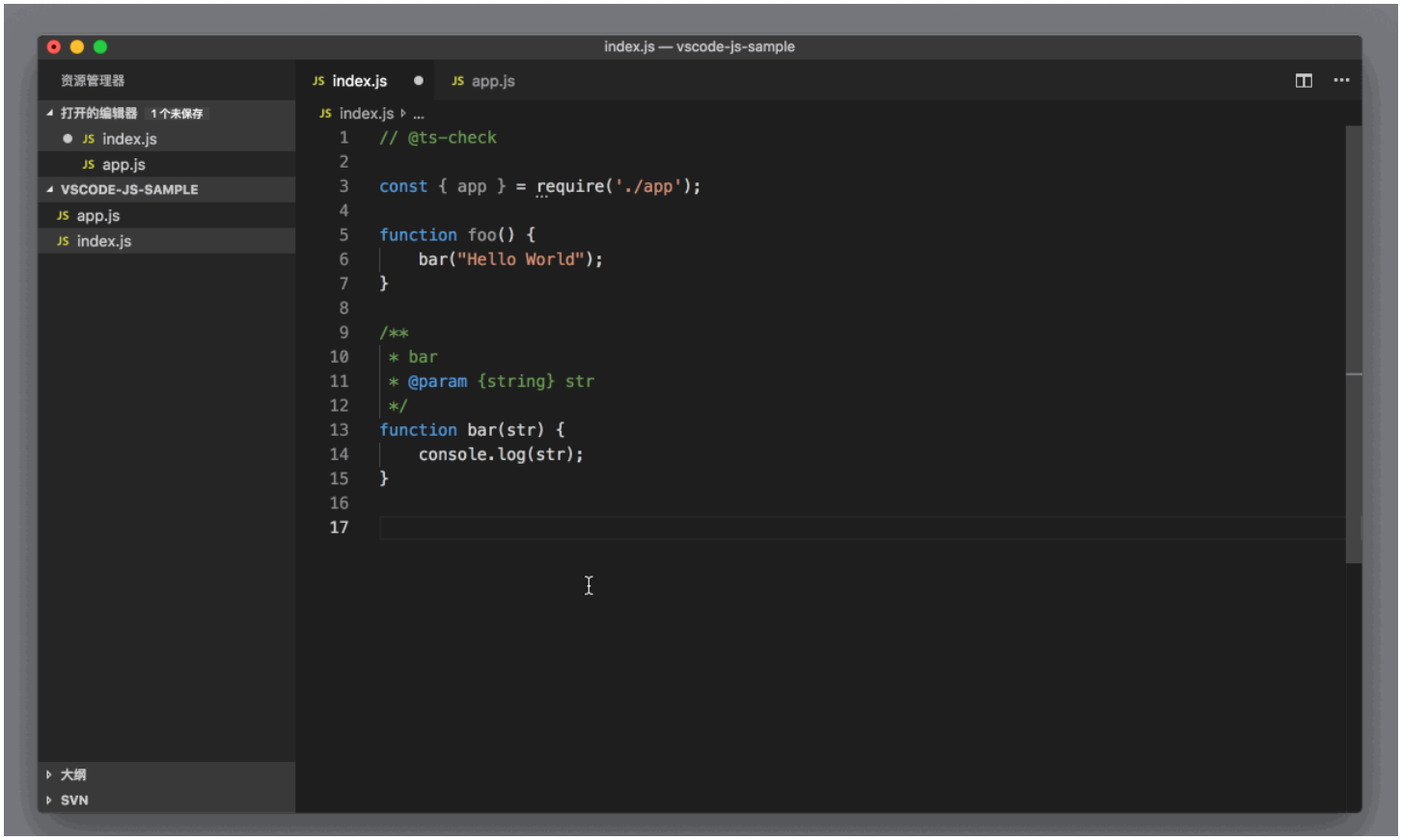


app.js

而在 index.js 中，我们替换掉 lodash 的引用，而引用 app 这个模块。

```
const { app } = require('./app');
```

引用后，当我们在 index.js 书写代码时，就能够得到 app 这个函数的提示了。



app 模块的代码提示

对于 app.js 这个模块，VS Code 的处理思路跟处理 lodash 那个 npm 包的引用是一样的。VS Code 先是通过 `./app` 这个相对路径找到模块，然后看这个模块里 export 哪些函数和对象，接着在 index.js 提供提示。

但是，JavaScript 程序员都知道，JavaScript 世界里的模块系统，从来都没有一个广泛接受的标准，即使有标准，社区也会使用一些前沿的、还在测试探讨阶段的新语法，常见的模块有：AMD、CommonJS、ECMAScript 6 Module，等等。而社区广泛使用的打包工具 Webpack，更是允许你各种自定义模块的引用方式。这时VS Code 的 JavaScript 语言服务，要想同时支持所有的模块系统，可就头疼了。

举个最简单的例子——模块绝对路径引用。在上面的例子里，我们引用 app 这个模块时，使用的是相对地址：

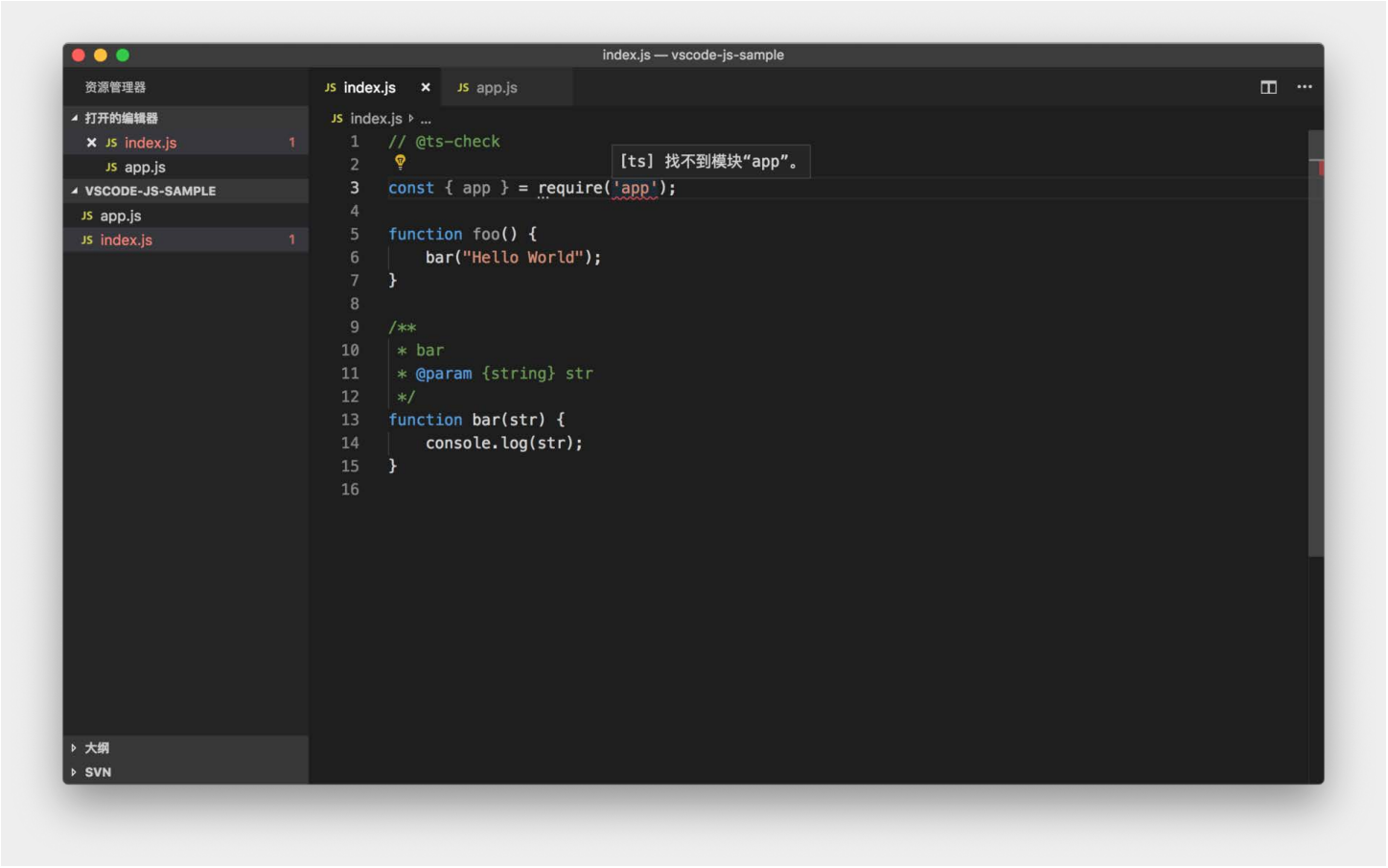
```
const { app } = require('./app');
```

这个很好处理，我们到同一个目录下去找这个 app 模块就行了。而如果使用了绝对路径，比如：

```
const { app } = require('app');
```

这个就比较麻烦了。JavaScript 的语言就不知道，究竟是把这个当作 npm 包，然后到 node_modules 文件夹下去寻找呢，还是从当前项目的根目录下去寻找这个模块呢？如果项目中使用了 babel 或者 webpack 的话，又该怎么去理解这些工具对模块的处理呢？

此时，在 VS Code 的编辑器里，我们已经能看到错误提示了。



模块引用错误提示

jsconfig

为了解决这个问题，我们需要引入一个特殊的配置文件 jsconfig.json。这个配置文件用于提示 JavaScript 语言服务，当前项目的 JavaScript 代码，哪些文件是属于这个项目的？JS 语法是哪个版本？而又该去哪里寻找模块？

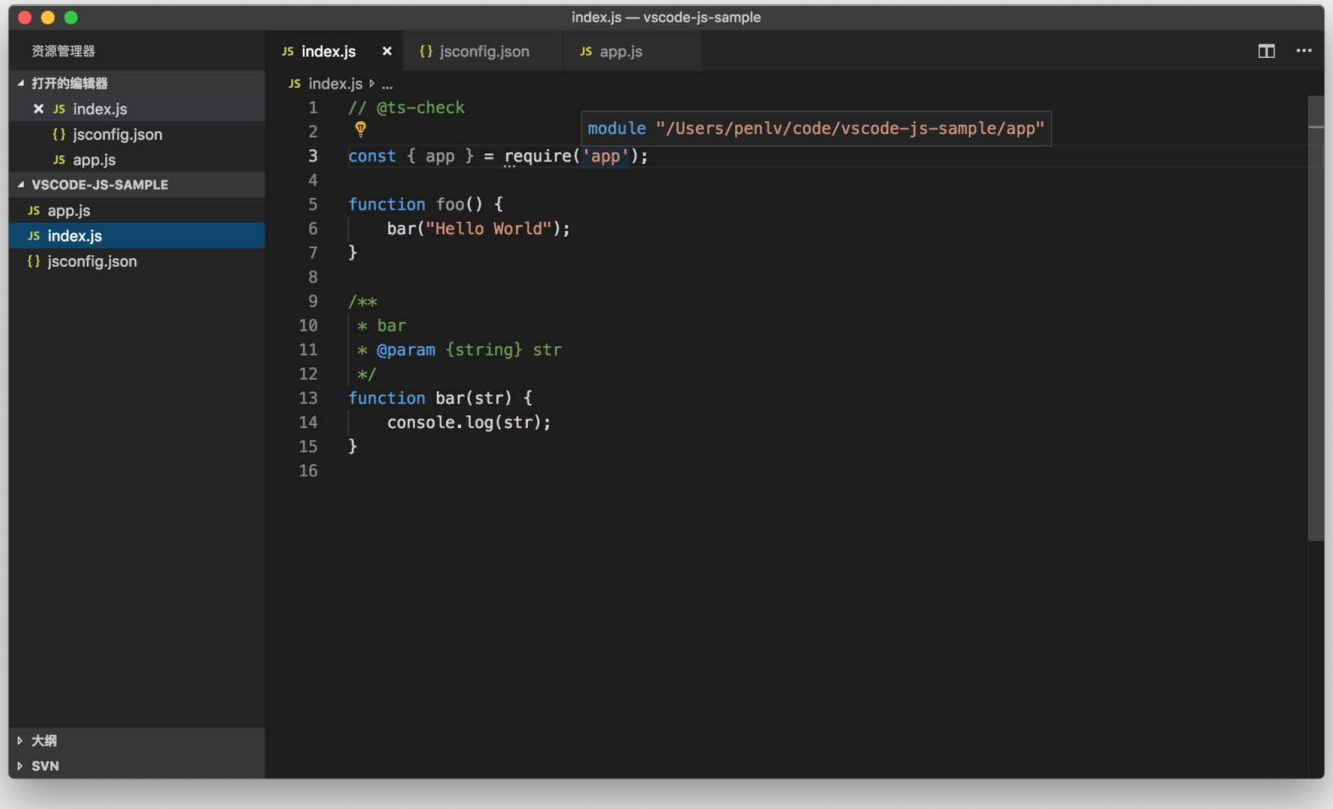
下面，让我们在文件夹下创建如下内容的 jsconfig.json 文件：

```
{
    "compilerOptions": {
        "module": "commonjs",
        "target": "es2016",
        "baseUrl": "."
    },
```

```
    "exclude": [
        "node_modules",
        "**/node_modules/*"
    ]
}
```

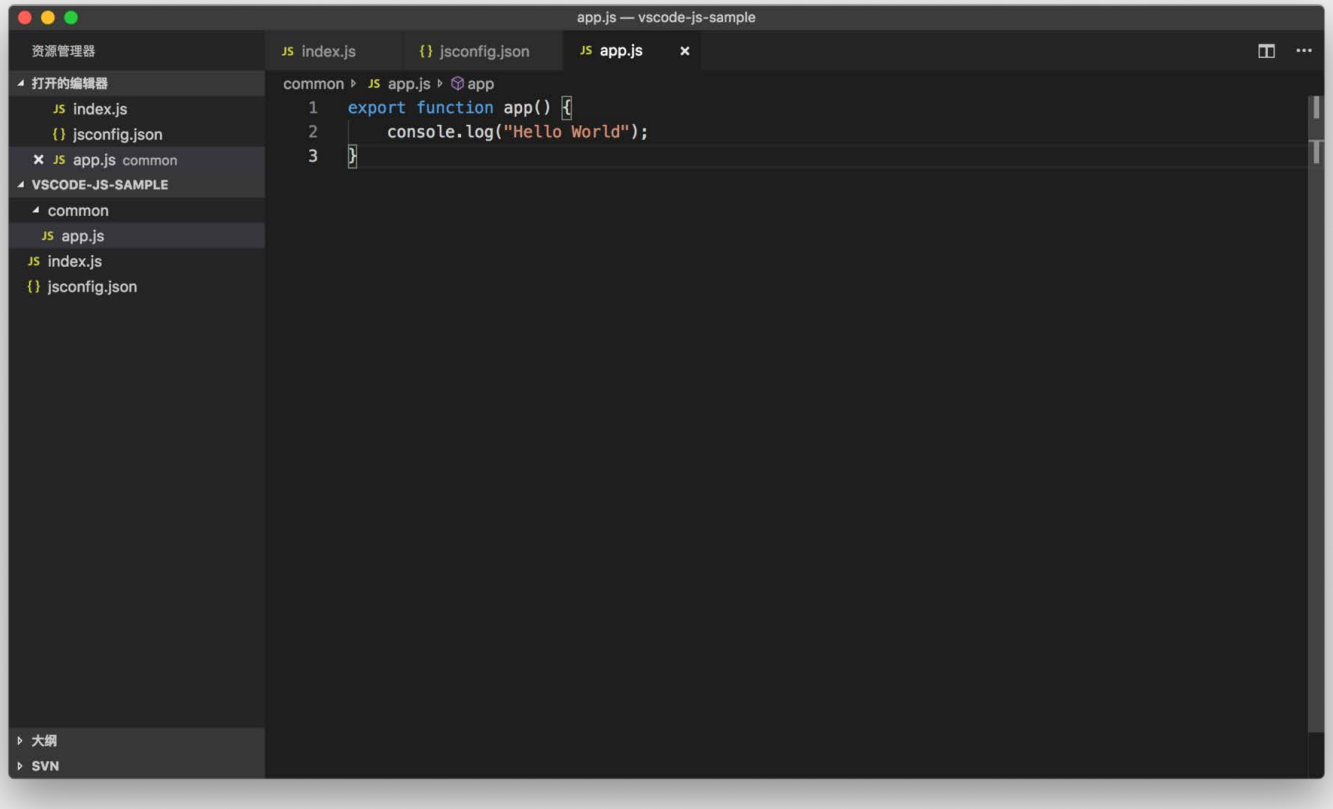
这个文件告诉 VS Code ，当前项目里的 JavaScript 语法是使用 es2016 的，引用模块时，如果模块的名称是绝对路径，那么请从根目录（也就是 .）开始寻找。

有了这个文件之后，VS Code 的 JavaScript 语言服务立刻就找到 app 这个模块了。



VS Code 通过 jsconPg 找到模块

jsconPg 甚至还允许指定多个文件夹依次进行模块的查找。下面我们创建一个文件夹 common，然后把 app.js 挪入到 common 中。



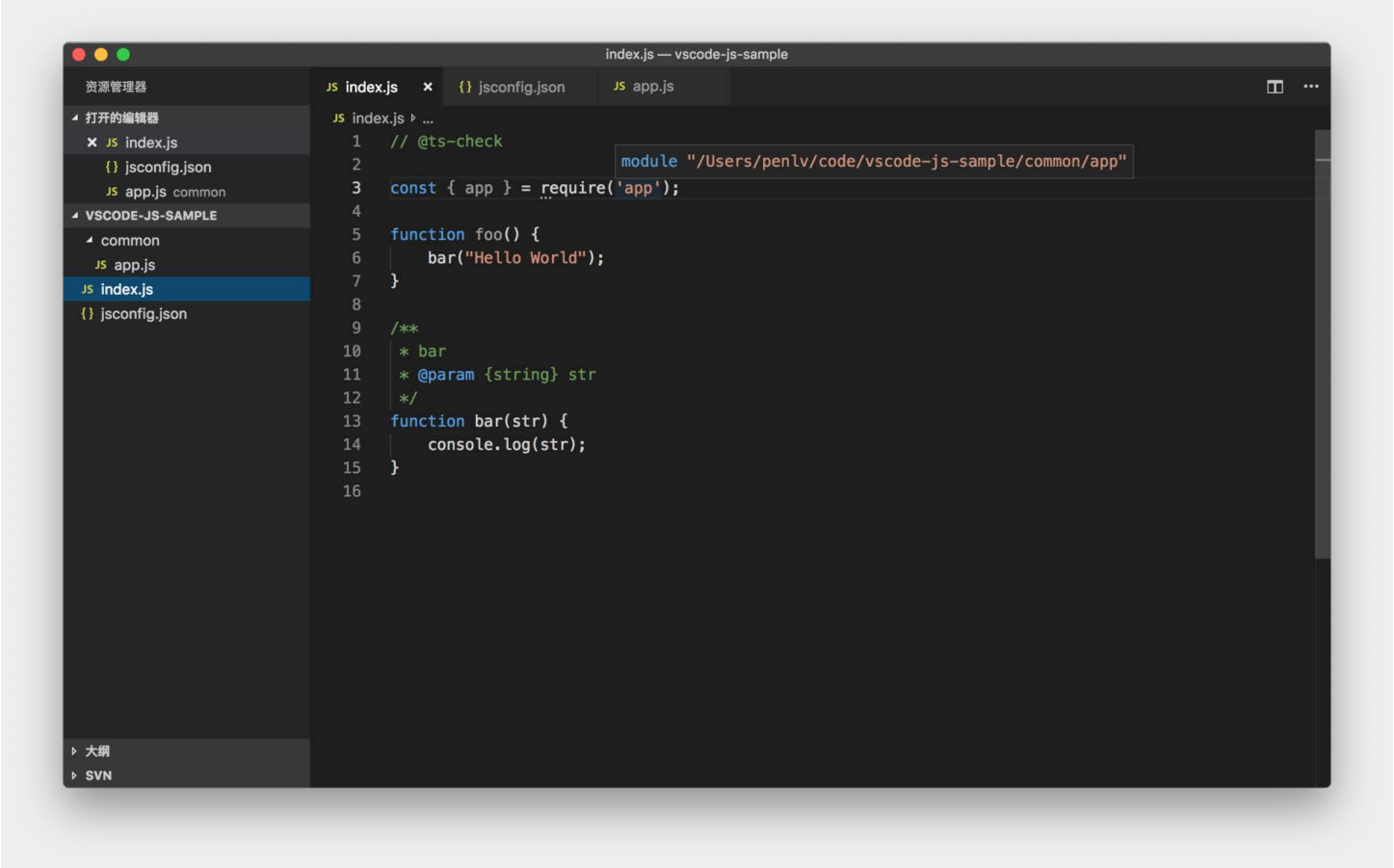
common/app.js

然后在 jsconPg 中，进行如下的修改：

```
{
    "compilerOptions": {
        "module": "commonjs",
        "target": "es2016",
        "baseUrl": ".",
        "paths": {
            "*": [
                "*",
                "common/*"
            ]
        }
    },
    "exclude": [
        "node_modules",
        "**/node_modules/*"
    ]
```

```
}
```

我们在上面的 jsconÞg 里添加一个 paths 属性，这里面就是对模块路径的映射。VS Code 会阅读这些映射，依次去寻找模块。此时当我们再打开 index.js ，将鼠标移动到 require('app') 代码上时，我们能够看到，此时 VS Code 正确地将 app 这个模块定位到了 common/app.js 这个文件。



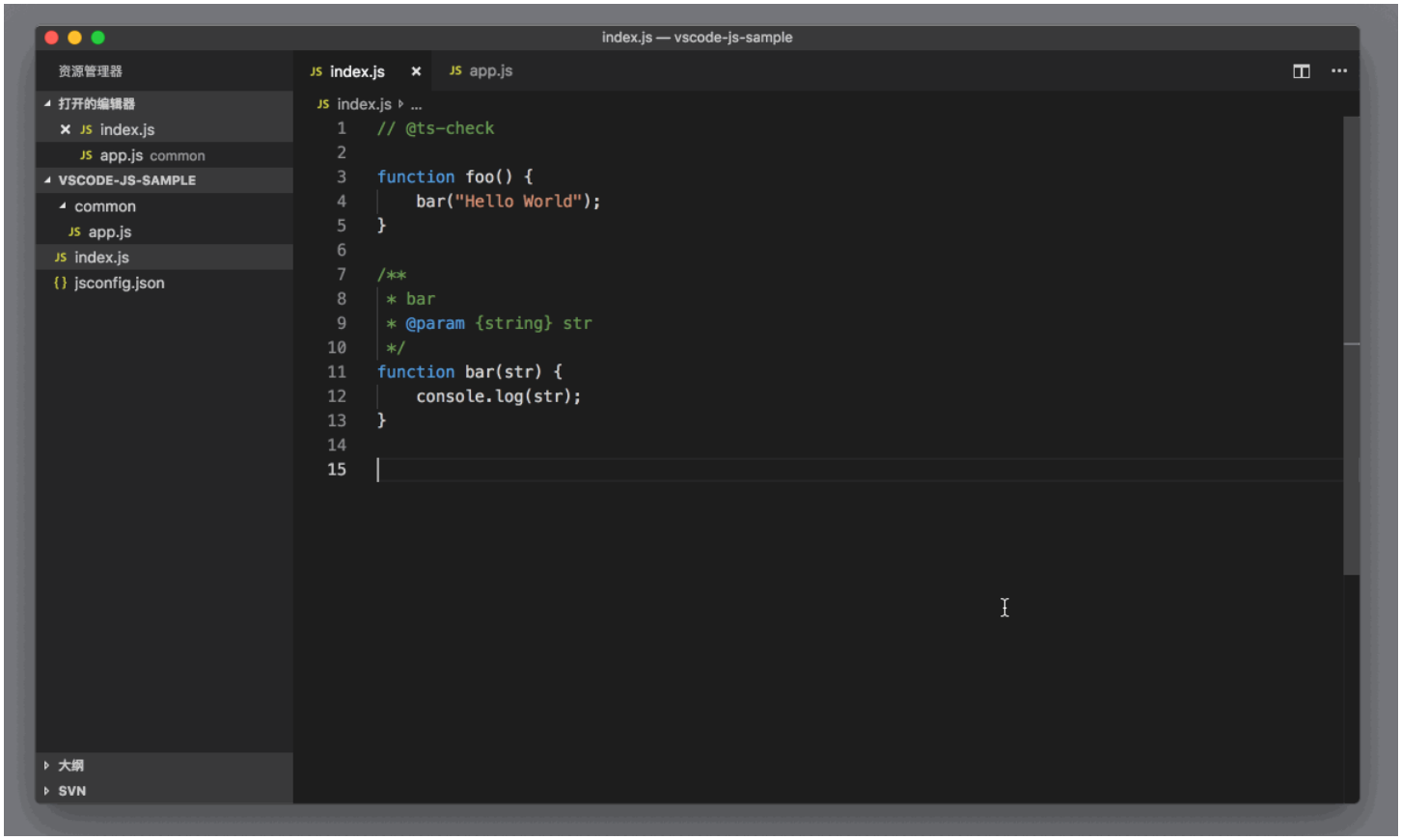
app 定位到 common/app 文件

在前面的专栏里，我们在介绍 VS Code 的智能语言服务功能时，有读者提了一个非常好的问题，那就是如果项目中使用了 Webpack 而且 Webpack 中设置了 resolve，那么 VS Code 里的代码跳转就不工作了。相信现在你应该已经明白是为什么了，如果你的项目里为 Webpack 做了特殊的模块地址映射，那么你也需要在 jsconÞg 里做同样的映射，这样 VS Code 就知道如何找到模块了。

既然说到模块，接下来就介绍两个非常有用的模块相关的功能。

自动模块引用

首先我们把 index.js 最上面的模块引用删除。然后在 index.js 的末尾，输入 app() 这段代码。



自动模块引入

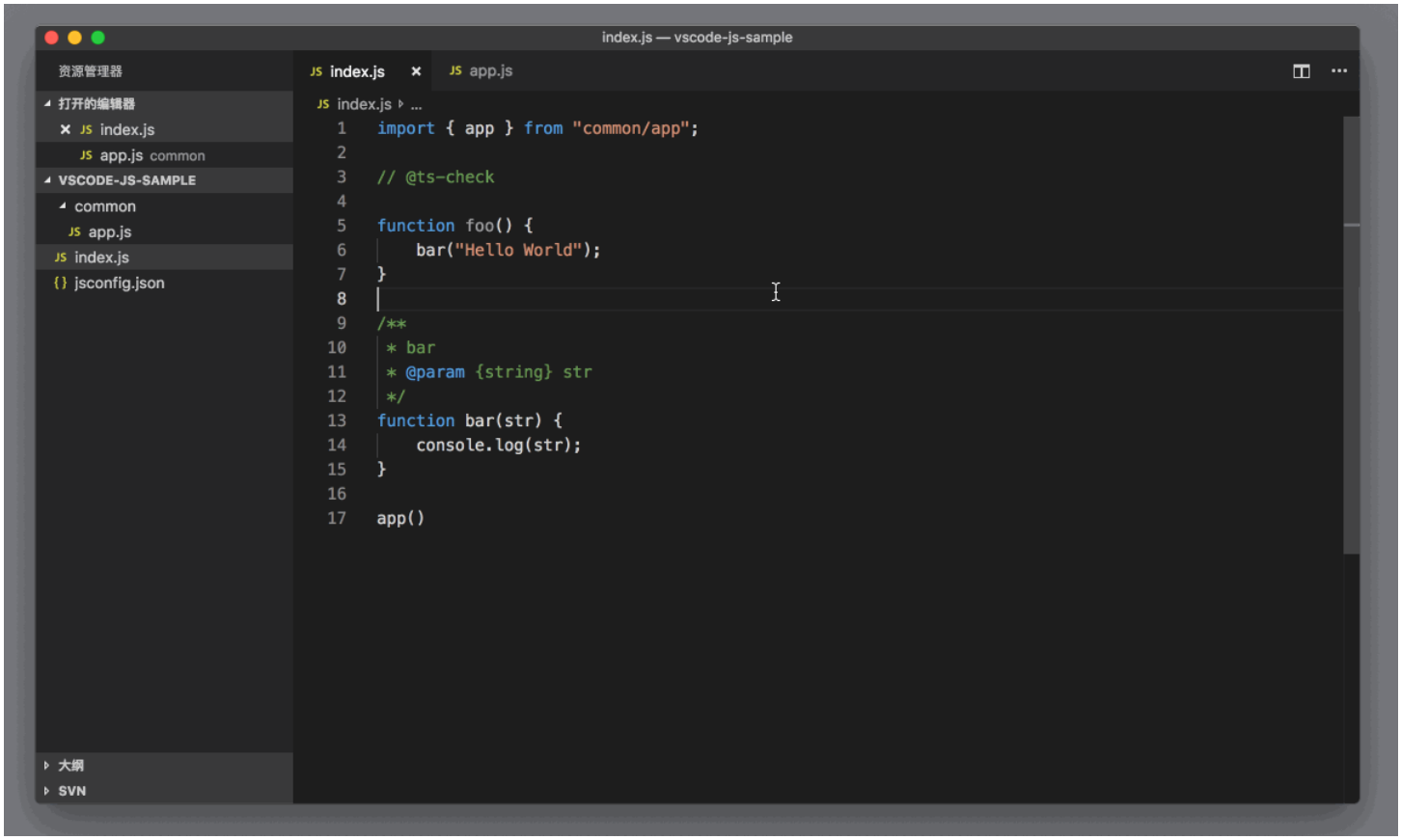
你可以看到，VS Code 自动为我们添加了模块的引用。这是因为 VS Code 已经知道当前项目里有哪些模块，每个模块提供了哪些方法，当你使用它们时，就会帮你将引用填入文件开头。

这个功能又叫**自动模块引用**（Auto Imports）。

自动模块更新

JavaScript 的模块地址很多都是跟文件地址相关的，阅读性很好，但是当我们移动文件地址的时候，可就麻烦了。因为这意味着我们需要把所有跟这个文件相关的模块引用全部修改一遍。不过好在VS Code提供了模块地址自动更新的功能。

下面我们把 app.js 从 common 这个文件夹下，移动到根目录下。



移动“ app.js”

从上面的动图中，你能够看到，VS Code 会问你是否要自动修改模块的引用，在选择“是”之后，VS Code 就会把 app 这个模块的引用地址，从 “common/app” 改成了 “app”。是不是非常实用呢?

代码审查 **tsconfig/checkJs**

上面我们提到了，可以通过在 JavaScript 文件里写入 @ts-check 注释，让 VS Code 对单个文件进行代码审查。不过如果项目不算大，而你打算对所有的 JavaScript 文件进行代码检查的话，那么也可以在 jsconÞg.json 文件的 compilerOptions 里添加 checkJs 属性，打开对 JavaScript 的检查。修改后的 jsconÞg.json 如下:

```
{
    "compilerOptions": {
        "module": "commonjs",
        "target": "es2016",
        "baseUrl": ".",
        "paths": {
            "*": [
                "*",
                "common/*"
            ]
```

```
        },
        "checkJs": true
    },
    "exclude": [
        "node_modules",
        "**/node_modules/*"
    ]
}
```

其实，jsconPg 还有很多其他实用的设置，你不妨利用 VS Code 的自动补全和代码提示，自己动手改一改，看看它们都是干嘛的。

Node.js

VS Code 的各种 JavaScript 功能，是通过 TypeScript 的编译器来实现的，但是它并没有局限JavaScript 代码是前端项目还是后端项目，VS Code 对它们的语言支持都是一致的。不过，这里不得不提 VS Code 的 Node.js 调试器。它是 VS Code 里的第一个代码调试器，可以说，VS Code 的代码调试 API，Node.js 是支持得最好的。从这个角度看，Node.js 在 VS Code 项目的地位，可以跟 TypeScript 比肩了。

那么，我们就来看看，VS Code 里对于 Node.js 调试，有哪几个有趣且实用的功能。

代码调试 Auto Attach

第一个就是**代码调试**（Auto Attach）了。在前面介绍 VS Code 的代码调试功能时，我举的第一个例子，就是打开一个 JavaScript 文件，以 Node.js 环境进行调试运行，然后又介绍了如何书写 launch.json 来提供相对复杂的代码调试配置。

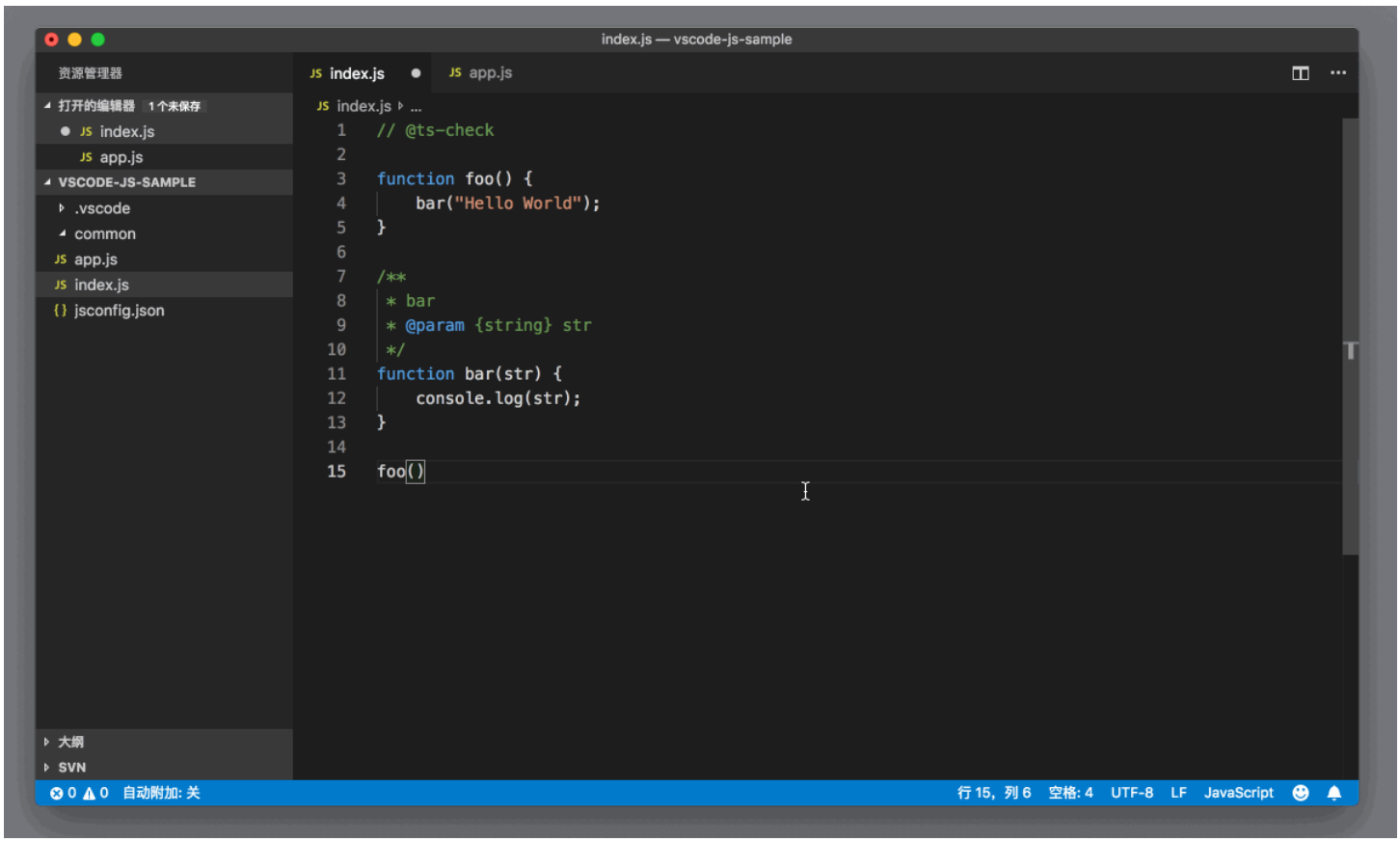
其实，Node.js 调试器则更进一步。如果我们在 VS Code 的集成终端里以命令行的形式调试代码的话，则可以无需 launch.json，直接将调试器挂载到运行的代码上。

首先，我们将 index.js 代码调整成 Node.js 支持的格式（请注意，这里我暂时不再引用 app.js 模块，而是只使用 index.js 内的函数）：

```
// @ts-check


function foo() {
    bar("Hello World");
}
```

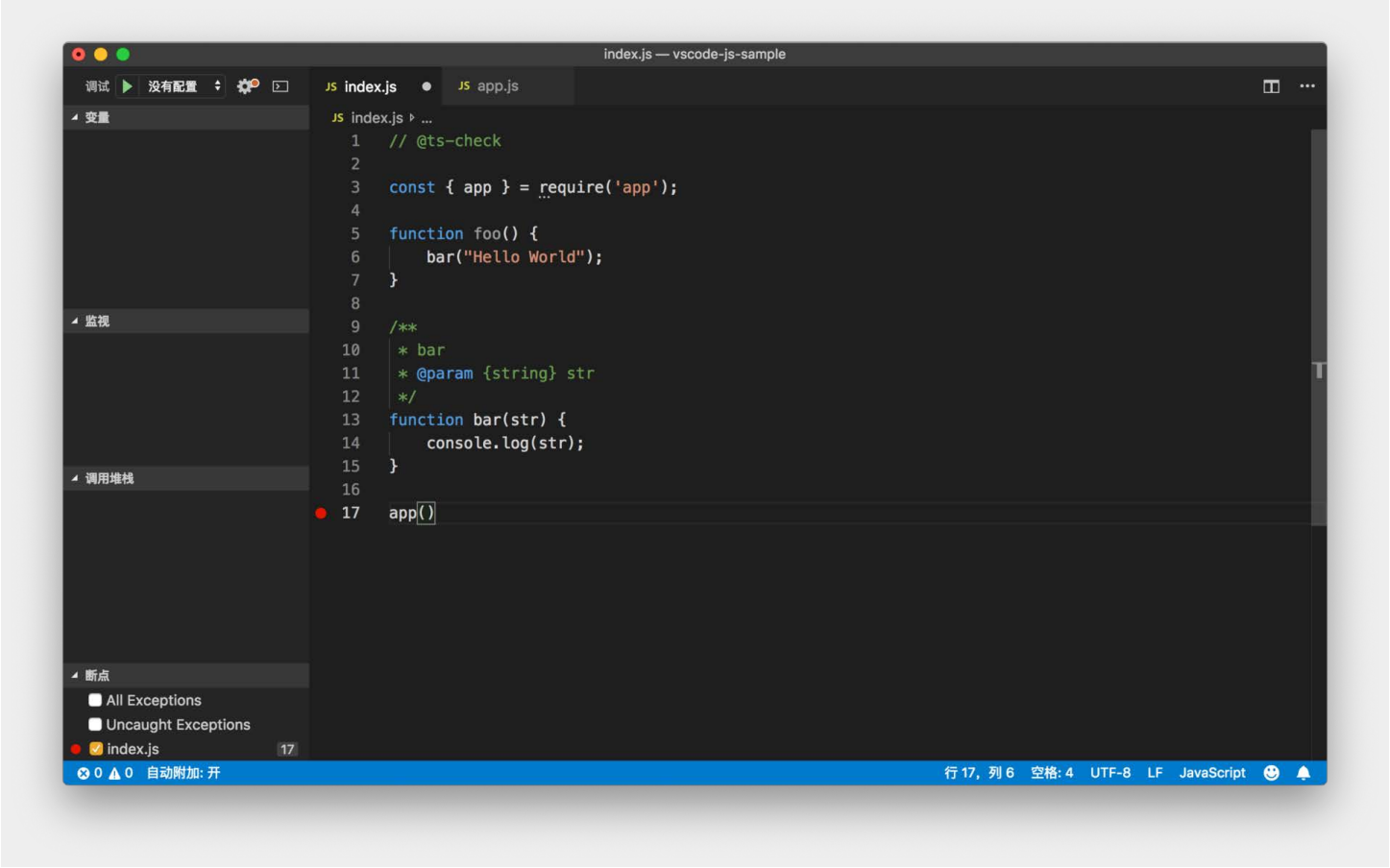
```
/**
 * bar
 * @param {string} str
 */
function bar(str) {
    console.log(str);
}

foo()
```

然后打开命令面板，执行“切换开关自动附加”（Toggle Auto Attach）命令；



Toggle Auto Attach

然后我们在 index.js 的第 15 行插入一个断点。

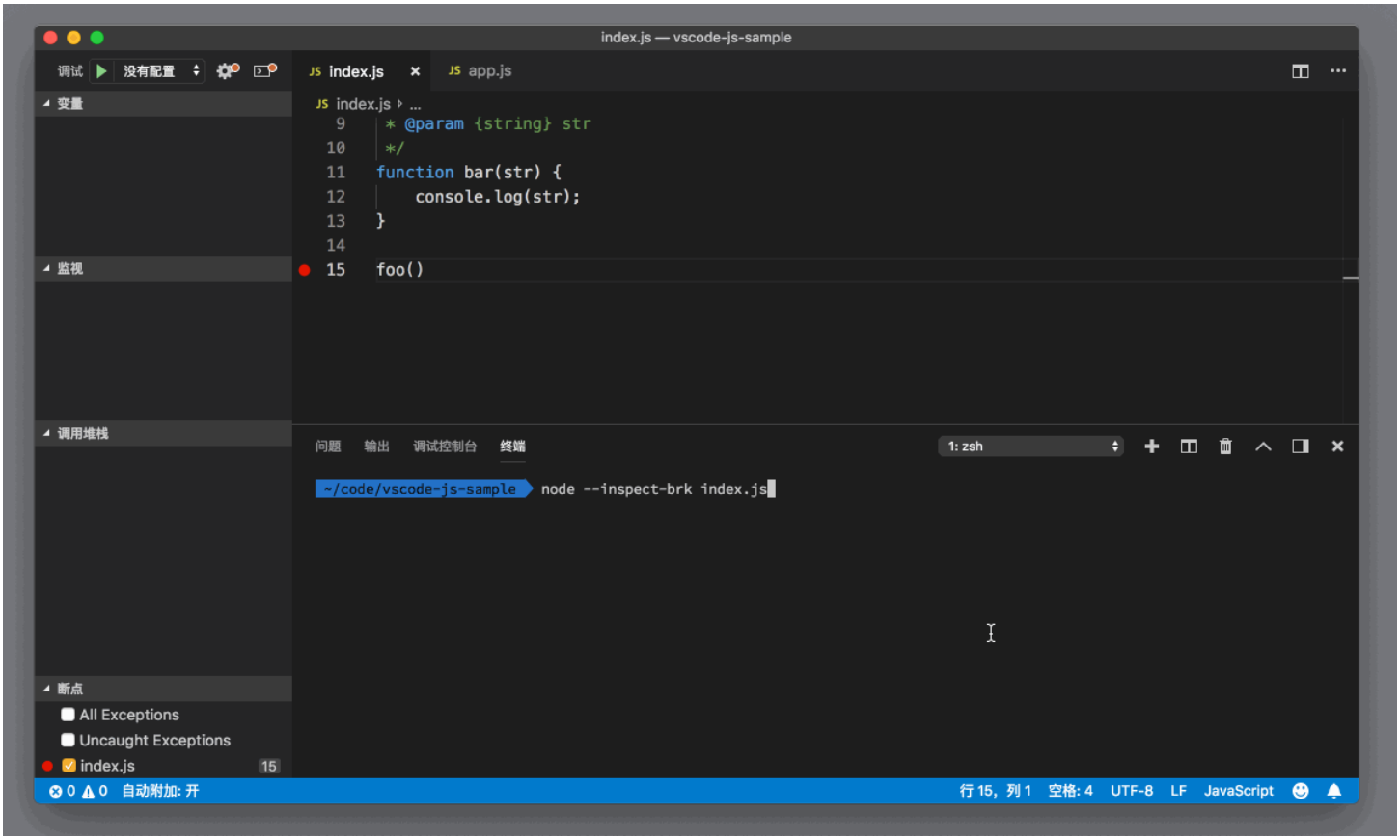


插入断点

最后，我们打开集成终端，输入 Node.js 的调试命令：

```
node --inspect-brk index.js
```

可以看到 VS Code 立刻进入了调试模式，然后在第 15 行停了下来。



进入调试模式

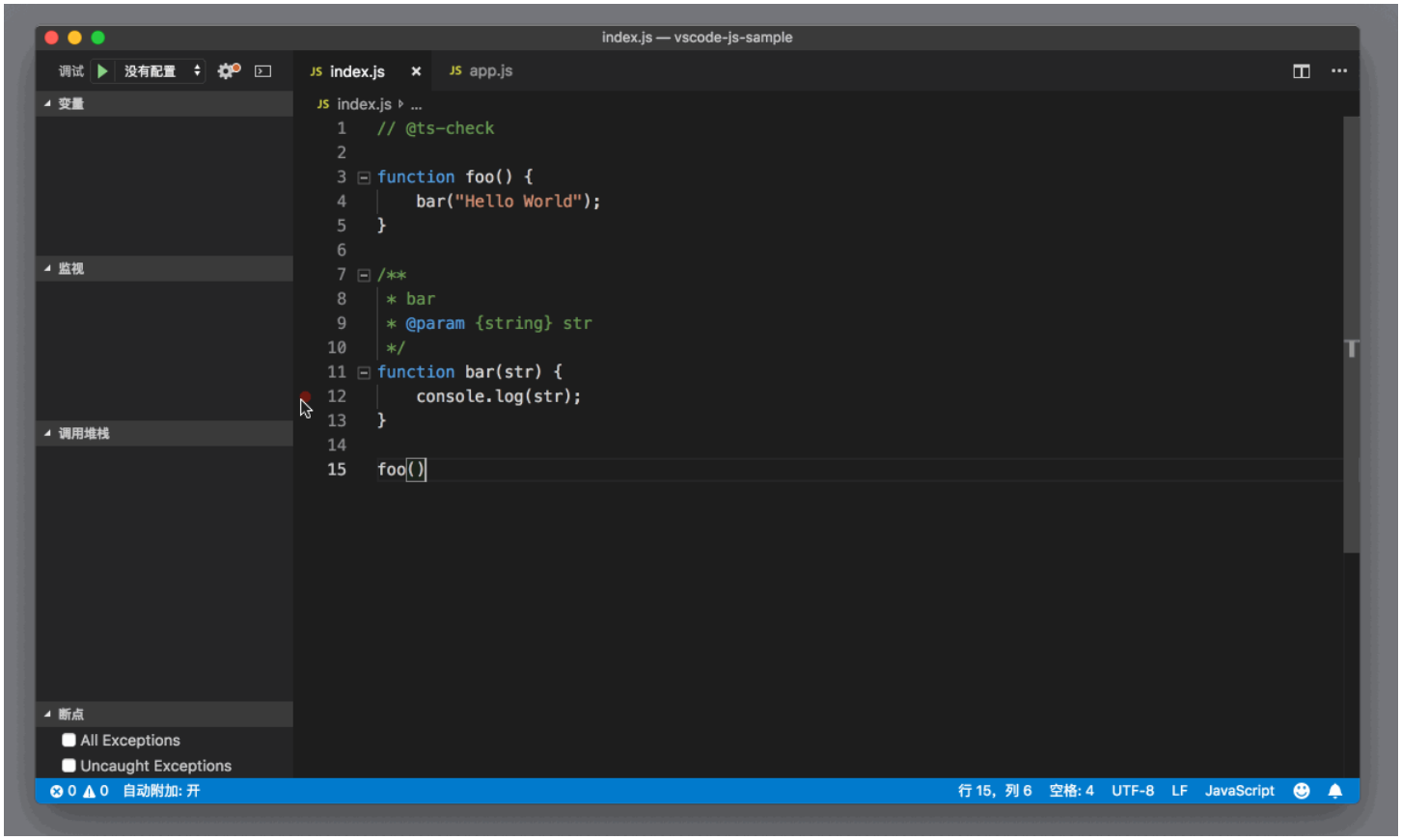
所以，如果你平时就是使用 JavaScript 直接写 Node.js，相信这个命令肯定能给你省去很多调整 launch.json 的麻烦。

记录点 **Logpoints**

下一个功能，叫做**记录点**（Logpoints），这个看名字不太好理解呢。不过相信你在开发 JavaScript 的过程中，肯定会经常在代码中输入 console.log() 来输出变量值以便调试。即使现在编辑器和浏览器的调试功能都已经非常强大了，但是很多同学依然喜欢这种简单的方式来调试代码。

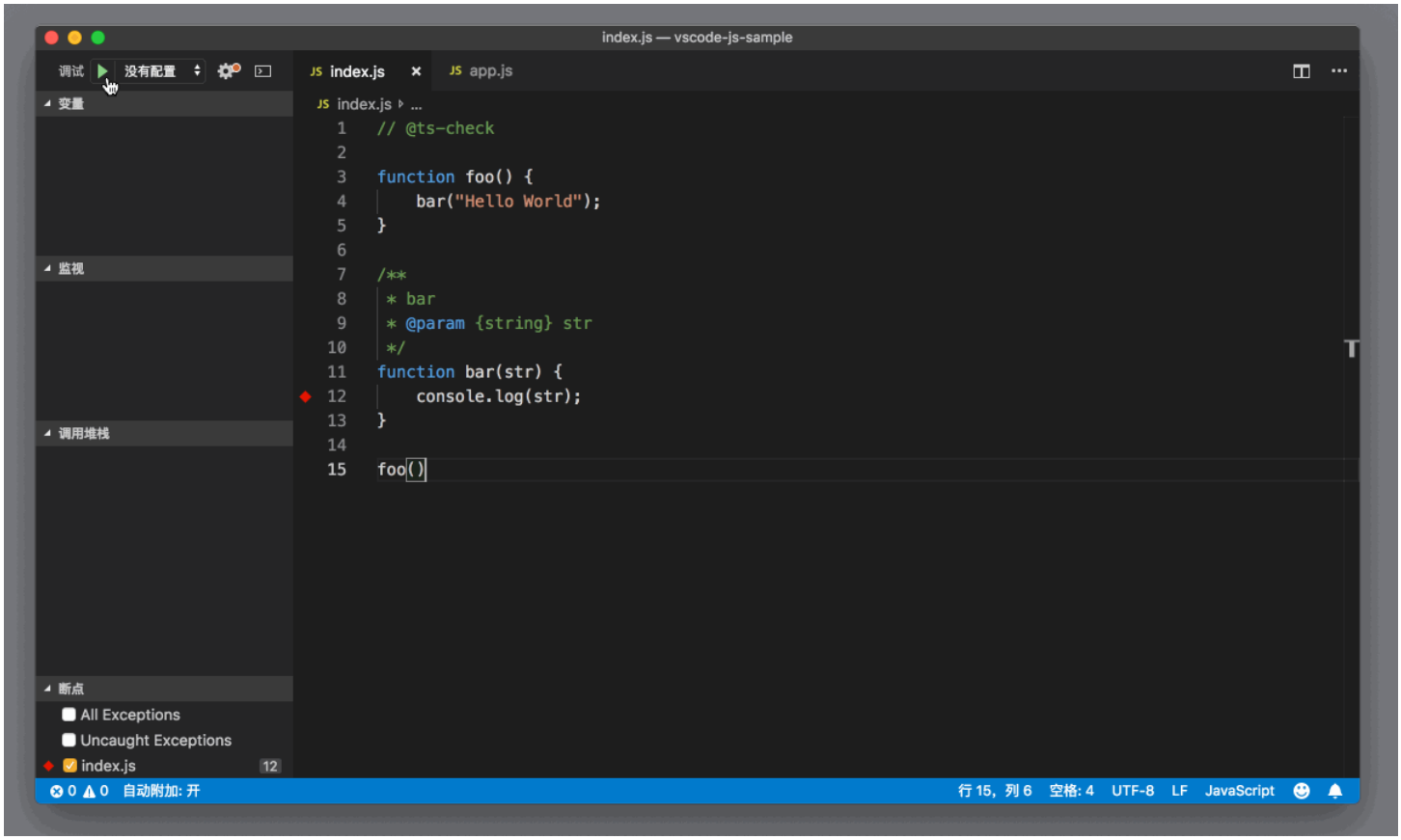
Node.js 调试的 Logpoints 功能，就是将 console.log 和断点结合起来，把 console.log 要输出的信息，通过类似于条件断点的方式执行并打印出来。具体操作是如何的呢？

首先我们在第 12 行行号前右击，从上下文菜单里选择“添加记录点”，然后从左侧的选择列表里，选择表达式。在输入框里输入我们想要输出的内容并且用花括号包裹 { str + “!” }，按下回车。



添加记录点

接着，我们 F5 调试当前这个文件，选择 Node.js 这个环境。由于我们其实并没有真正创建断点，所以代码很快执行结束。我们不妨打开调试面板看看，



我们能在调试面板里看到两个输出结果：

```
Hello World!        index.js:12
Hello World         index.js:12
```

第一条结果就来自记录点 Logpoints，而第二条则是我们代码中的 console.log(str)。看到这里，相信你就明白这个功能的用途了，你可以将原本需要写在代码里的 console.log ，放到记录点 Logpoints 中，一样可以得到输出结果，而且还不改变原有的代码。

小结

以上今天的全部内容了。关于 JavaScript 在 VS Code 里的支持，还有很多功能我们没有介绍，比如重构、Linting等，但是我们在之前的章节里都有涉及，相信你可以根据之前的内容，自己找到答案。

而我们今天重点介绍了 VS Code 的 JavaScript 的类型推倒是怎么实现的，如何为 JavaScript 项目添加配置以便 VS Code 更好理解，以及 VS Code 如何处理 JavaScript 的模块引用，等等。相信这些，你能够逐步一点点地引用到你的 JavaScript 项目中，并且帮助到你的工作。

精选留言

冰山北极熊

老师，我写的是python，我想反馈一个问题：当一个页面的代码超过1000行时，输入代码后，索引显示的非常慢，严重影响了写代码的速度？请问老师，这个有什么方法去优化吗？索引加载的速度

2018-11-06 16:08

CHENG

DEBUG CONSOLE面板的布局要是能和Chrome Dev Tools一样就好了， 每次调试的时候要在面板最下边的那个"狭长"的一行输入框里操作好不顺手...

2018-11-08 00:24

捞鱼的搬砖奇

老师我的vscode无法添加断点。在行号左侧没有间隙，点一下就切换到左边对应的活动栏了。需要在哪儿设置

2018-11-07 11:38

山石尹口

javascript下的CTRL+T没法搜索工作区中未打开文件中的符号，github上的讨论似乎也没有解决方法。

2018-11-06 22:17

Mr-L.x.D..

很喜欢最后提及的logpoint功能

2018-11-06 11:23