

32讲插件开发（一）： why、how和what



今天就到了专栏的最后一个模块：**插件开发**。这一模块大部分的知识要求你拥有一定的 JavaScript、Node.js 基础，所以如果你对此没有涉及的话，需要先学习一下 JavaScript 的知识。如果你对编程已经有经验的话，那么熟悉 JavaScript 和 Node.js 花不了多少时间。

当然，我也不可能把 VS Code 的每个插件 API 都介绍一遍，所以我会介绍几种主流的插件类型，尤其是一些你可能会用得着的插件类型。举个例子，VS Code 有版本管理相关的插件 API，你可以通过这套 API 集成某个版本管理软件，但是对于绝大部分人而言，这个 API 都是不会用得着，所以我就不会涉及这个 API 的细节。

好了，废话不多说，让我们开始吧。

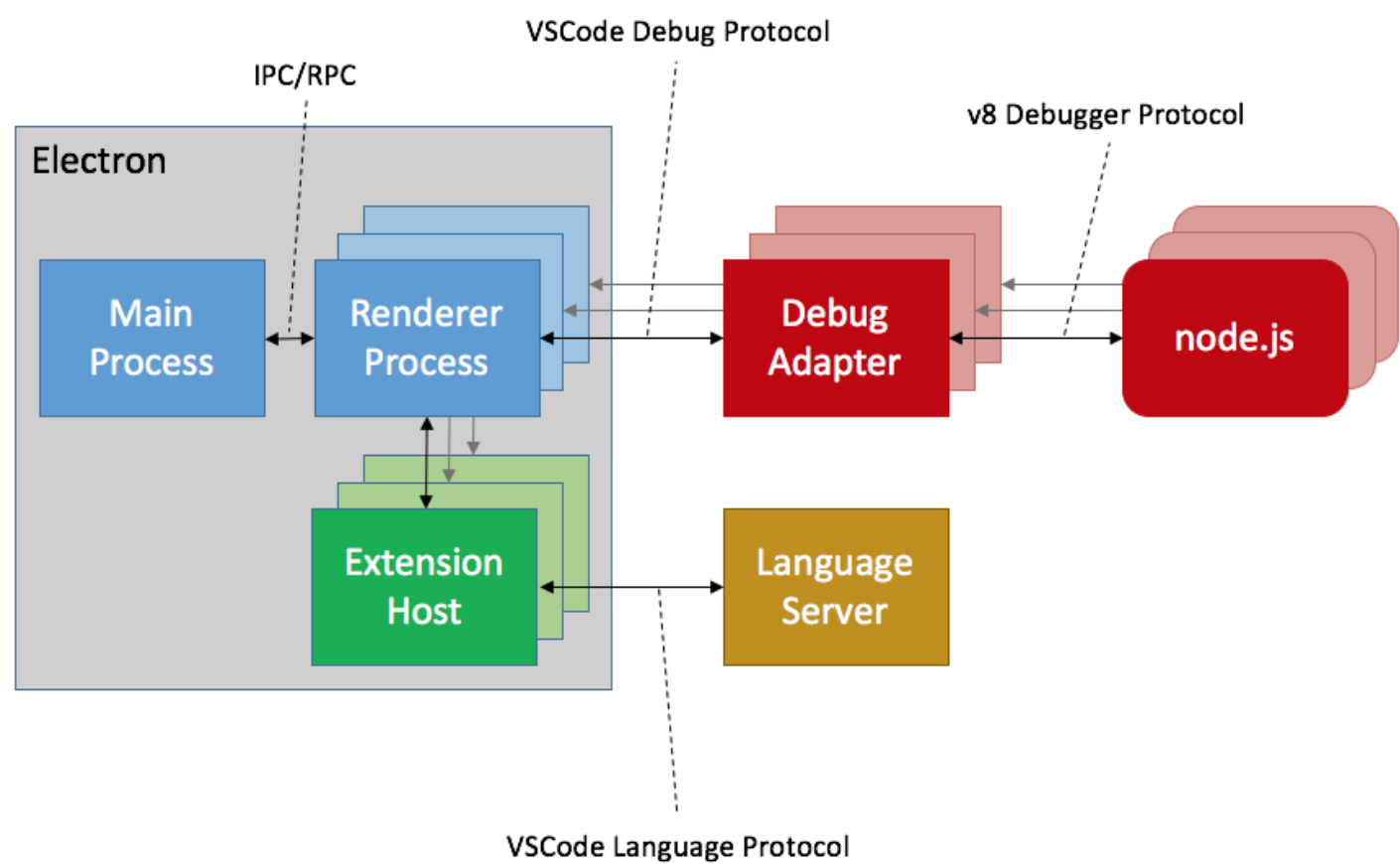
VS Code 插件架构

VS Code 是通过 Electron 实现跨平台的，而 Electron 则是基于 Chromium 和 Node.js，比如 VS Code 的界面，就是通过 Chromium 进行渲染的。同时，VS Code 是多进程架构，当 VS Code 第一次被启动时会创建一个**主进程**（main process），然后每个窗口，都会创建一个**渲染进程**（Renderer Process）。与此同时，VS Code 会为每个窗口创建一个进程专门来执行插件，也就是 **Extension Host**。

除了这三个主要的进程以外，还有两种特殊的进程。第一种是**调试进程**，VS Code 为调试器专门创建了 Debug Adapter 进程，渲染进程会通过 VS Code Debug Protocol 跟 Debug Adapter 进程通讯。

另一种则是**Language Server**，我们前面在介绍 VS Code 的语言支持时也提到过。

下面就是 VS Code 的进程架构图了。



VS Code 插件架构

在上图中，绿色的就是插件进程 Extension Host 了。VS Code 创建 Extension Host 进程的方式，就是创建一个新的 Electron 进程，并且以 Node.js 的形式运行。也就是说，这个进程就是一个完整的 Node.js 进程，Node.js 版本就是你使用的 Electron 中的 Node.js。

对于一个插件作者而言，你可以无需关心 VS Code 的这套架构，在书写 VS Code 插件的时候，你只需要知道：

- 首先，这个插件就是一个 Node.js 应用；
- 其次，在这个 Node.js 应用中，你可以直接访问 VS Code 的 API，通过这些 API 来操作 VS Code，你并不需要知道插件进程是怎么跟渲染进程通讯的；
- 最后，每当你打开一个窗口时，VS Code 会为这个窗口创建插件进程，并且按需要激活插件。也就是说，同一时间，你的代码有可能被运行多次。

如何创建一个插件

VS Code 的插件既然是一个 Node.js 应用，那么官方自然也会提供了基于 NPM 的工具链来帮助你创建和维护插件。

首先你需要的是 yeoman，一个脚手架工具。通过 yeoman 你可以快速创建代码模板，如下所示：

```
npm install -g yeoman
```

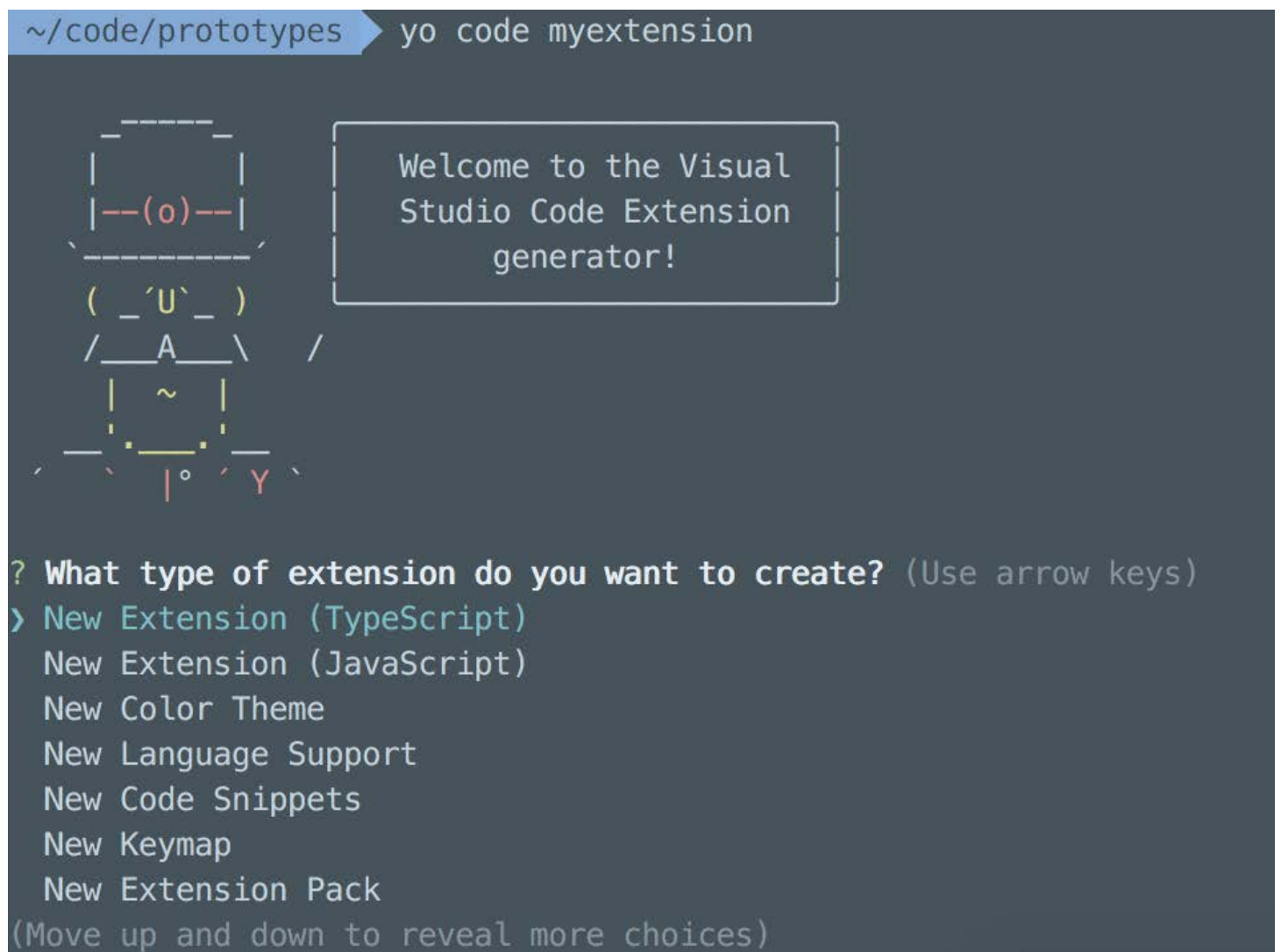
然后你需要安装 VS Code 的模板：

```
npm install -g generator-code
```

有了脚手架，你就可以创建一个 VS Code 的插件模板了。接下来运行：

```
yo code myextension
```

请注意，第三个参数将是你新创建的插件的文件夹名字。



VS Code 插件脚手架

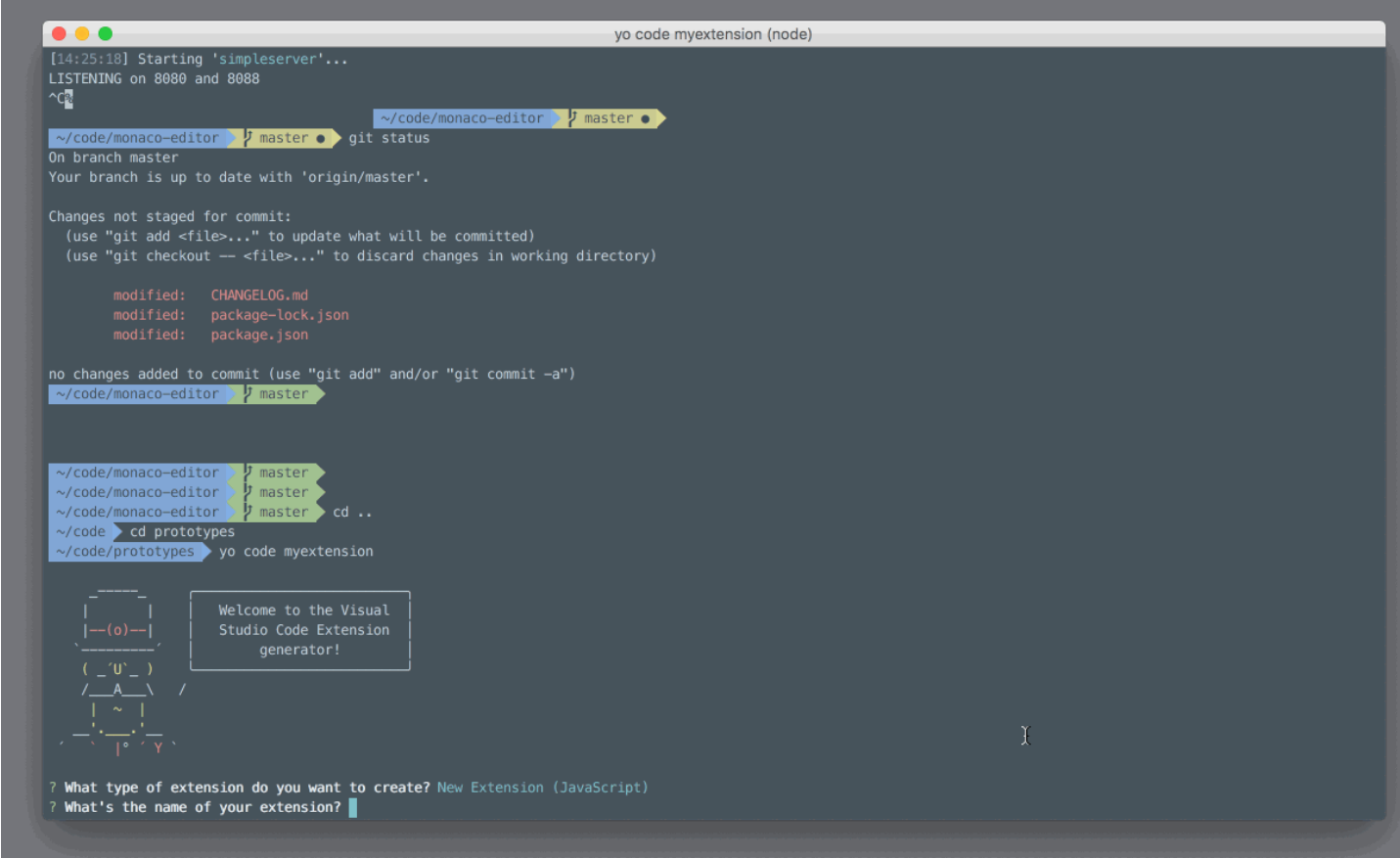
由上图，你可以看到有七个插件模板：

- 前两个是通过编程来提供插件功能，你可以选择 TypeScript 或者 JavaScript，结果都是类似的，因

为 TypeScript 最后也需要被编译成 JavaScript 再发布；

- 第三个是主题插件，你可以将你自己创建的主题分享给其他人；
- 第四个是语言支持，也就是语法高亮、语言定义等；
- 第五个是代码片段的分享；
- 第六个则是分享快捷键；
- 第七个就是对多个插件进行组合分享。

关于主题（Color Theme）、快捷键（Keymap）、代码片段（Code Snippet）的分享，我会在下一讲进行介绍。语言支持之后也会涉及。今天，我们先讲述第二个选项 **“New Extension (JavaScript)”**。



```
[14:25:18] Starting 'simpleserver'...
LISTENING on 8080 and 8088
^C
~/code/monaco-editor master
~/code/monaco-editor master git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   CHANGELOG.md
        modified:   package-lock.json
        modified:   package.json

no changes added to commit (use "git add" and/or "git commit -a")
~/code/monaco-editor master
~/code/monaco-editor master
~/code/monaco-editor master
~/code/monaco-editor master cd ..
~/code > cd prototypes
~/code/prototypes yo code myextension

Welcome to the Visual
Studio Code Extension
generator!

? What type of extension do you want to create? New Extension (JavaScript)
? What's the name of your extension? 
```

VS Code 插件脚手架生成代码

接下来，你会依次被提示输入插件的名字、介绍、想要用哪个账号发布、是否要打开 type check 以及是否要使用 git 等。你可以暂时按照我的样例进行输入，之后也可以再根据需要修改。

输入全部问题后，脚本就会自动地创建文件，安装需要的 dependencies。全部结束后，脚本会提示你，可以运行下面的脚本打开这个插件的代码。

```
cd myextension
code .
```



VS Code 插件示例

VS Code 的脚手架，默认为我们创建了不少的文件。不过像 .gitignore、.eslintrc.json、[README.md](#) 这些文件的作用想必你已经比较熟悉了。对于这个插件而言，最重要的是下面几个文件：

- package.json 我上面提到了，VS Code 的插件就是一个 Node.js 的应用，package.json 里记录了这个 Node.js 应用的信息。同时，插件的信息也会被记录在这个文件内。
- extension.js 这个文件是当前插件的全部代码。
- .vscode 脚手架工具已经为我们提供了调试配置、任务配置等，有了它们，我们就不用自己花时间书写了。

好了，下面我们来看看 extension.js 和 package.json。看完它们，你就对 VS Code 插件是如何运行的有很好的理解的。

extension.js的内容在删除了所有的注释后，如下：

```
const vscode = require('vscode');

function activate(context) {

    console.log('Congratulations, your extension "myextension" is now active!');

    let disposable = vscode.commands.registerCommand('extension.sayHello', function () {
```

```
vscode.window.showInformationMessage('Hello World!');

});

context.subscriptions.push(disposable);
}
exports.activate = activate;

function deactivate() {
}
exports.deactivate = deactivate;
```

第一，我们引用了 `vscode` 这个库。通过引用这个库，我们就能够使用 VS Code 的插件 API 了。

第二，我们创建了 `activate` 函数并且将其输出。VS Code 的插件进程在激活这个插件时，就是调用这个被输出（`export`）的函数。也就是说，这个函数，就是这个插件的入口。

相对应的就是 `deactivate` 函数，当我们禁用这个插件或者关闭 VS Code 时，这个函数就会被调用了。

下面我们再来看看 `activate` 这个函数：

```
function activate(context) {
  console.log('Congratulations, your extension "myextension" is now active!');
  let disposable = vscode.commands.registerCommand('extension.sayHello', function () {
    vscode.window.showInformationMessage('Hello World!');
  });

  context.subscriptions.push(disposable);
}
```

这个函数首先输出了 `log`，告诉我们插件已经被成功激活了。接着，我们使用 `vscode.commands.registerCommand` 注册一个名为 `extension.sayHello` 的命令，这个命令的实现，是 `registerCommand` 的第二个参数，我们通过调用 `vscode.window.showInformationMessage`，在界面上调出一个提示框，内容则是 `Hello World!`。

不过，光有 `extension.js`，这个插件是无法运行的。VS Code 会根据条件来激活插件，而这个激活条件写在了 `package.json` 中，那么我们一起看下 **package.json**。

```

{
  "name": "myextension",
  "displayName": "myextension",
  "description": "my extension",
  "version": "0.0.1",
  "publisher": "rebornix",
  "engines": {
    "vscode": "^1.29.0"
  },
  "categories": [
    "Other"
  ],
  "activationEvents": [
    "onCommand:extension.sayHello"
  ],
  "main": "./extension",
  "contributes": {
    "commands": [
      {
        "command": "extension.sayHello",
        "title": "Hello World"
      }
    ]
  },
  "scripts": {
    "postinstall": "node ./node_modules/vscode/bin/install",
    "test": "node ./node_modules/vscode/bin/test"
  },
  "devDependencies": {
    "typescript": "^2.6.1",
    "vscode": "^1.1.21",
    "eslint": "^4.11.0",
    "@types/node": "^8.10.25",
    "@types/mocha": "^2.2.42"
  }
}

```

上面这个文件，跟普通的 npm 的 package.json 只有三处不同。

第一处是 engines。

```
"vscode": "^1.29.0"
```

它指定了运行这个插件需要的 VS Code 版本。比如 “^1.29.0” 就是说明，要安装运行这个插件必须要使用 VS Code 1.29 及以上版本。

第二处是 activationEvents。

```
"activationEvents": [  
  "onCommand:extension.sayHello"  
]
```

这个属性指定了什么情况下这个插件应该被加载并且激活。在我们这个例子里，激活条件是，当用户想要运行 “extension.sayHello” 这个命令时，就激活这个插件。

这个机制能够保证，当我们需要使用这个插件的时候，这个插件才被激活，尽可能地保证性能和内存使用的合理性。

第三处是 contributes。

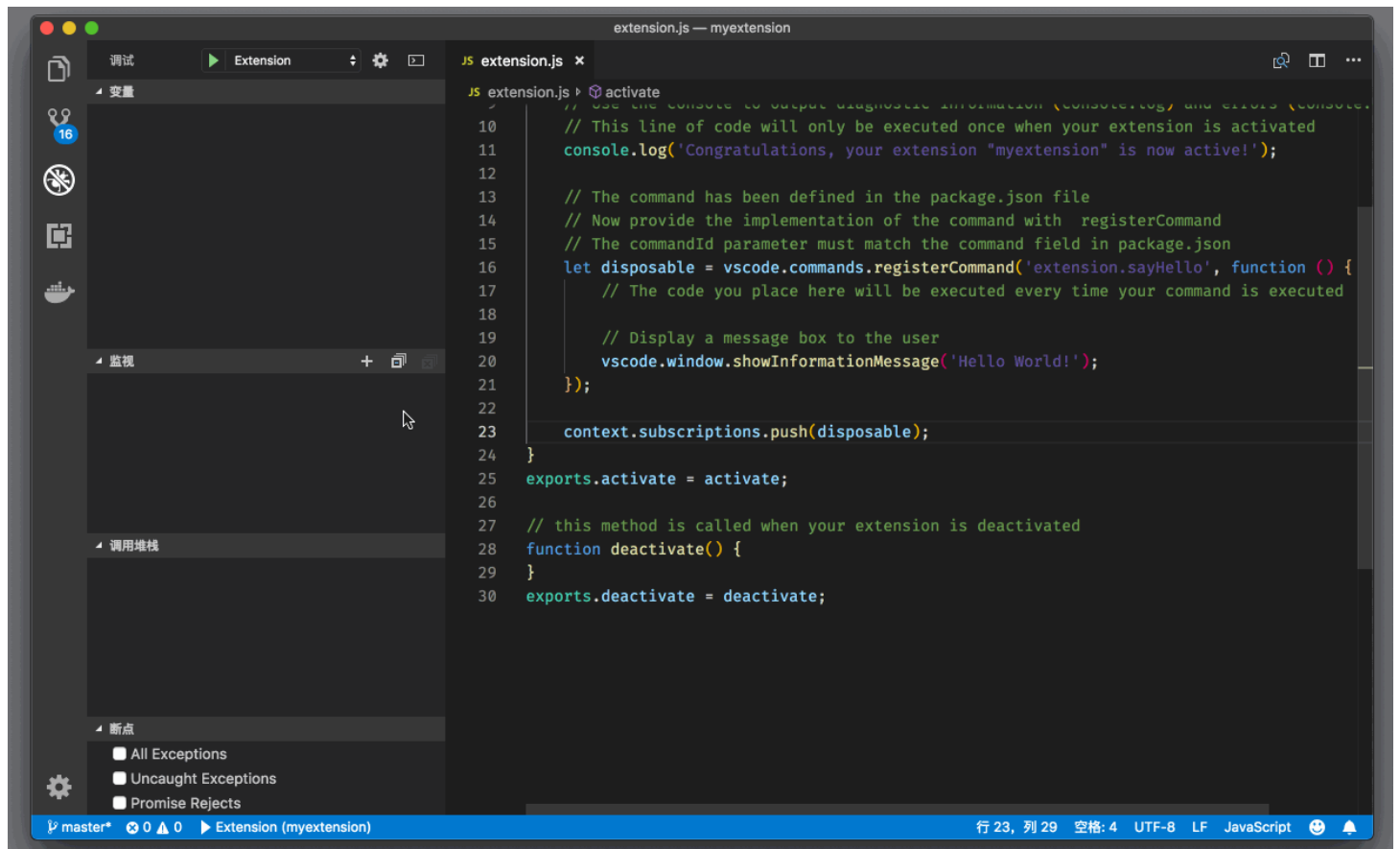
```
"contributes": {  
  "commands": [  
    {  
      "command": "extension.sayHello",  
      "title": "Hello World"  
    }  
  ]  
},
```

这个属性指定了，我们这个插件给 VS Code 添加了一个 command，这个 command 的 id 是 “extension.sayHello”，跟 extension.js 中写的一样。而这个命令的名字，叫做 Hello World。

如果不写这个属性的话，VS Code 是不会把这个命令注册到命令面板中的，我们也就没法找到这个命令并且执行了。

运行插件

好了，现在我们对这个插件的实现方式和注册方式已经有了了解，下面就到了运行和调试代码的时候了。VS Code 的插件代码脚手架已经为我们提供了 `launch.json`，所以我们只需要按下 `F5` 即可启动代码。



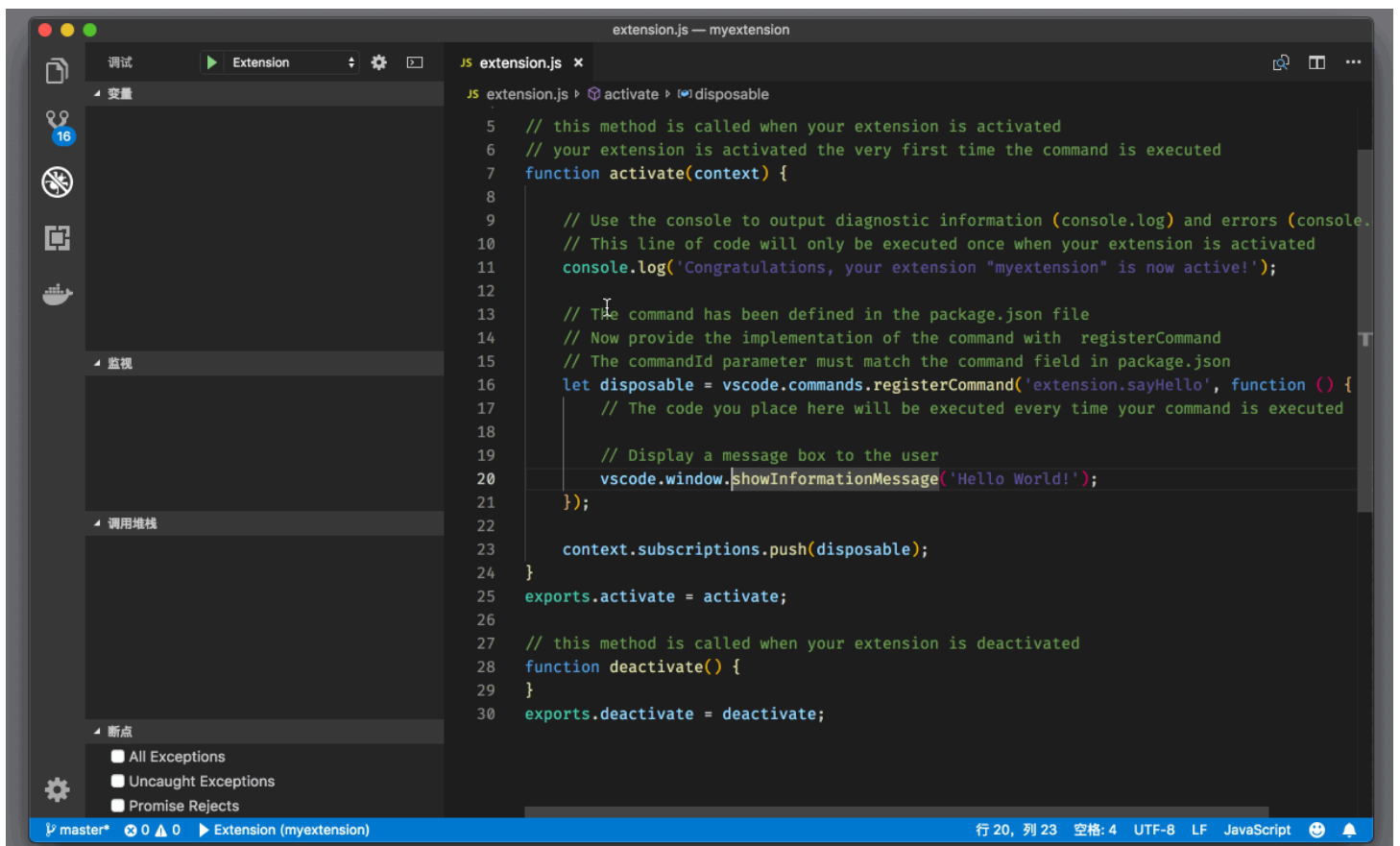
运行插件

代码启动后，VS Code 会打开一个新的窗口，这个窗口中就运行着我们本地书写的代码。此时我们打开命令面板，搜索 “Hello World” 并且执行。



运行 Hello World 命令

上面我提到了，这个插件只有在“Hello World”命令被执行时才会被激活。那下面我们不妨把这个调试窗口关掉，然后再 activate 函数中加上断点，重新试一次。



从上面的动图里你可以看到，当“Hello World”命令被执行时，首先被唤起的就是 activate 函数，然后是“showInformationMessage”被执行。

小结

好了，以上就是今天内容的全部。我们还没有深入到 VS Code 的插件 API 中去，只是带你了解一下 VS Code 的插件架构，插件示例里的几个重要文件的作用，以及成功地将一个插件运行起来并且激活、执行命令。更多的内容，我在接下来的几篇文章里会逐步介绍，当然，你也可以自己修改代码，VS Code 可是为 JavaScript 代码提供了不错的智能提示，相信你可以轻松找到 VS Code 有哪些可用的插件 API

。



精选留言



Moorez

npm install -g yeoman 这个命令现在已经废弃了

变成 npm -g install yo

文档 <http://yeoman.io/migrate.html>

2018-11-25 00:16



谢mingmin

终于等到插件开发了

2018-11-25 09:35



L

太好了

2018-11-24 10:49

