37讲插件开发（六）：VSCode插件维护和发布要点

今天，就到了插件开发的最后一讲了。前面的内容，我们主要在介绍 VS Code 的插件 API 是如何使用的，有哪些分类。今天我们换个思路，先看看 VS Code 的插件 API，在设计上有哪些通用之处，以及有哪些是我们在写插件时要注意的。另外，我们还会一起看一下，在书写 VS Code 的插件时，使用 Node.js 模块时有什么注意事项。最后我会介绍一下插件发布里的一些注意事项和技巧。

好了，闲话少说，让我们直接开始今天的内容吧。

插件 API 的 Design Pattern

VS Code 的插件 API 的发布流程首先是发出提议 Proposal，看看社区的反馈如何。这一类 API 会出现在 vscode.proposed.d.ts 文件中，而稳定版本的 API 则是在 vscode.d.ts 里。

一个 API 进入 proposed 状态并不需要什么流程，但是要进入 stable 的话，就要经过整个团队的 review 了。基本上，一个 API 要发布到 stable 中，需满足以下几个条件：

- 首先，插件 API 不会将 UI 直接暴露给插件。VS Code 的界面（也就是 DOM）的渲染完全由 VS Code 控制，插件 API 可以做的，就是将 UI 上的渲染逻辑翻译成 Data Provider 的形式，插件提供内容，VS Code 负责渲染。
- 其次，如果一个 API 的运行时间可能比较长，那么这个 API 应该支持 Promise，并且可以取消（也就是我们在 vscode.d.ts 里常看到的 Cancellation Token）。
- 最后，插件 API 能够正确地处理对象的生命周期。VS Code 使用了 Dispose 模式，大部分 VS Code 插件 API 生成的对象，都会拥有一个 dispose 函数属性，然后运行这个函数就可以将这个对象销毁。

基于上面的这些 API 设计原则，我们也能够得出一个好的插件实现应该有如下特性：

- 对于长时间运行的任务，如果用户选择取消，那么插件应该能够终止任务。
- 插件能够及时地删除不再使用的对象，以及正确的时候 dispose VS Code 生成的对象，减少内存的使用。
- 插件在给 VS Code 插件 API 提供数据的同时，能够做到增量更新，尽可能地减少 VS Code 重新渲染组件。

只有做到上面这些，才能尽可能地保证插件的性能。插件提供的功能是一方面，但是如果性能出众的话，就真的是一个好插件了。

Node.js 模块使用

我们前面几讲提到过，**VS Code 的插件，其实就是一个 Node.js 应用**。那么如何管理 Node.js 的 dependencies，也是插件应该关心的。在使用第三方的 Node.js 模块时，要注意以下几点：

第一，很多简单的功能，其实可以自己实现，过多地使用第三方模块，会导致代码量不必要地增大。代码量增大，就相应地减慢了插件的下载和更新。同时插件被激活时，需要加载各个 Node.js 模块，模块越多，速度也就越慢。所以，**使用模块要克制**。

第二，如果可以的话，借助 webpack 对插件进行打包，并且开启 treeshaking，把没有使用的代码删除掉。和上一条的原因是一样的。

第三，对于性能要求比较高的应用，你可以考虑使用 Node.js 的 Native Module 或者 Web Assembly。最新版本的 VS Code 里，已经支持了 Node.js 新的 Native Module API （N-API）和 Web Assembly了。不过这两者之间也各有优劣：

1. 在 NAPI 之前，大家都在使用 NAN 来管理 Node.js Native Module，但是一旦 VS Code 升级了 Electron，导致 Node.js 版本发生变化，所有的 Native Module 就不能工作了。NAPI 的出现，解决了这个问题，你再也不用担心 Electron 升级的问题了。但是 NAPI 也并没有解决发布的问题，你依然得为每个不同的平台（Windows，macOS，Linux）分别编译 Native Module，应该说比较麻烦。
2. 相比于 Native Module，使用 Web Assembly 就要好很多，因为 Web Assembly 天然就是跨平台的。但是它也有缺点，那就是无法访问系统 API，所以如果你的代码必须要访问到一些原生的 API，可能还是得用 Native Module。

以上我提的几点，相信你也看出来了， 重点依然是性能。对于大部分插件而言，business logic 都不是特别复杂，而性能往往就是区分度，所以如果能够借助 Native Module、Web Assembly，再或者 Webpack 等打包工具，给你的插件代码提提速，那就非常给力了。

发布

我们插件部分一直都还没有介绍插件的最终发布，不过不用担心，这个内容跟插件 API 相比，可就简单多了。你只需创建一个 Visual Studio Online 的账户，然后使用 vsce 这个 npm 包就能发布了。现在 Marketplace 更是允许你直接在后台发布，而无需使用命令行。关于更多的细节，还请阅读官方文档。这里我想讲一讲 VS Code 插件的版本管理和依赖。

在插件的 package.json 文件中，有这样一个配置：

```
"engines": {
    "vscode": "^1.29.0"
}
```

这段配置的意思是：这个插件至少要求用户安装 1.29 版本的 VS Code 。“^1.29.0” 的书写方式，也跟 npm 包的版本书写方式一模一样。

那么我们什么时候需要更新这个 engine 值呢？我的建议是：**当且仅当你使用了某个新的 API，而这个新 API 要求了用户必须使用某个版本的 VS Code时，就值得你去更新这个 engine 值了**。更新完之后，只有新版本的 VS Code 用户，才会收到插件的更新，也就是说如果用户还在使用老版本的话，就不会收到更新了。

不过你也不必担心更新了 engine，导致用户量的减少，因为VS Code 的大部分用户，都会在新版本发布之后的一到两个月更新到最新的版本，也就是说，很快你的用户数量就会恢复正常。而且在用户还没有完全升级的情况下，如果有什么 bug，你还可以及时修复，而不会波及太多的用户。

小结

好了，以上就是插件开发相关的全部内容了。正如我一直强调的，VS Code 的插件开发，跟开发一个 Node.js 应用没有区别，而你要使用的 API，都写在 vscode.d.ts 这个 typings 文件里。如果你想看看这些插件 API 的样例代码，也可以自行下载试试看。希望你也可以写出不错的插件，提升自己和大家的工作效率！

精选留言

Maiza

哇 这么就结束了 意犹未尽 哈哈

2018-12-06 22:08

一步

跟着专栏学习了两个多月了，对 vscode 更加了解了，各个功能主键在哪，怎么用，怎么设置都清晰了

最工具的熟悉直接的体现就是工作效率的提升

现在专栏结束了，希望老师在出几期文章，来解答一些文章疑问，或者对 vscode 的 github 源码仓库的简单讲解什么的，或者分析一下 vscode 整体设计的架构

2018-12-06 13:58