22讲基于自然语言或者纯文本的设置界面，总有一款适合你

在上一讲中，我简单介绍了 VS Code 编辑器布局发展的历史，通过对其发展史的一定了解，可以有助于我们更好地掌握快捷键和命令的使用。

今天的内容，我会分“两步走”。

第一步，介绍如何在VS Code中寻找和修改配置、快捷键，而且这一部门的内容，我依然会像上一讲一样，以 VS Code 的历史为基础，带你了解 VS Code 的设置和快捷键配置的 UX 是如何一步步演变过来的，以及它们各自的优、缺点是什么。第二步，也就是在文章的最后，我会介绍如何使用文本编辑器，对 VS Code 的快捷键进行高级定制。

修改设置

相信你早就熟悉了，VS Code 是以文件和文件夹为核心的，用户的设置、快捷键绑定等，也都是以文件的形式存储在用户的机器上。同时，VS Code 把这一切都开放给用户，也就是说你可以直接对这些文件进行修改。

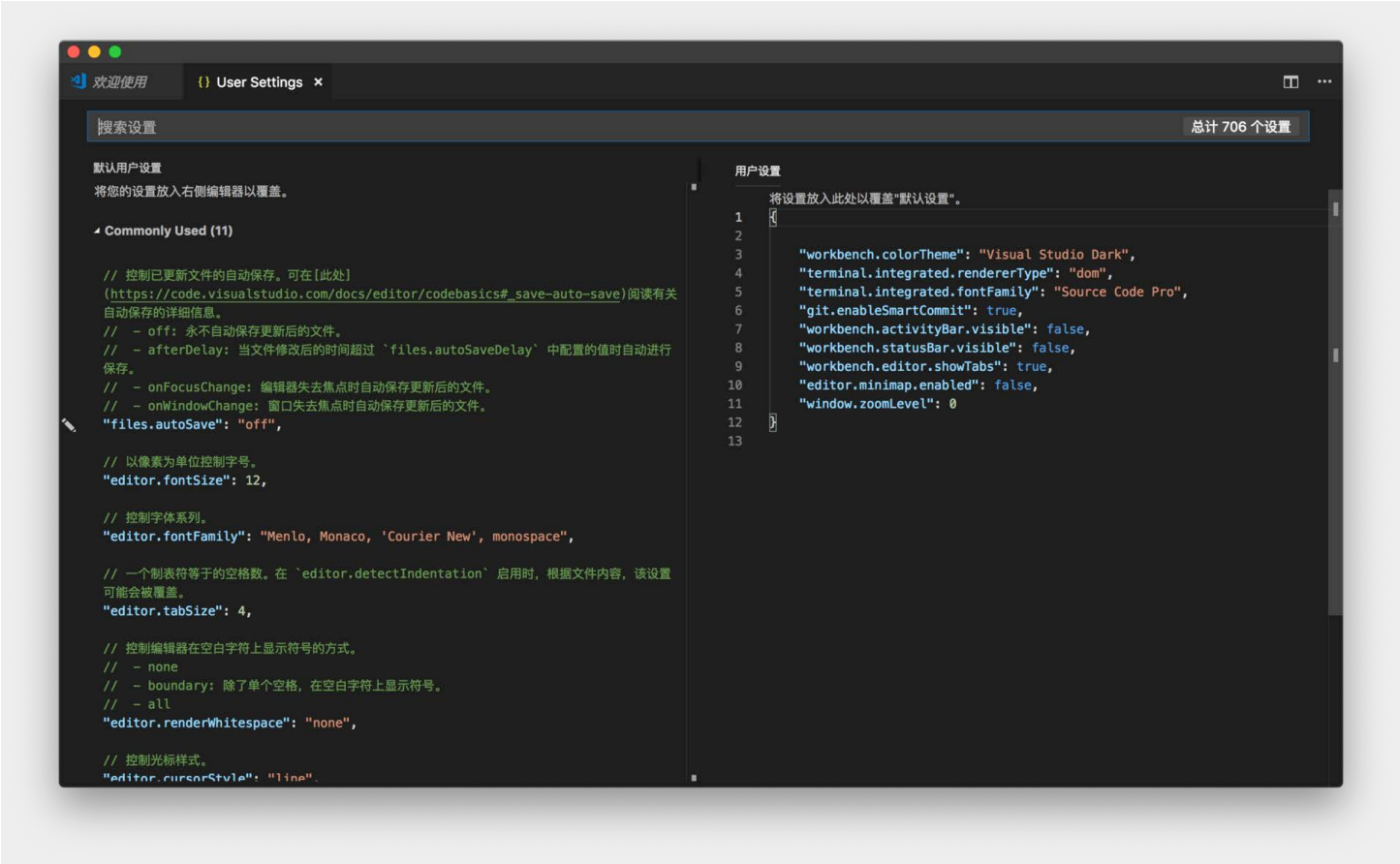
文本编辑器

相信通过学习专栏之前的文章，你已经尝试过了修改代码片段（Code Snippet）配置、修改快捷键绑定以及修改个人设置等操作。不过还有些操作我还没有做过介绍。

VS Code 的最新稳定版里，有两个不同的设置编辑器。下面，让我们先打开命令面板，搜索 “打开设置(JSON)”[Open Settings(JSON)]，然后执行。

打开设置(JSON)

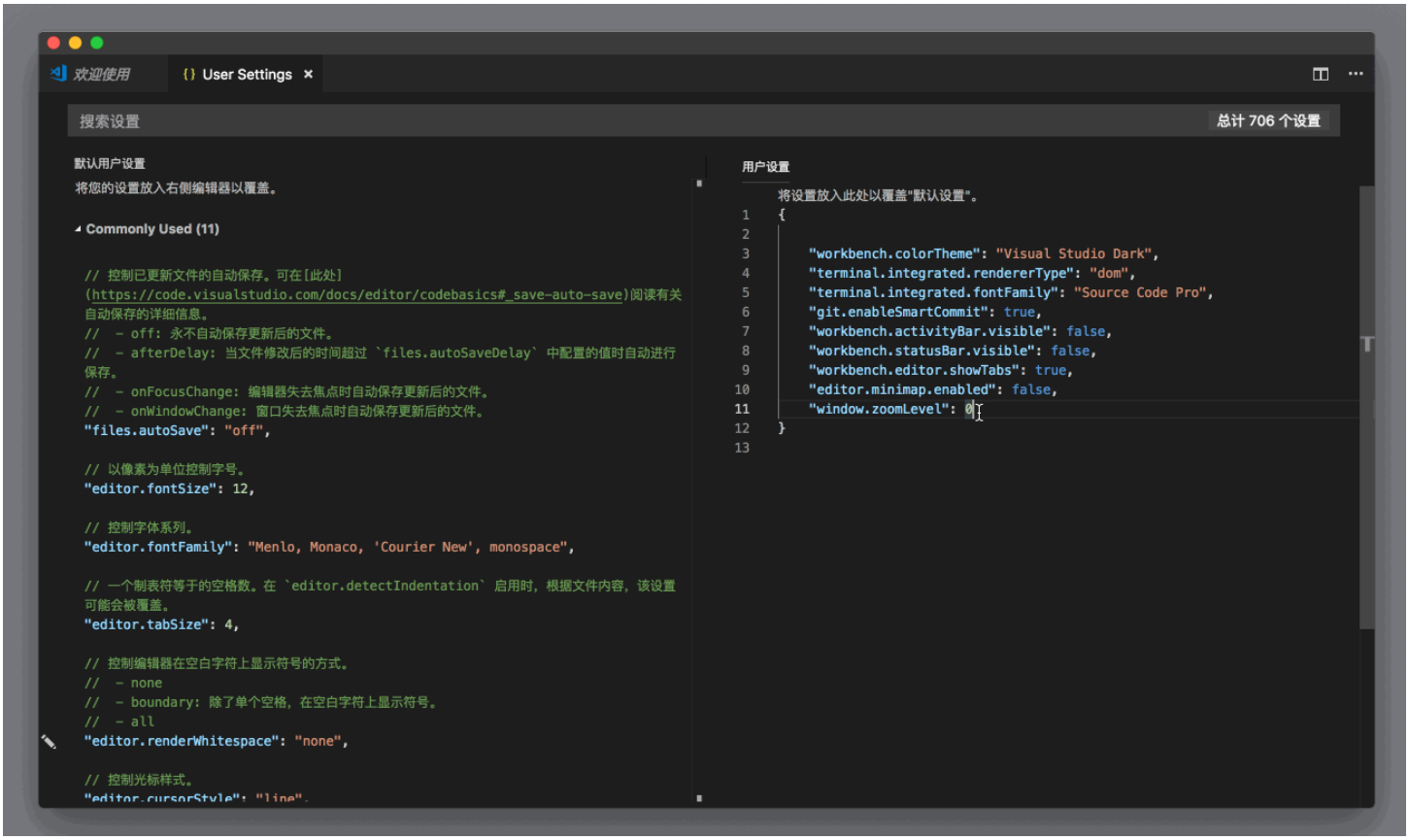
此时，我们能看到并排的两个编辑器。



并排的编辑器

编辑器左侧是 VS Code 支持的各种设置；而右侧，则是我们非常熟悉的 JSON 文本。而且在整个界面的最上方，还有一个输入框，以供我们搜索设置。

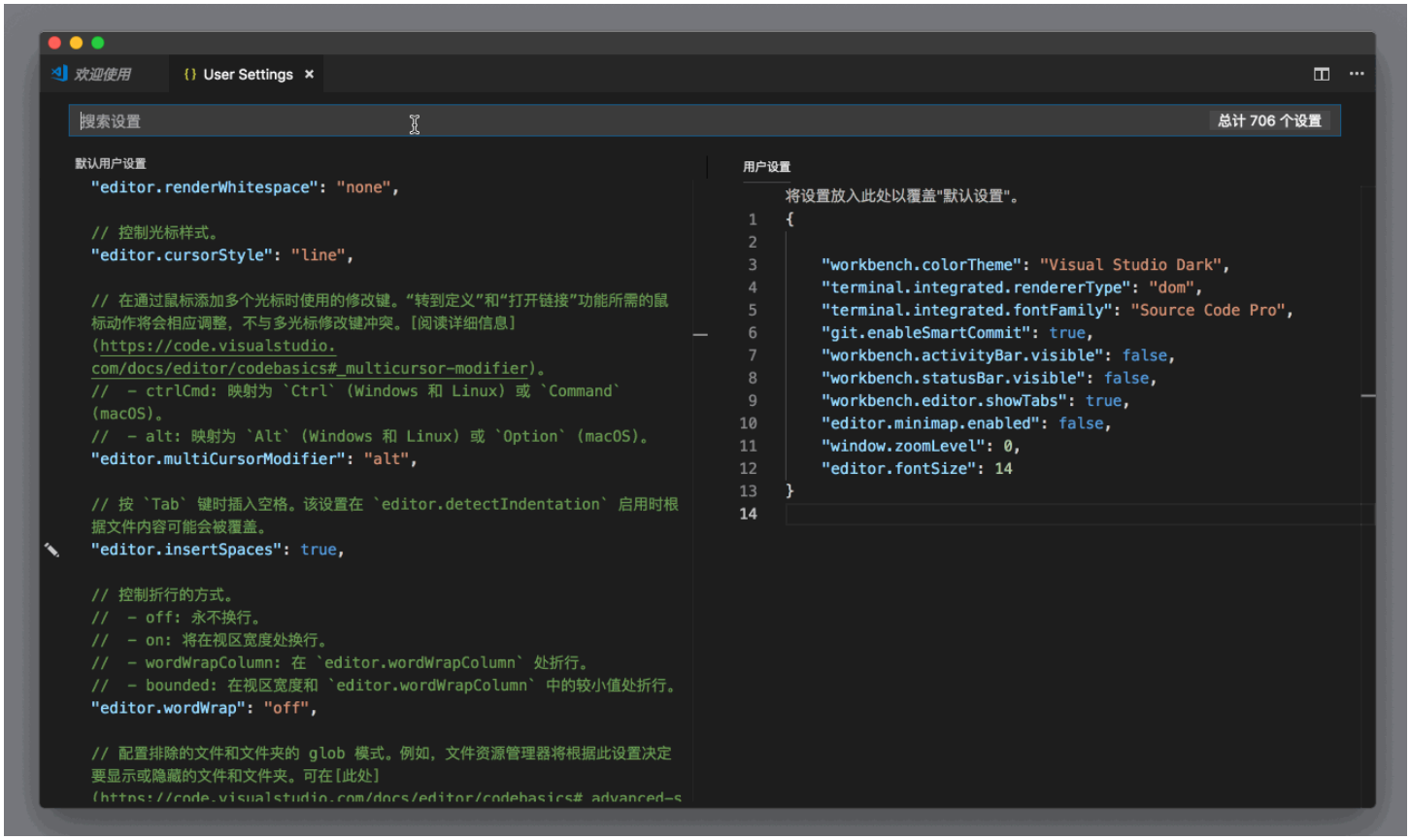
VS Code 早期修改设置的方式有两种。第一种就是在左侧找到设置，然后“复制粘贴”到右侧。第二种是直接在右侧修改，VS Code 为右侧的编辑器提供了自动补全功能。



修改设置自动补全

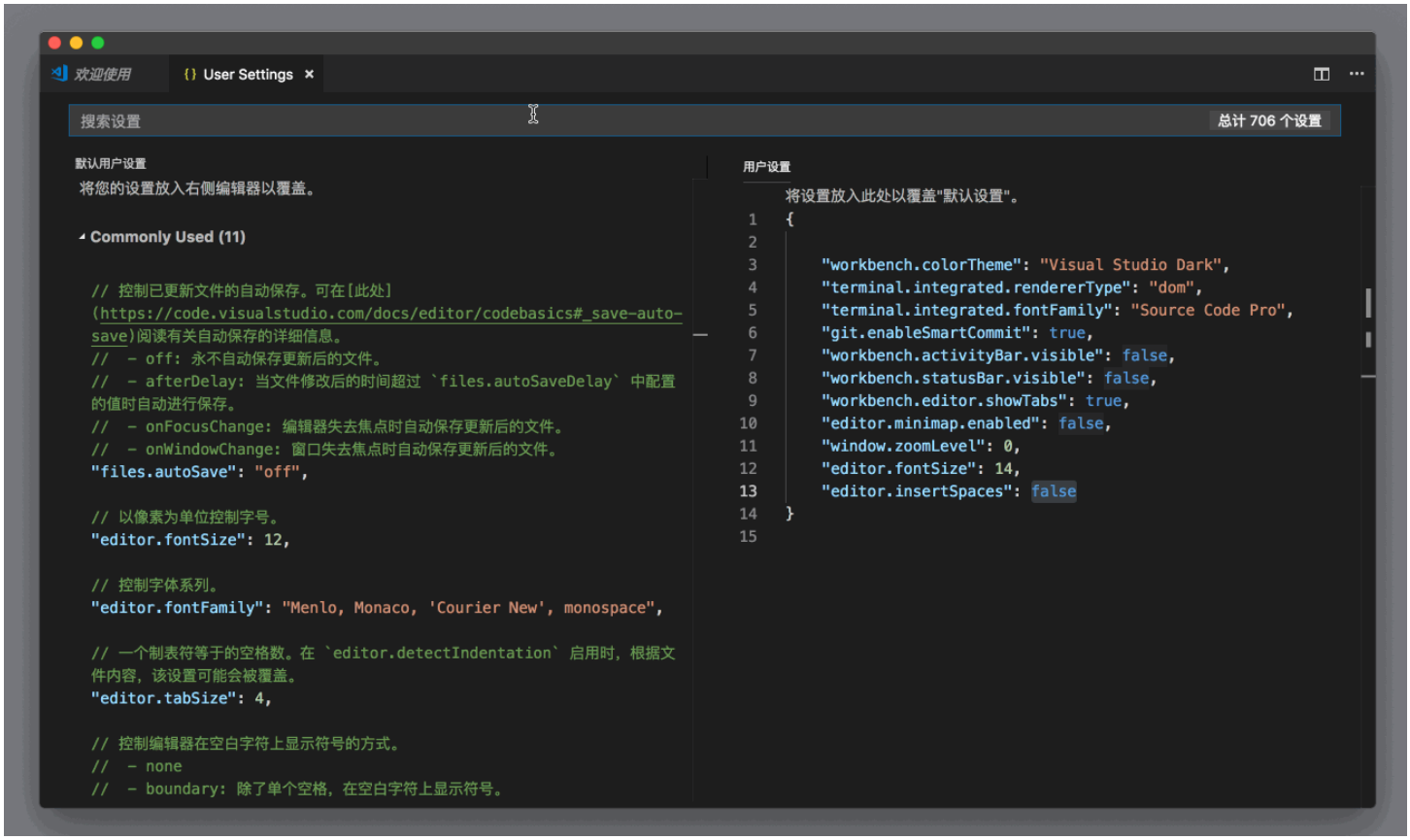
在上面的动图中你能看到，我通过输入 fontsize，依靠自动补全找到了 editor.fontSize 这个设置，然后输入到文件中，保存后编辑器的字体大小很快发生了改变。所以通过修改这个设置文件，我们就能够第一时间看到修改设置后的变化。

当然，我们也可以直接在搜索框里输入设置的名称，找到对应设置后，将鼠标移动到行号附近；这时候我们能看到一个铅笔形状的图标，点击它，就能够看到这个设置允许的值有哪些；选择好我们想要的那个配置后，这个设置就会被自动写到右侧的 JSON 文件里。



搜索设置并修改

说到设置的搜索，其实最初 VS Code 只会进行单词的模糊匹配，也就是说如果你不小心打错字了，VS Code 就找不到合适的设置了。不过，很快 VS Code 就为这个搜索增加了自然语言的处理。还是使用上面的例子，当我想修改字体的大小时，我可以在搜索框里输入 how to set editor font size。即便我不小心把 font size 打错成了 fontize ，VS Code 也能找到正确的设置项。



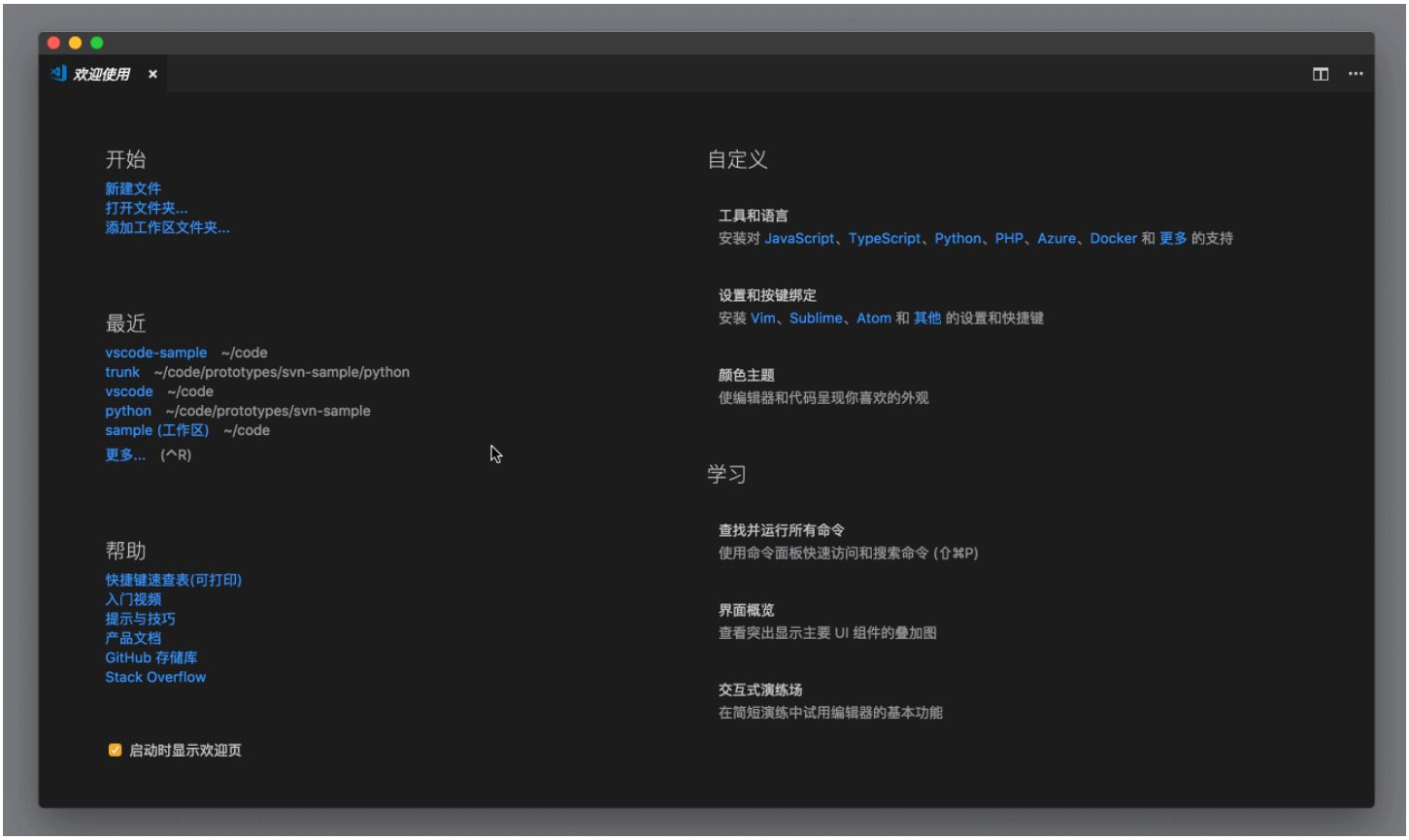
自然语言搜索

虽然VS Code有很不错的搜索和自动补全，但是我刚使用它的时候，还是挺不适应的。不过这属于 VS Code 设置思路的一部分，就是把细节暴露给用户，让用户也知道设置是如何存储的、格式是怎样的，等等。像我自己在习惯这套搜索方式后，再跑到某些默认打开就是图形化设置界面的编辑器时，反而有点手足无措，我会纠结该如何才能找到我想要的设置。

当然，做产品不能固步自封。对于陪伴 VS Code 一路走来的用户而言，他们已经非常熟悉和习惯这套理念和操作方式了。可是对于新用户而言，VS Code 基于文本的设置方式，之于他们反而成为了一个障碍。很多新用户会觉得，“我不过是想改一个字体大小，为什么 VS Code 要给我打开两个并排的编辑器，甚至有的时候还需要我自己去修改设置文件？”

图形化界面

为了解决这个问题，VS Code 在最近的几个版本里添加了一个图形化界面。我们可以在命令面板搜索“打开设置(UI)”并执行。



设置 UI 界面

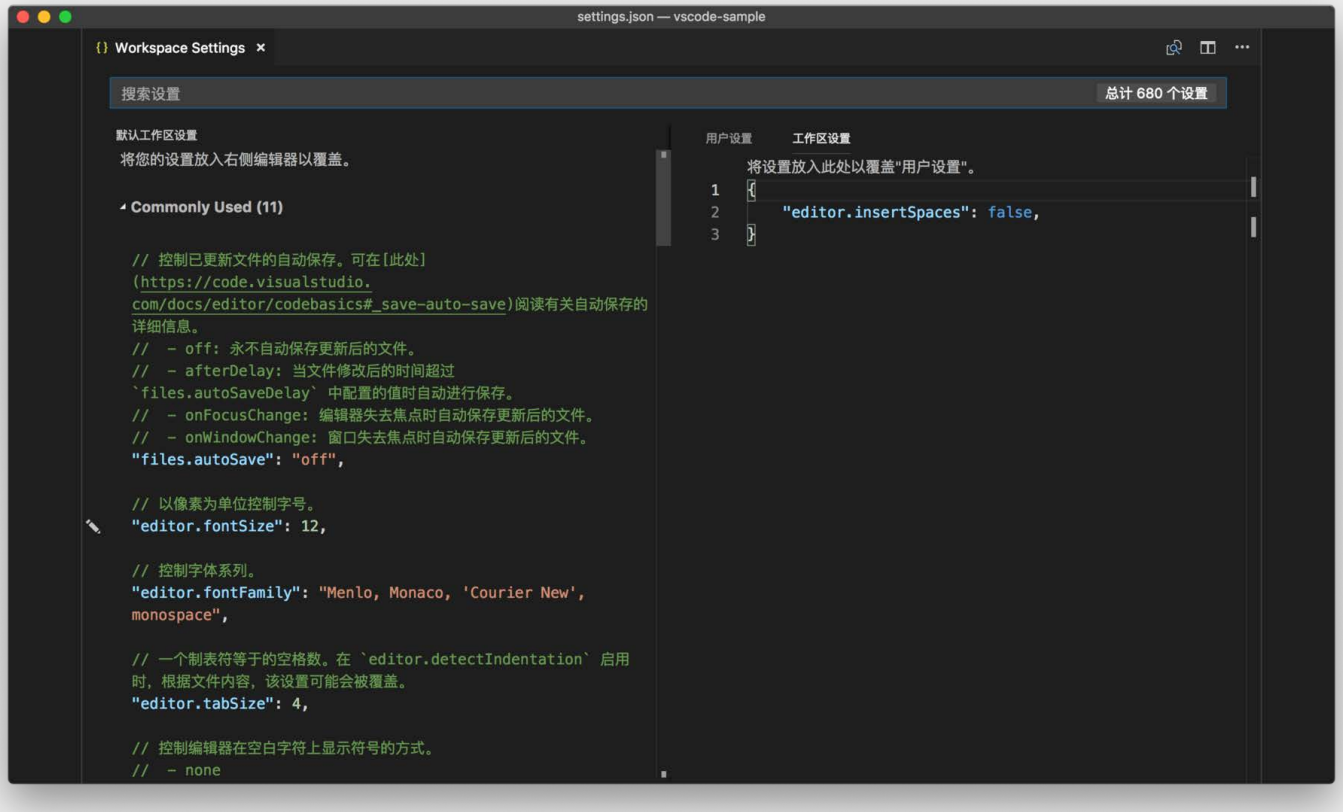
使用这样的图形化界面进行设置修改，好处当然也是很明显的：它能尽可能地减少修改配置的难度，并减少我们犯错的可能。

这里值得注意的是，图形化设置界面是在自然语言搜索之后才出现的，也就是说，虽然我们现在面对的是下拉框、复选框之类的，但我们依然能够使用跟前面一样强大的搜索功能。

工作区支持

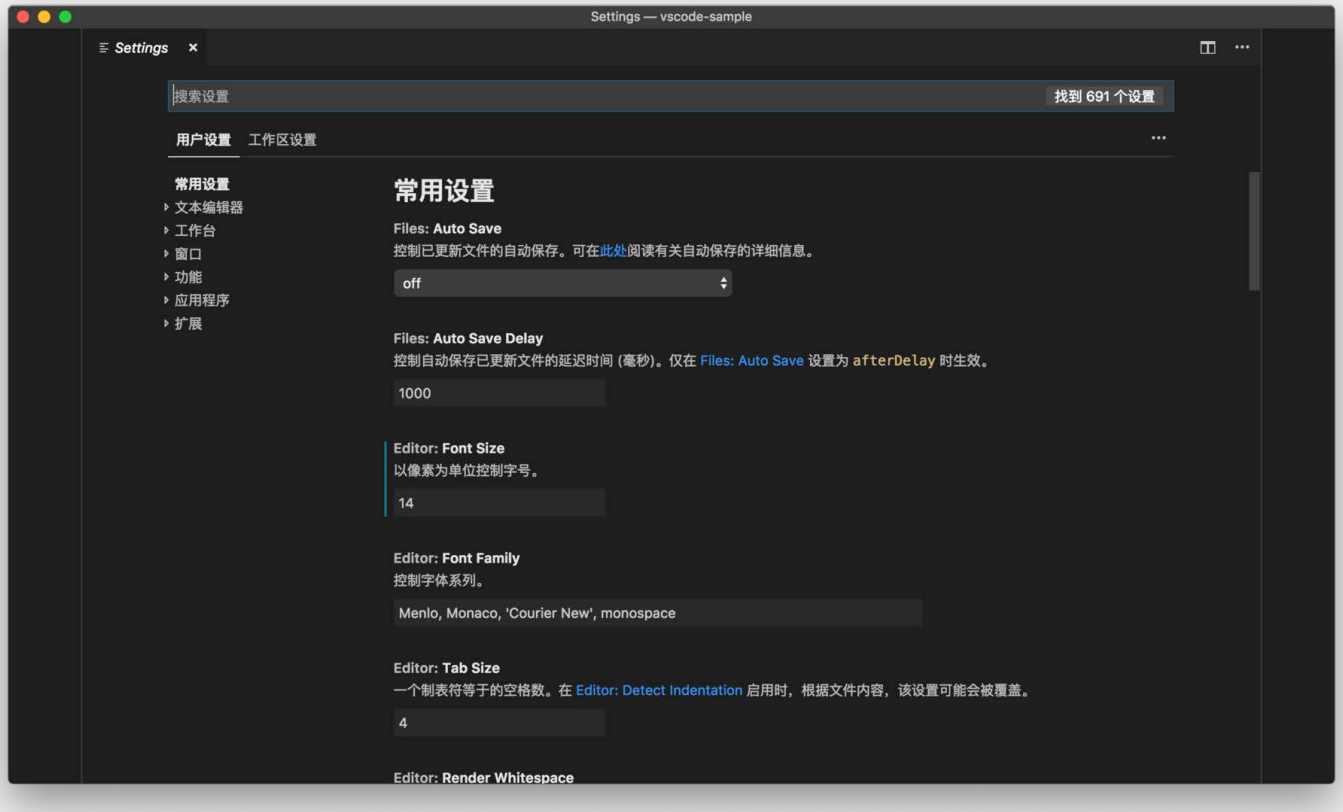
在上面的操作截图里，我们可以看到VS Code 的窗口里并没有打开任何的文件夹。而如果我们打开了一个文件夹，而且这个文件夹下有 .vscode/settings.json 文件的话，此时我们再打开设置界面，则能够看到一些变化。

比如在基于文本的设置界面里，我们可以在右侧的 JSON 编辑器里看到 “用户设置”和“工作区设置”两个选项。如果我们修改工作区设置的话，它会立刻被保存到 .vscode/settings.json 里。



工作区设置

而在图形化设置界面里，我们也同样可以在搜索框下进行个人和工作区设置的选择。

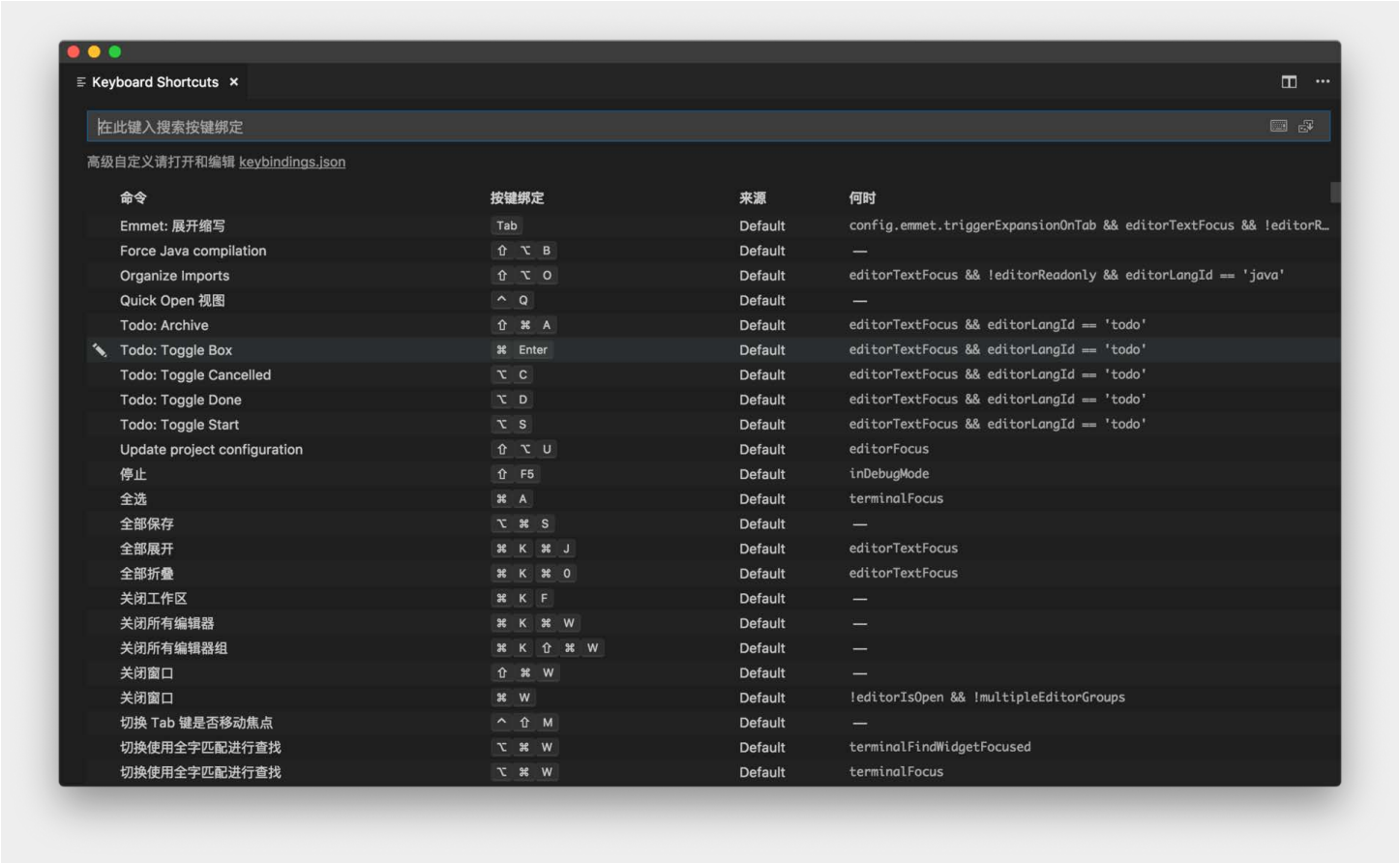


修改快捷键

介绍完设置，我们再来说说快捷键绑定。下面我主要从图形化界面和文本界面两个角度来阐述。

图形化界面

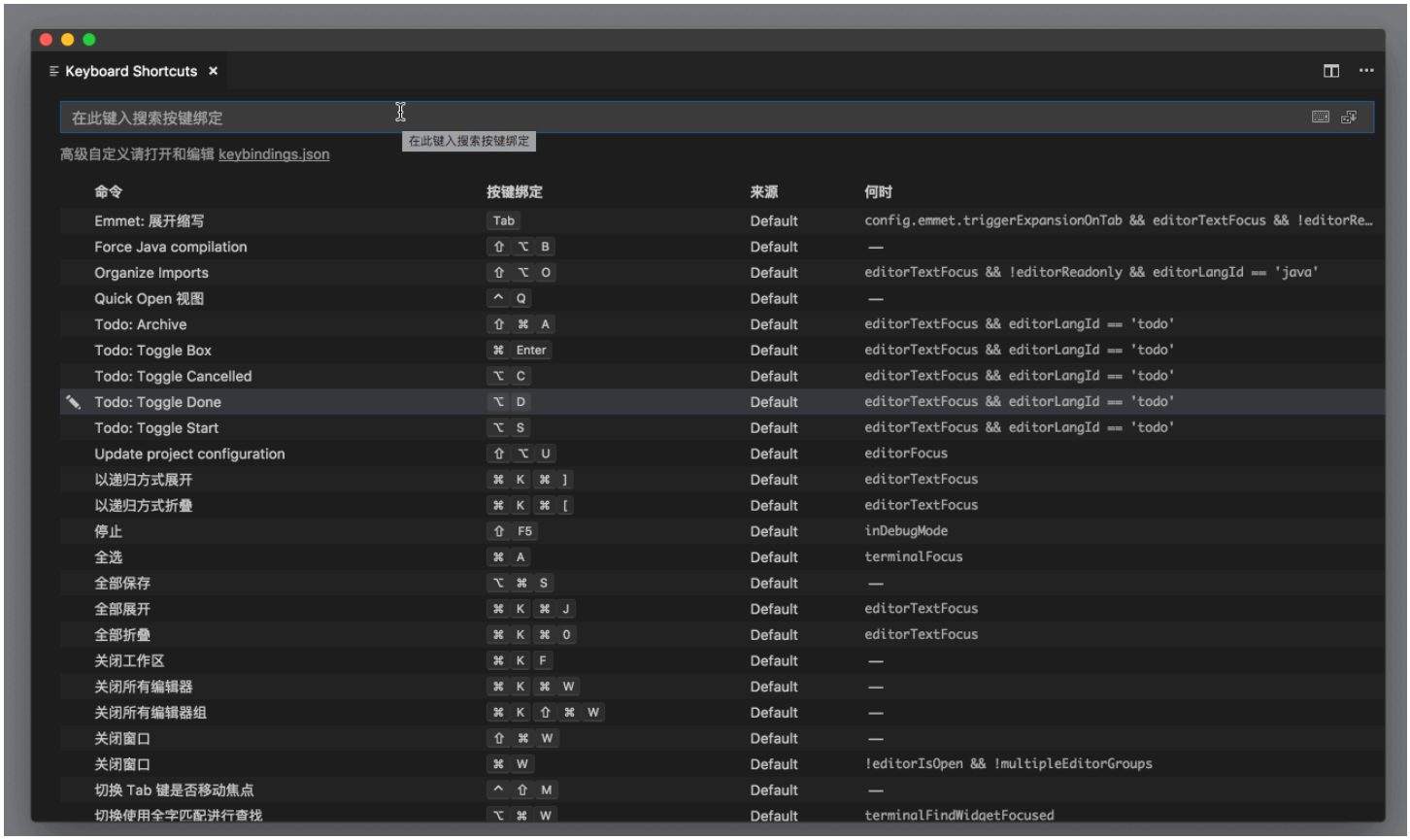
在专栏的最开始，我就介绍了如何使用快捷键界面去搜索和修改界面——在命令面板里“打开键盘快捷方式”（Open Keyboard Shortcuts）并执行。



快捷键修改界面

修改某个命令的快捷键也很简单，搜索并找到命令后，双击命令项就可以对快捷键进行修改了。如果你不熟悉的话，可以先复习一下第4讲“如何做到双手不离键盘？”的内容。

不过，这里我要介绍一个 VS Code 本月新增的功能（注意你需要更新到最新版本的 VS Code）。我们有的时候发现某个快捷键执行的命令不是我们想要的，这说明可能某个插件或者我们自己的配置有问题。要想看看某个快捷键究竟绑定在哪个命令上，我们可以在快捷键界面的搜索框里输入这个快捷键对应的字符。比如我可以在搜索框里输入 “cmd+s” 来看看它是不是绑定到了文本保存这个命令上。

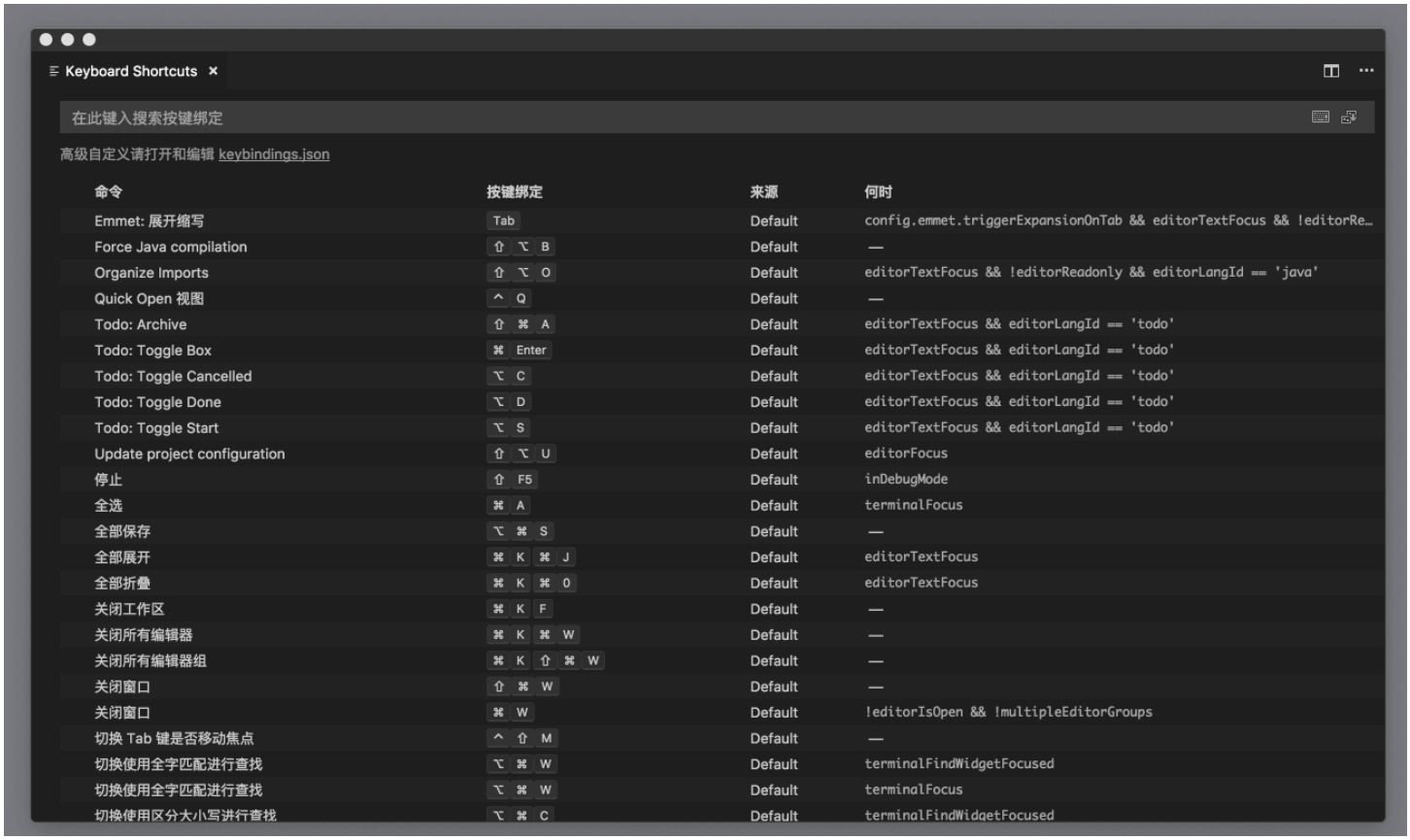


搜索 cmd+s

不过要手动输入 “cmd+s” 还是很麻烦的，更不要说如果我们要输入的是 “cmd+option+shift+s” 之类的复杂快捷键了。在最新版本里，这个搜索框的最右侧，多出一个小按钮。

输入框功能按钮

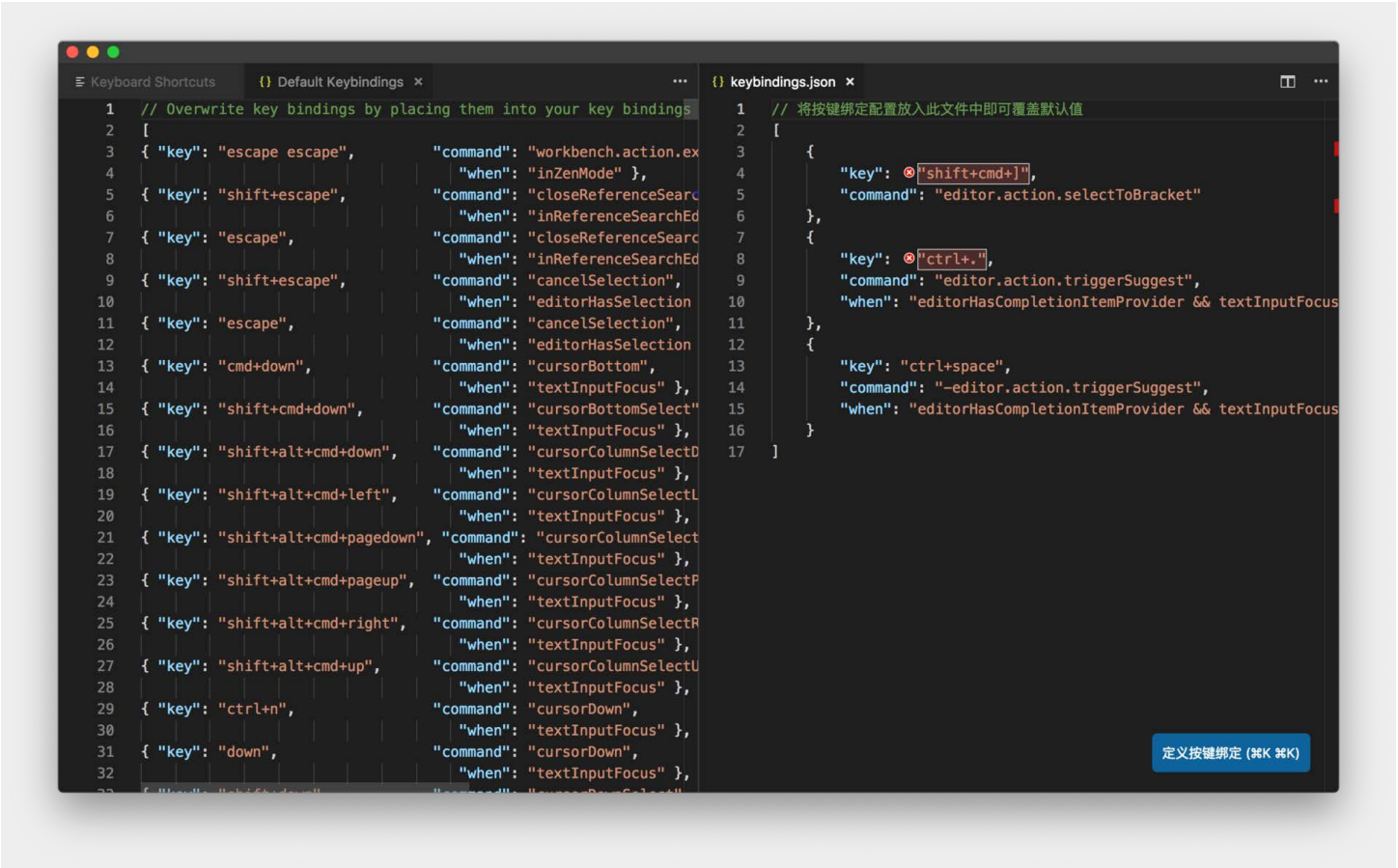
这个按钮的作用是**键盘录制**。如果我们点击这个按钮，激活键盘录制模式，此时，当我们在这个搜索框里按下键盘，键盘按钮所对应的快捷键文本就会被输入到搜索框中。所以有了这个按钮，即便搜索再复杂的快捷键，也不怕麻烦了。（第二个按钮则是对结果进行排序，老版本也有。）



键盘录制

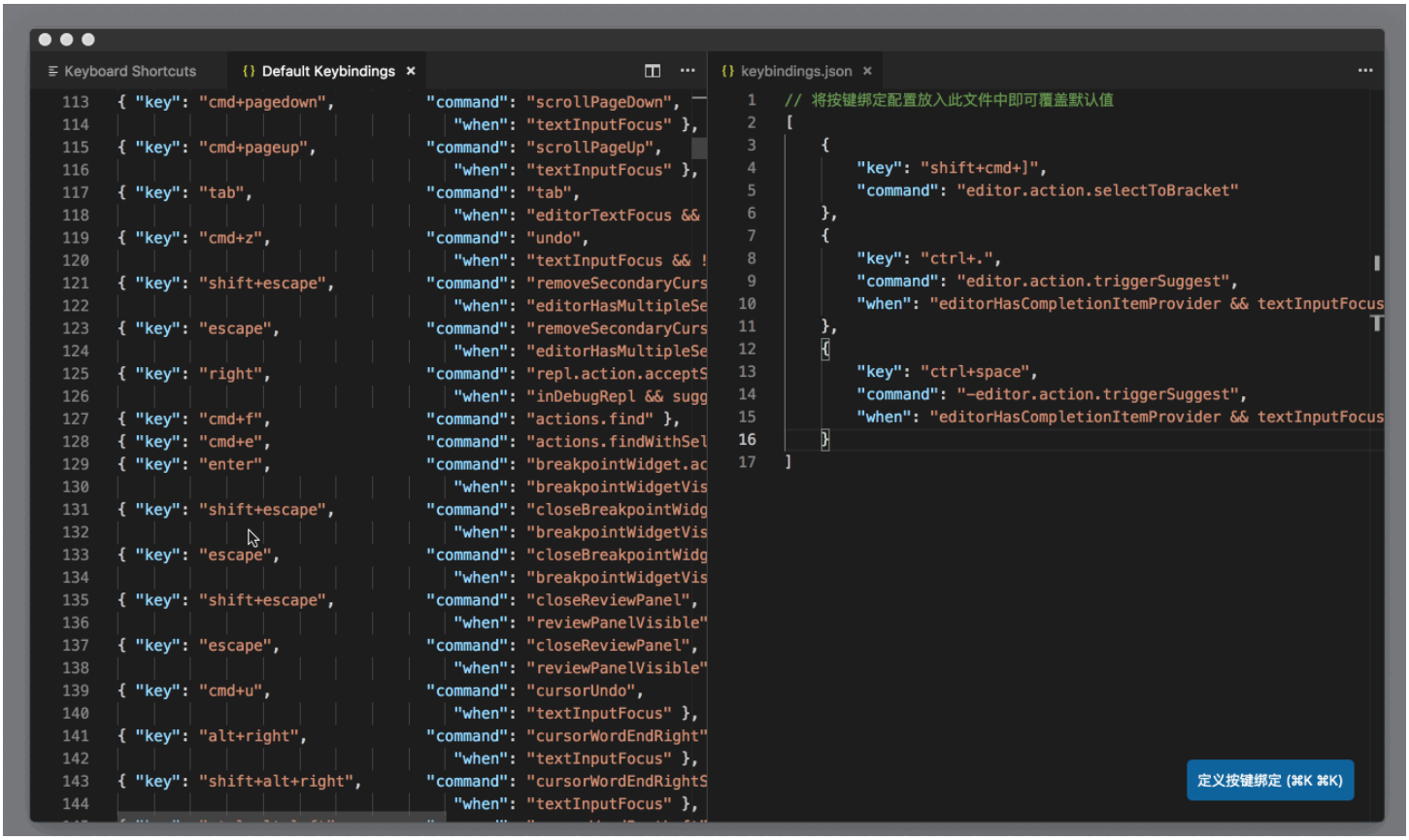
文本界面

虽然我在专栏里一直都是介绍使用图形化的快捷键修改界面，但是 VS Code 快捷键修改，在最开始的时候，跟编辑器设置界面是一样的，也是基于文本编辑器的。你可以在快捷键界面的搜索框下，看到一行提示“高级自定义请打开和编辑器 keybindings.json”。你不妨点击这个链接，效果如下图：



快捷键修改图形界面

如果你要**搜索**某个命令或者快捷键，可以在左侧编辑器唤出搜索框；而**添加**快捷键，则是使用右下角的“定义按键绑定”按钮。



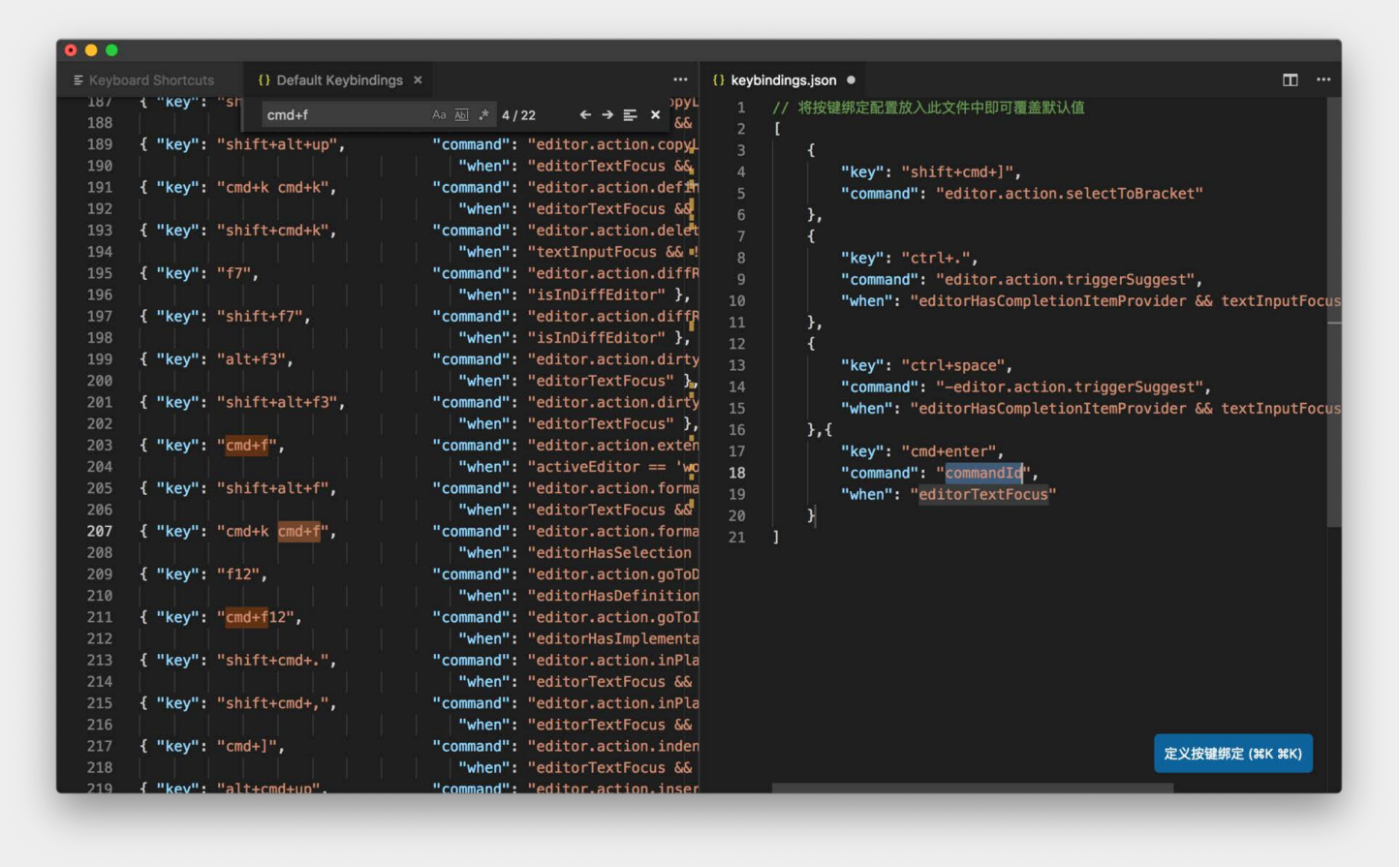
搜索和定义快捷键绑定

你是不是会觉得，虽然我们更好地理解了快捷键设置都是绑定在 keybindings.json 这个文件里的，格式也是 JSON，但是好像这个文本编辑器完全比不上图形化的界面啊？

别着急，接下来咱们就要聊一聊今天文章里技术含量最高的知识点了。

快捷键高级定制

如果你按照上面的介绍，点击了 “定义按键绑定” 的蓝色按钮的话，你会看到右侧的文本编辑器中，插入了一段文本。



快捷键文本编辑器

如果你是跟着专栏内容一章一章走过来的话，你一定立刻就看出来了，这段插入的文本，是一个代码片段（code snippet，不熟悉的话，可以复习专栏第 10 讲 “拒绝重复，你的代码百宝箱：如何书写code snippet？”）。

```
{
  "key": "cmd+enter",
  "command": "command",
  "when": "editorTextFocus"
}
```

Command

这段代码片段，有两个 Tab Stop。第一个 Tab Stop 停留在 "command" 这个属性的值中，它的意思是，我们想要为哪个命令指定特殊的快捷键。这个 command 在左侧的编辑器中也可以查询得到。

When

而第二个 Tab Stop，则是在 "when" 这个属性的值中。这个 "when" 是什么意思呢？顾名思义，它说的是在什么情况下这个快捷键绑定能够生效。此时 "when" 的值已经有一个占位符 （placeholder）了，叫做"**editorTextFocus**"，它代表着光标聚焦在代码编辑器的文本上。如果光标在编辑器的文本上时，那么 "editorTextFocus" 就是 true，那么这个"when"的条件就生效了，这则快捷键绑定就会生效。而假如光标处在集成终端里，此时 "editorTextFocus" 就是 false 了，这个 "when" 就不生效了，同样也就不会绑定这个快捷键了。

在 "when" 条件里，除了editorTextFocus 外，我们还有非常多的值可以使用，并且加以组合。比如集成终端对应的是 terminalFocus，资源管理器对应的是 ÞlesExplorerFocus 。除此之外，你也可以利用VS Code的文档去查询全部可以使用的值。

而在书写 "when" 条件时，VS Code 还支持几个基础的操作符。这样我们就能够书写相对复杂的条件语句了。

1. ! 取反。比如我们希望当光标不在编辑器里时，绑定一个快捷键，那么我们可以使用 !editorFocus，使用 ！ 进行取反。
2. == 等于。when 条件值除了是 boolean 以外，也可以是字符串。比如 resourceExtname 对应的是打开的文件的后缀名，如果我们想给 js 文件绑定一个快捷键，我们可以用 "resourceExtname == .js"。
3. && And 操作符。我们可以将多个条件值组合使用，比如我希望当光标在编辑器里且编辑器里正在编辑的是 js 文件，那么我可以用 "editorFocus && resourceExtname == .js" 。
4. =~ 正则表达式。还是使用上面的例子，如果我要检测文件后缀是不是 js，我也可以写成 "resourceExtname =~ /js/"，通过正则表达式来进行判断。

你可以试着从简单的&&和==等操作符开始，然后再使用正则表达式，进行更复杂的条件判断。

Key

我们回过头来再看下这段快捷键绑定的代码：

```
{
  "key": "cmd+enter",
  "command": "command",
```

```
  "when": "editorTextFocus"
}
```

在这个 JSON 对象里第一个键是 key，也就是你将要使用的快捷键。如果你是使用“定义按键绑定”按钮来生成的，那么 VS Code 会根据你的键盘布局来自动生成这个文本。你当然也可以自行修改，不过我不建议这么做，因为 VS Code 为了适应各种不同的键盘布局，在 key 这个值上还是有很多特殊要求的。

比如说，你想给某个快捷键绑定上“cmd+\”这个键，VS Code 会自动为你输入 “cmd+[Backslash]”。因为不同键盘上的 “\” 键的布局可能会不同，VS Code 使用统一的 [Backslash]来进行指代。你当然也可以输入 “cmd+\” （请注意，这里我使用了两个 \，因为我们在 JSON 里需要进行转义 escape），但是如果你把 keybindings.json 分享给其他人的话，是不一定能够保证生效的。

如果你对如何手动输入 key 很感兴趣，可以参考VS Code 中相关的文档，该文档里介绍了 VS Code 针对不同键盘布局 （keyboard layout）所做的特殊处理。

删除快捷键

我们在快捷键的图形界面里也介绍过，可以使用上下文菜单删除某个快捷键绑定。不过，删除操作在 keybindings.json 里是如何体现的呢？其实非常简单，假如说我们不想使用 “cmd+s” 来保存文本，首先我们找到 “cmd+s” 对应的快捷键绑定设置，

```
{
  "key": "cmd+s",
  "command": "workbench.action.files.save"
}
```

然后我们只需在右侧的 keybinding.json 里，添加一条新的快捷键绑定，如下：

```
{
  "key": "cmd+s",
  "command": "-workbench.action.files.save"
}
```

在这条新的快捷键绑定里，“command” 的值在开头添加了一个 - 减号。这个减号就告诉 VS Code 我们希望给这个命令解除快捷键绑定。

对了，在解绑快捷键时，我们也可以使用 “when” 条件。这样的话，我们就能够做到只在特定情况下解

除某个快捷键的绑定。

小结

以上就是我们今天的全部内容了。我们介绍了 VS Code 的设置和快捷键绑定界面的演变过程，相信你已经非常熟悉 VS Code 是如何保存和处理这些配置文件的了。同时，我们还介绍了如何对快捷键绑定做一些相对复杂的定制，在后面的文章里，我还会介绍如何将这些快捷键绑定以插件的形式更好地分享给其他人。

在构思这篇文章的时候，我还在想是不是要把工作区这个部分我没有涉及到的工作区设置或者快捷键全部介绍一遍。但是授人鱼不如授人以渔，相信你了解了如何搜索、修改和定制 VS Code 的设置和快捷键后，你能够自己寻找到想要的答案。

精选留言