18讲为你的项目打造Workflow（上）

在上一讲中，我们一起学习了如何在编辑器中使用集成终端。有了集成终端，我们在运行构建、测试脚本，甚至是产品发布的时候，就可以不用离开编辑器了。但是，我们有没有办法让这些工作再快一点？或者有没有办法能够更方便地得到这些任务运行的反馈？能不能够将这些任务的执行跟我们的日常代码操作结合到一起？

好了，我就不卖关子了，今天我们就一起来了解一下 VS Code 的任务系统。

执行任务

任务系统的目的，是将各种形形色色的任务脚本尽可能地统一化，然后提供一套简单但又定制化强的方式操作它们。

这里举个我自己的例子。比如，我在使用 git 进行版本管理，当我在终端里要创建一个的新的分支时，得输入 “git checkout -b branchName”，但是要每次都打全这个命令太麻烦了，于是我自己创建了一个 bash 的别名（alias）gco 用来替代 “git checkout -b”。而 VS Code 的版本管理更近了一步，它在状态栏上添加了一个按钮，我只需点击一下状态栏，就可以创建分支了。

任务系统也希望提供一样的效果，比如把我们日常使用的脚本命令，通过命令面板或者快捷键迅速执行，并且还可以将这一套快捷操作的方式，分享给工作在同一个项目上的同事。

任务自动检测

VS Code 的任务系统的第一大功能，就是对任务的自动检测。下面我们一起来看下这个功能。

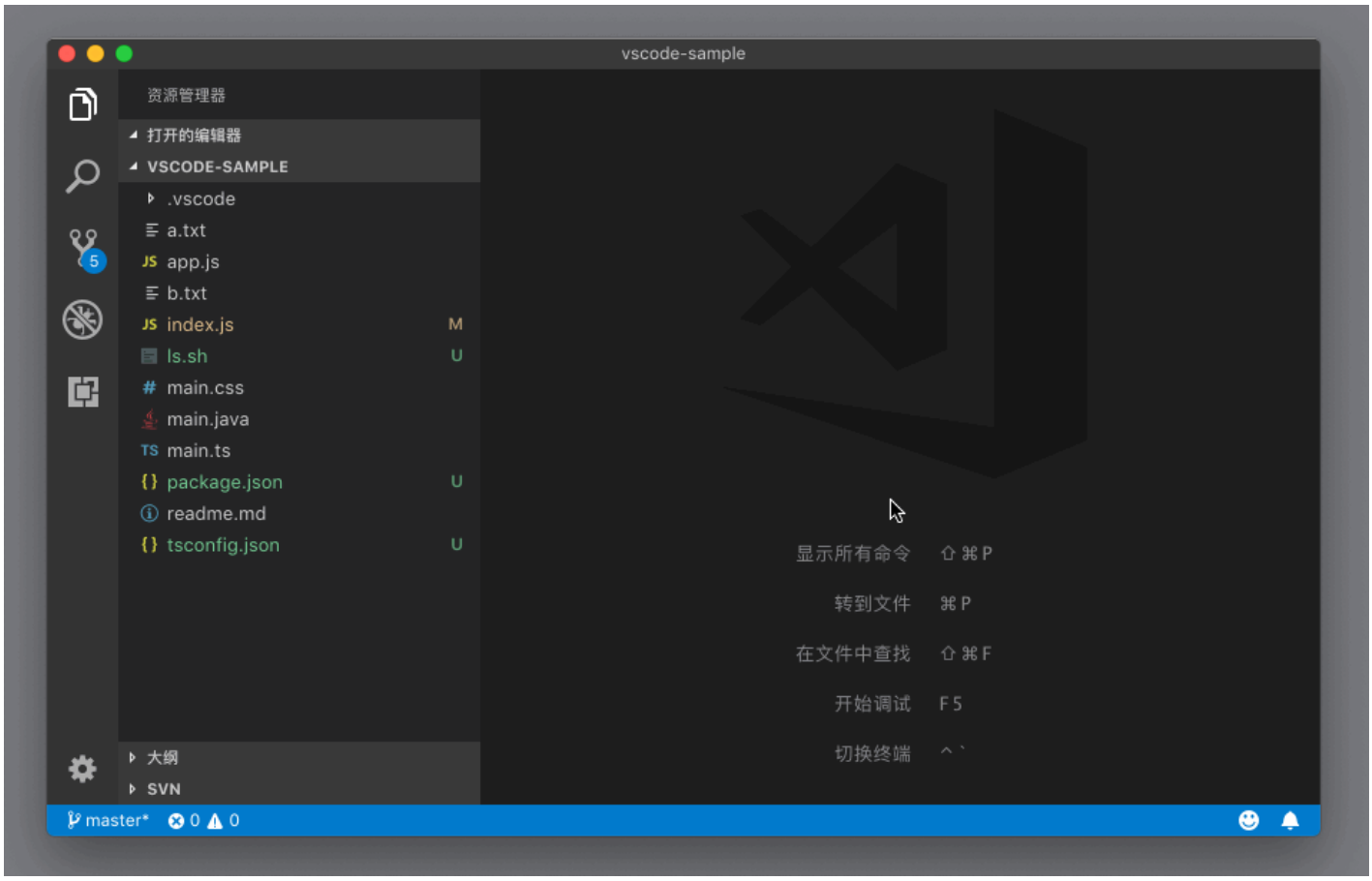
如果你的项目或者文件夹里有 typescript、grunt、jake、gulp、npm 这几个脚本工具的配置文件的话，VS Code 会自动读取当前文件夹下的配置。举个例子，我们打开一个文件夹，这个文件夹下有个 package.json 的文件，它是 npm 的配置文件，代码内容如下：

```
{
 "name": "sample",
 "scripts": {
  "test": ""
 }
}
```

然后当我们打开命令面板，搜索 “运行任务”（Run Task）时，紧接着我们就能看到两条跟npm相关的任务：

- npm install
- npm test

第一条npm install是 npm 用于安装包的命令；第二条 npm test，则是我们在 package.json 里指定的脚本名称，但是 VS Code 也同样检测出来了。此时我们选择 npm install 这条命令的话，VS Code 就会打开一个集成终端，然后运行这条脚本。



自动检测任务并且运行

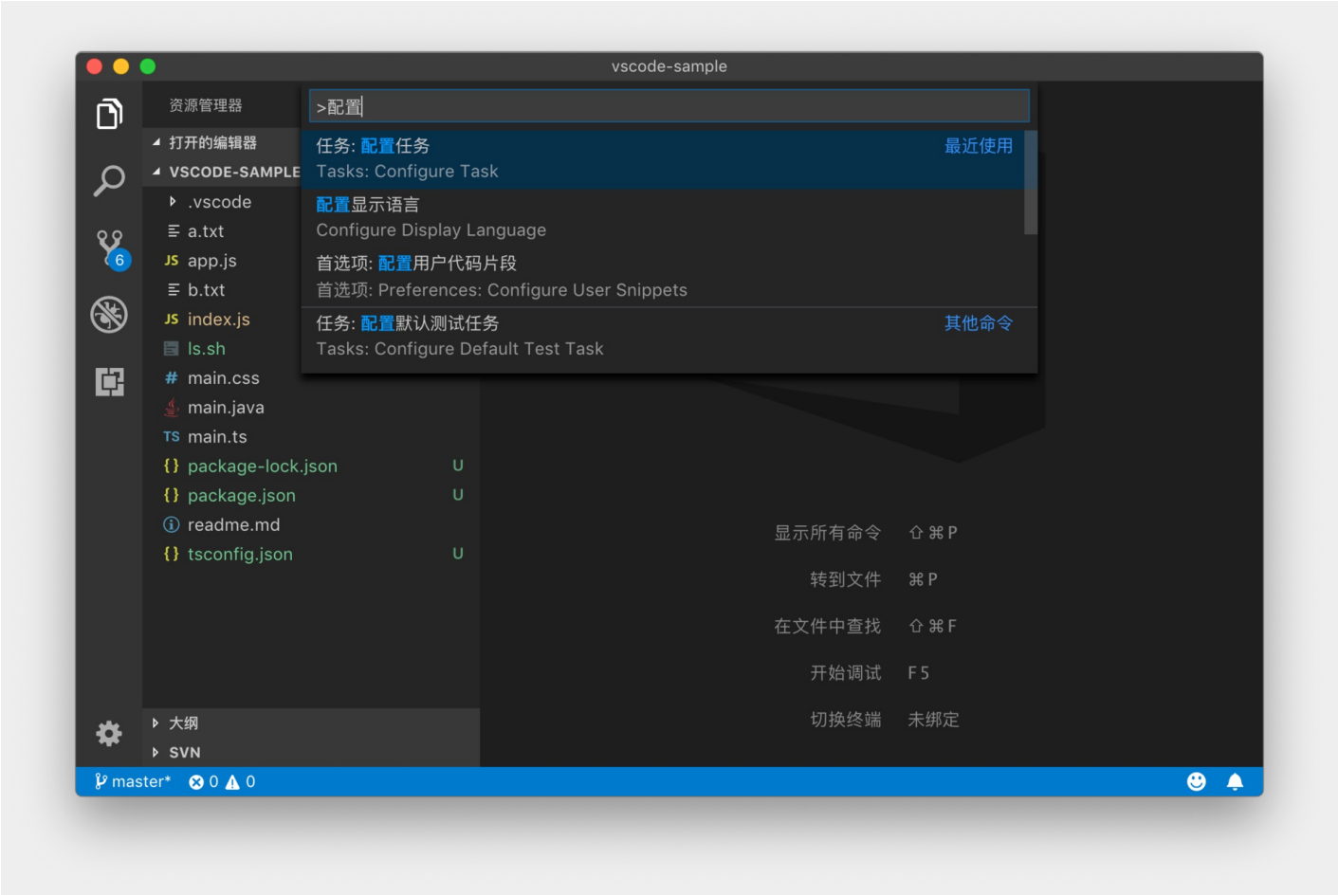
在上面的动图中，除了 npm 的两条任务，我们还能看到 typescript 关于编译的两条任务，这是因为当前文件夹下有 tsconfig 文件，VS Code 觉得这是一个 typescript 项目，所以进行了自动任务检测。

上面我们提到的 npm 等五个脚本工具相关的任务，VS Code 是会自动检测的，同时 VS Code 还开放了类似的 API 给插件，允许插件来实现一样的功能。比如说我曾经在 Ruby 插件里也实现了 Rake 的自动检测，所以当你安装 Ruby 插件后，打开一个使用了 Rake 的 Ruby 项目，从命令面板里执行 “运行任务”，你同样可以看到所有的 Rake 任务。如果你使用的某个脚本工具相关的插件还不支持这个功能，可以去提 issue 建议插件加上。

自动检测任务，还只是小小的第一步，因为它只是从我们已经写好的脚本里把任务读取出来。下面我们看看，如何来自己配置任务。

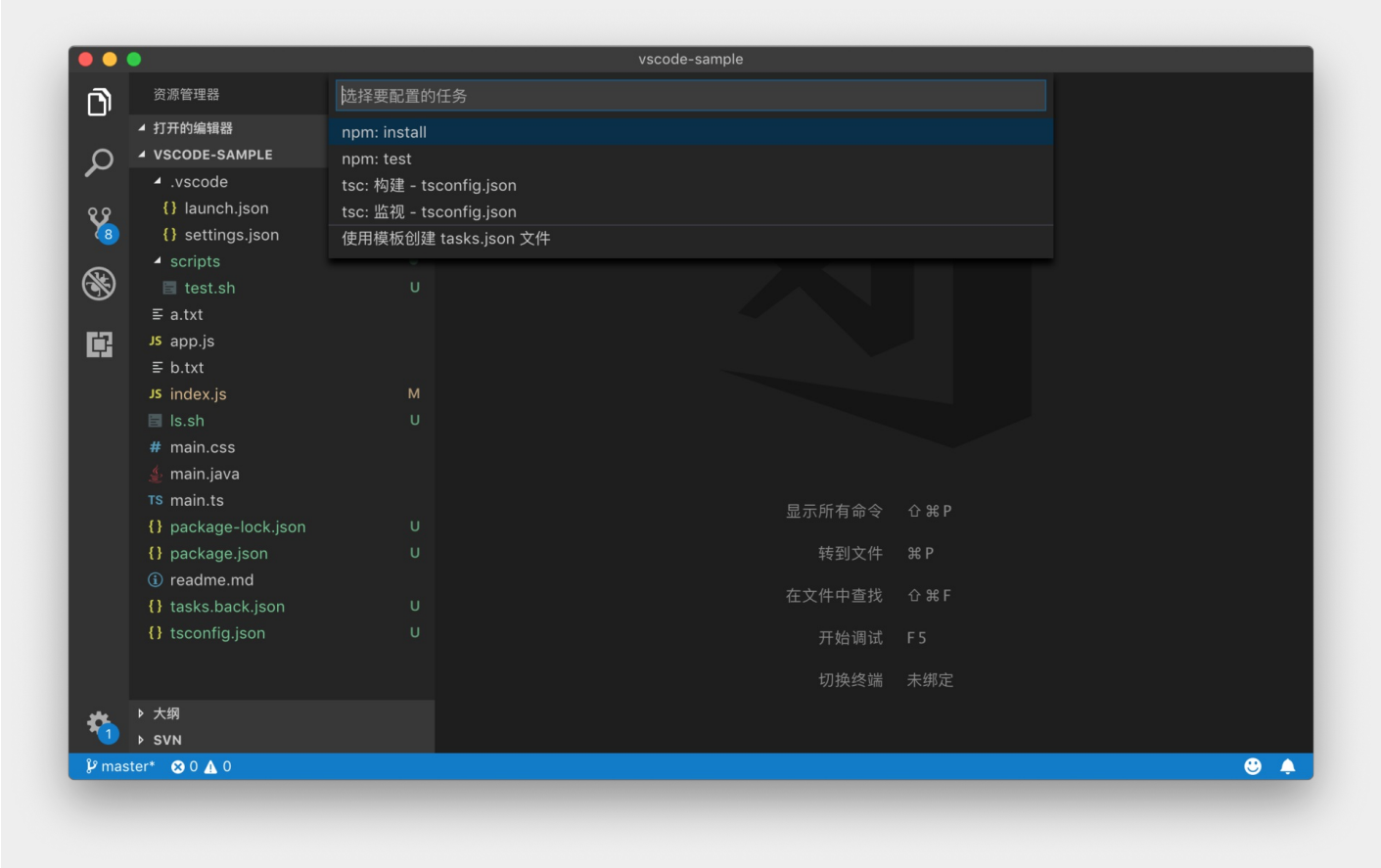
自定义任务

首先我们在命令面板里，搜索 “配置任务”（Configure Task）并执行。



配置任务

我们能够看到一个下拉框，这里面提供了多个不同的选项。



选择要配置的任务

如果我们选择第一个，也就是`npm: install`这一项的话，VS Code 会立刻在`.vscode`文件夹下创建一个 tasks.json 文件，它的格式是 JSON，可读性很好且易于修改。

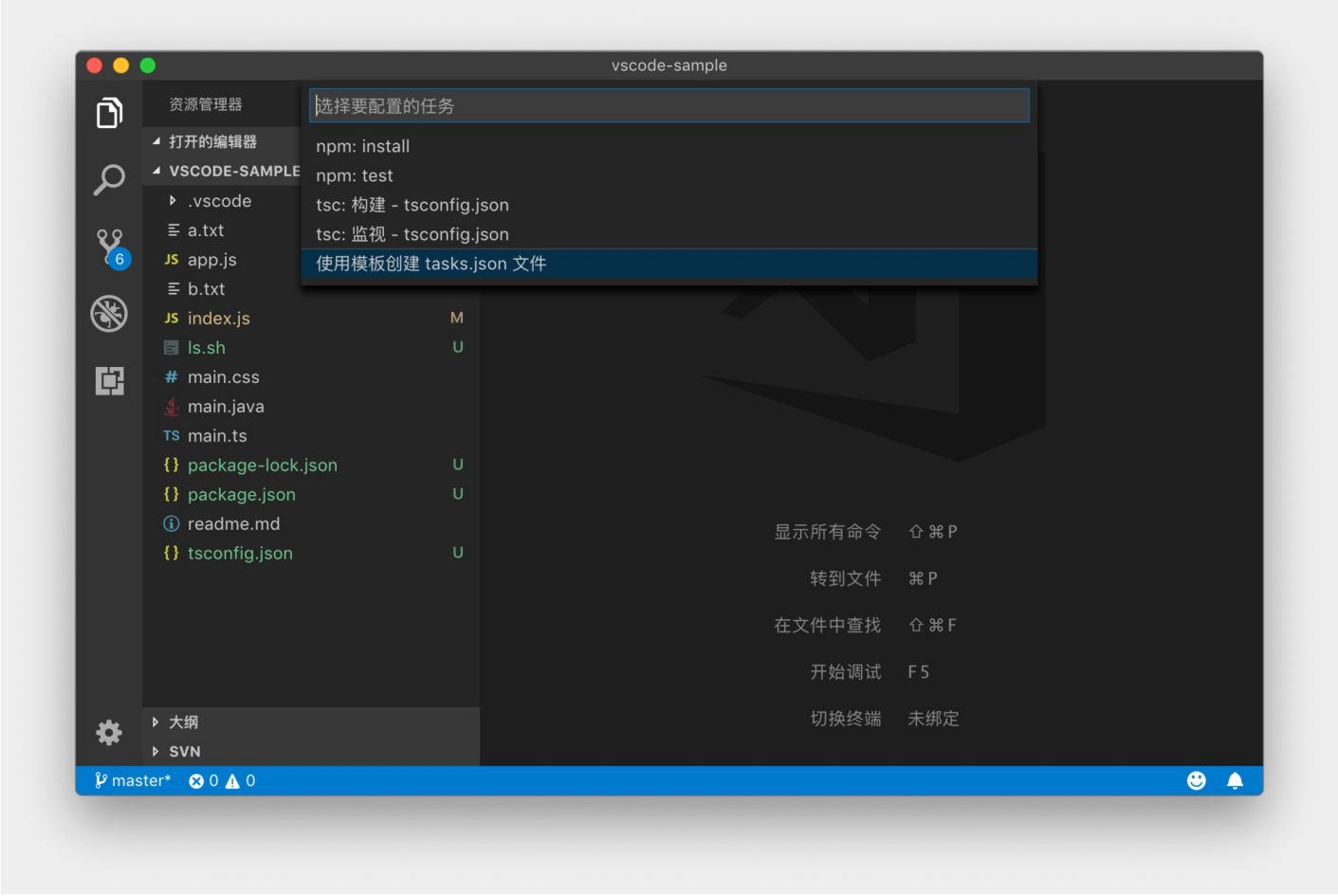
```
{
 // See https://go.microsoft.com/fwlink/?LinkId=733558
 // for the documentation about the tasks.json format
 "version": "2.0.0",
 "tasks": [
  {
   "type": "npm",
   "script": "install",
   "problemMatcher": []
  }
 ]
}
```

我们能够看到，这个文件里有一个属性叫做tasks，它的值是一个数组，这就是我们可以在当前文件夹下使用的所有任务了。现在这个模板里，只有一个任务 npm，它有三个属性。

第一个属性是 type，它代表着你要使用哪个脚本工具，我们这里使用是 npm。第二个是 script 脚本，则是我们想要 npm 工具执行的某个脚本。第三个属性 problemMatcher，这个我放在后面的内容里介绍，暂时看不懂也没关系，稍安勿躁。

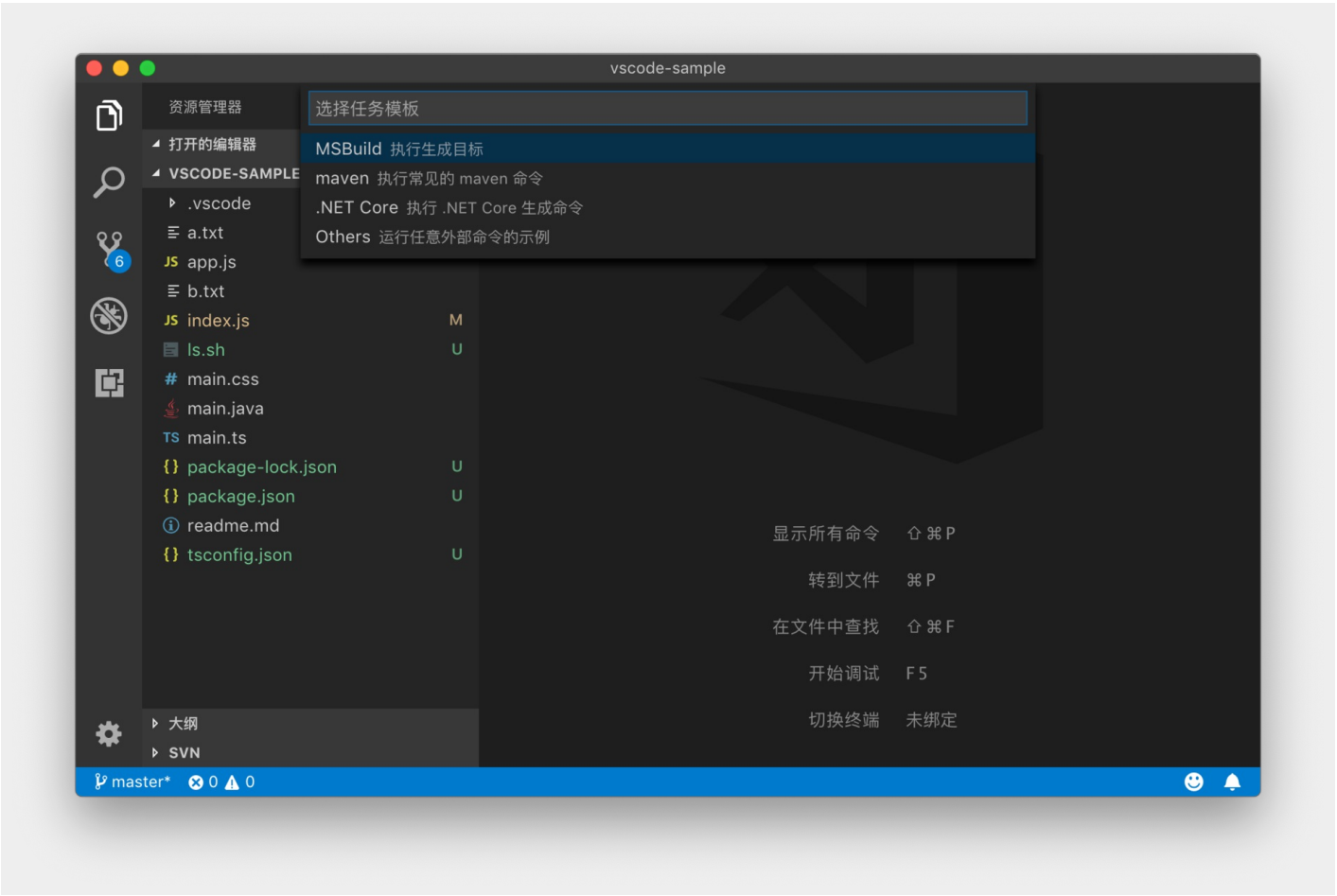
可以看得出来，这个任务相当于上面自动检测的任务系统的一个映射。我们把 npm 脚本自己的配置文件，转变成了 VS Code 任务系统的配置文件，也就是tasks.json。

但是这种类型的任务，受限于 VS Code 或者插件所支持的脚本工具，缺乏一定的灵活性。我们把 `.vscode/tasks.json` 文件先删除，然后重新打开命令面板，执行 “配置任务”（Configure Task）。不过这一次，我们选择最后一项，使用模板创建 tasks.json 文件。（请注意，这里我们是为了从 0 了解任务系统，所以才把之前的 tasks.json 文件删除，如果你已经在项目中使用 tasks.json ，大可不必这么做，照着文章读下去就可以了。）



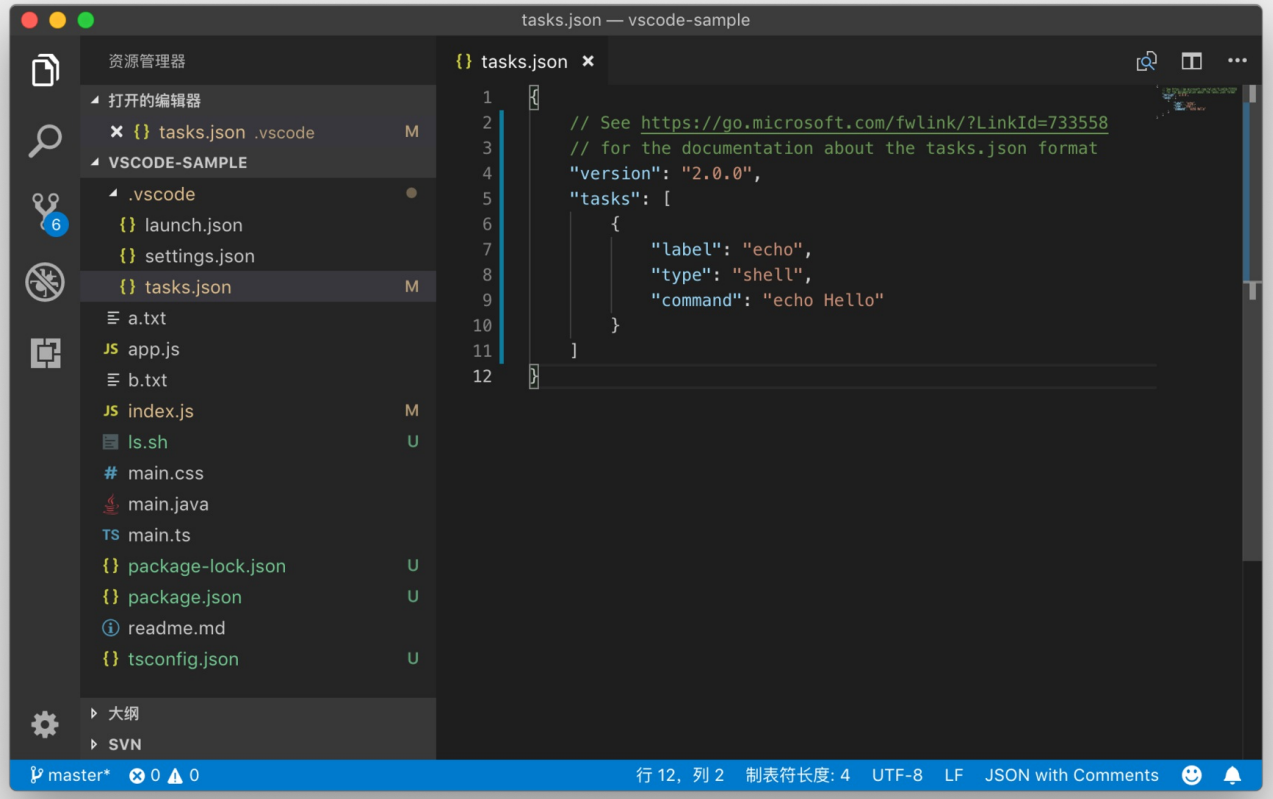
选择使用模板来创建任务

紧接着 VS Code 就问我们了，希望使用哪种模板。这里模板的多少，同样取决于你装了哪些插件。默认情况下，VS Code 为 MSBuild、Maven、.NET Core 提供了模板，而最后一个 Others，则是一个通用的模板，我们一起来看下它。



选择任务模板

选择完 Others 之后，VS Code 在当前文件夹根目录下的 .vscode 文件夹中，创建了 tasks.json 文件。



自定义任务模板

这个文件的内容如下：

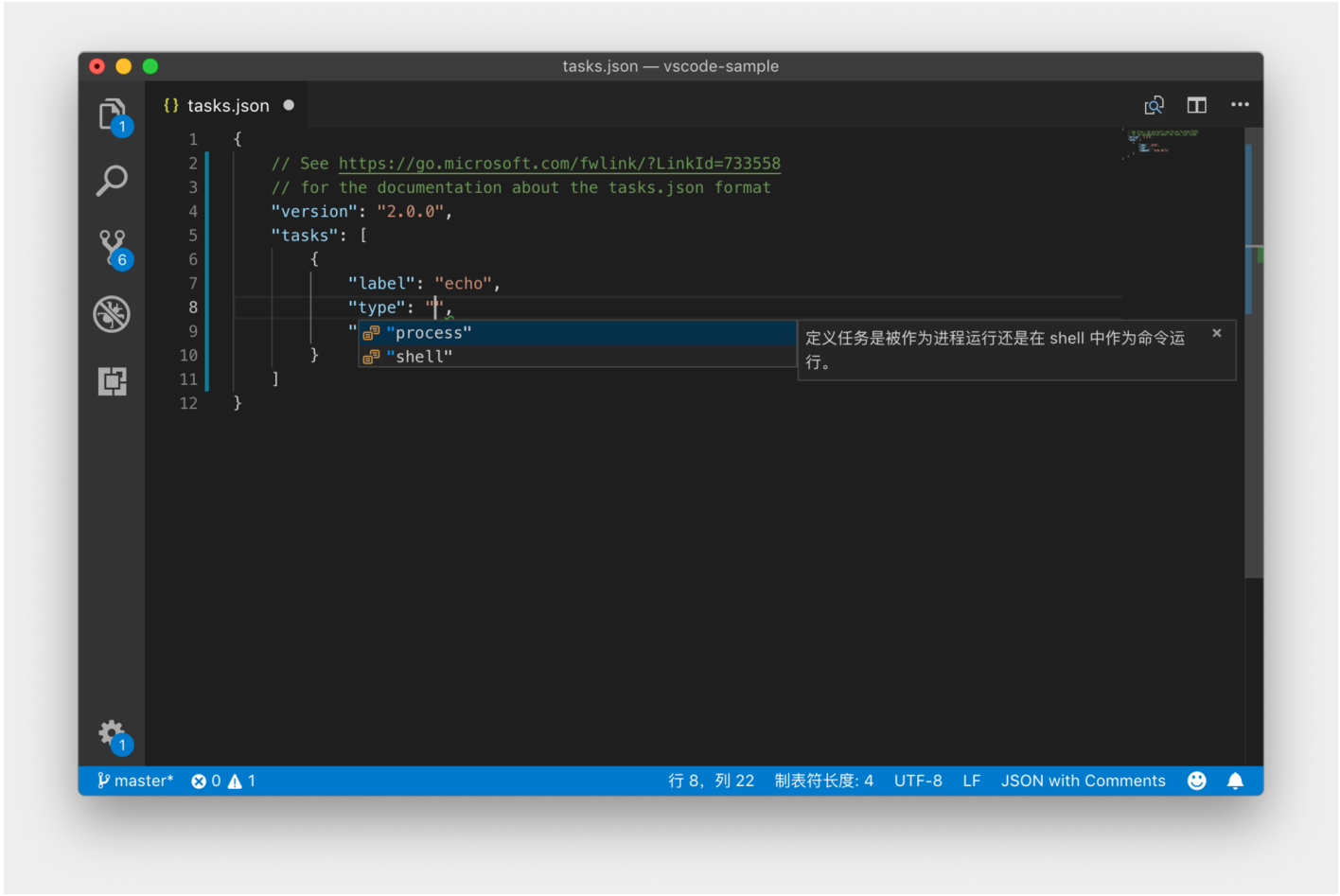
```
{
 // See https://go.microsoft.com/fwlink/?LinkId=733558
 // for the documentation about the tasks.json format
 "version": "2.0.0",
 "tasks": [
  {
   "label": "echo",
   "type": "shell",
   "command": "echo Hello"
  }
 ]
}
```

这个文件跟最开始我们看到的非常接近，tasks 属性下有一个任务，只不过它的三个属性跟之前很不一样。

第一个属性是 label 标签，就是这个任务的名字。我们在命令面板里执行任务会需要读到它，所以它的值应该尽可能地描述这个任务是干什么的。

第二个属性是 type 类型。对于**自定义的任务**来说，这个类型可以有两种选择，一种是这个任务被当作进程来运行，另一种则

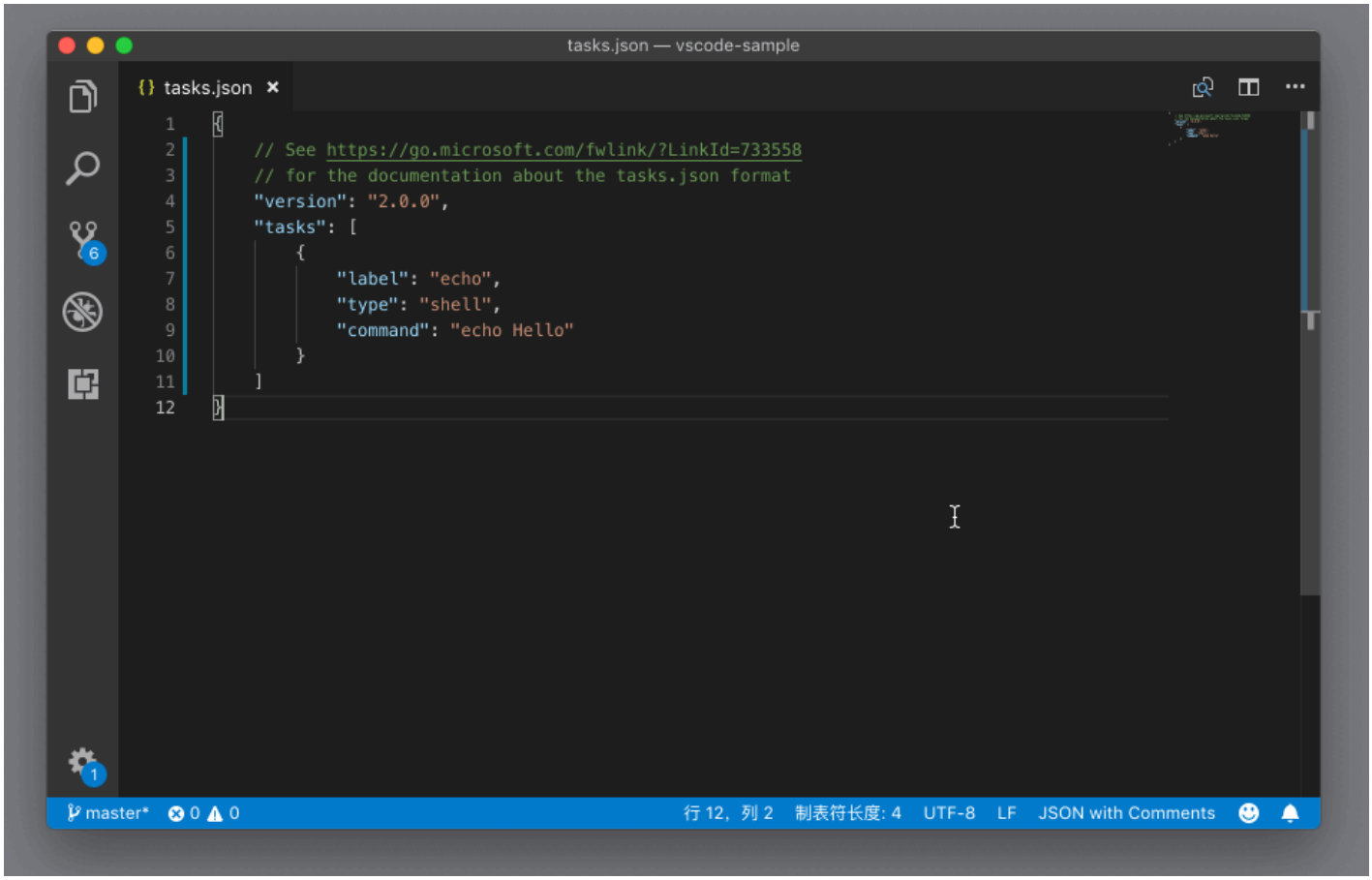
是在 shell 中作为命令行来运行。默认情况下我们会在 shell 下运行，而且这个 shell 命令行将会在集成终端里执行，shell 的选择则会尊重我们在集成终端的配置。比如在我的例子中，我是在 macOS 下运行的，系统默认的 shell 是 zsh，那么当我运行这个脚本时，就会在 zsh 里执行。



自定义任务类型

第三个属性是命令command，它代表着我们希望在 shell 中运行哪一个命令，或者我们希望运行哪个程序。

好了，看完这三个属性，你一定希望第一时间试试看这个任务。下面我们要做的就是打开命令面板，搜索“运行任务”，选择“echo”这个任务（这个就是我们在label里写的名字），按下回车后，VS Code 会问我们 “选择根据何种错误和警告扫描任务输出”，这个问题涉及到任务系统的另一个重要功能，我会在后面介绍，现在就选择第一个选项 “继续而不扫描任务输出” 好了。



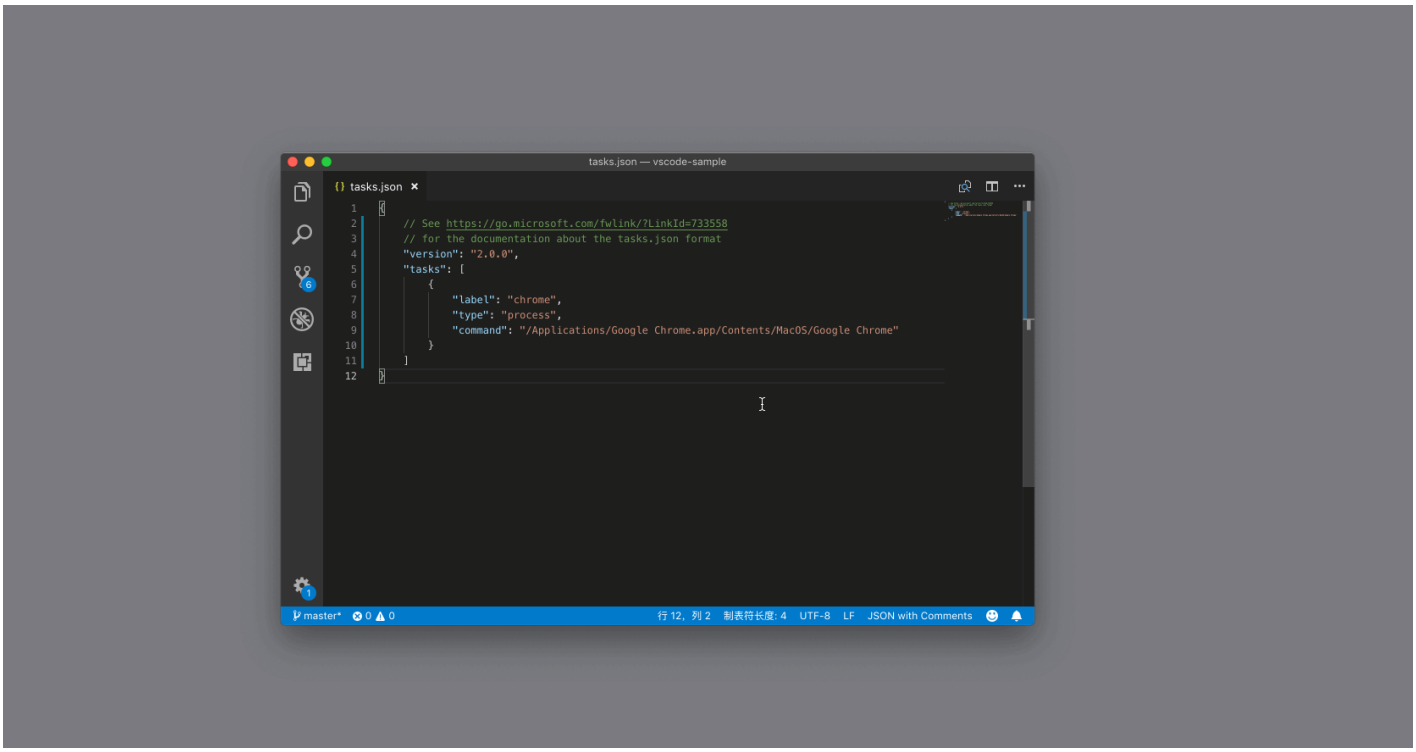
执行自定义的任务

到这里你可能会吐槽，为了完成一个任务，搞得好复杂啊。别着急，等我们把各个功能都介绍完毕，就能够选择快速的方式运行了。

刚才上面我们提到了 “type” 类型，还支持 “process”， 也就是以进程的形式运行。如果我们选择这个类型，那么就需要在 “command” 里提供程序的地址。比如下面的例子里，我提供了 Chrome 浏览器在 macOS 下的地址。

```
{
 "version": "2.0.0",
 "tasks": [
  {
   "label": "chrome",
   "type": "process",
   "command": "/Applications/Google Chrome.app/Contents/MacOS/Google Chrome"
  }
 ]
}
```

我们运行看看它的效果：



使用任务打开 Chrome

我们成功地执行这个任务，打开了浏览器。如果我们把这个任务分享给同事，而他们使用的系统是 Windows 或者 Linux，那么这个任务就没法使用了，因为Chrome 的地址完全对不上号。不过我们可以为 Windows 或者 Linux 系统指定特定的地址，书写的方式如下（请注意，在你的操作系统上，Chrome 的地址可能不完全跟下面的代码样例一样）：

```
{
 "version": "2.0.0",
 "tasks": [
  {
   "label": "chrome",
   "type": "process",
   "command": "/Applications/Google Chrome.app/Contents/MacOS/Google Chrome",
   "windows": {
    "command": "C:\\Program Files (x86)\\Google\\Chrome\\Application\\chrome.exe"
      },
      "linux": {
          "command": "/usr/bin/google-chrome"
      }
  }
 ]
}
```

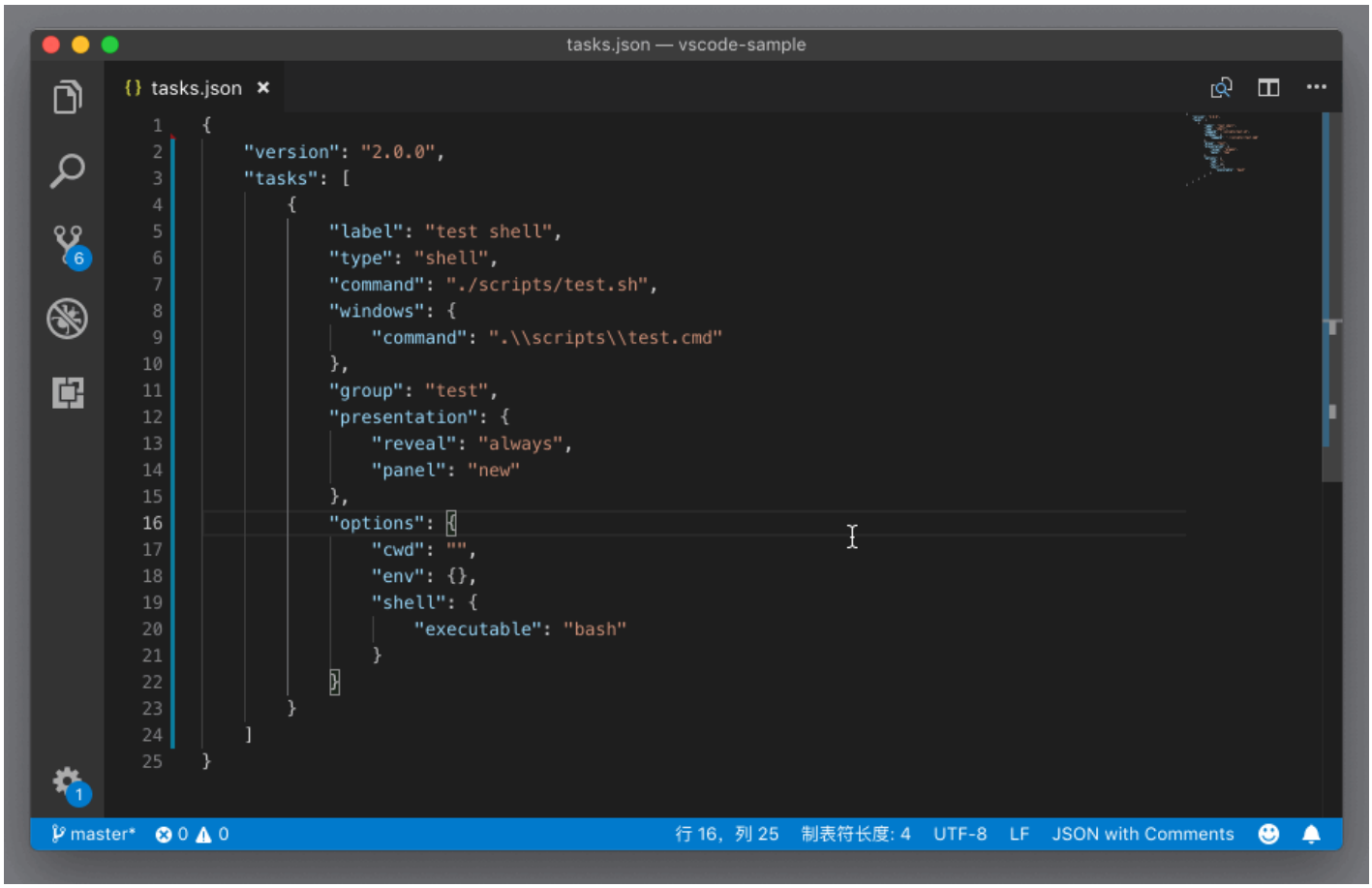
分组和结果显示

下面我们再看一个更复杂一些的任务，来学习一下任务系统配置里的其他属性。在下面的任务里，我们能够看到“label”“type”“command” 这几个熟悉的属性，它们的意思是，在 shell 下运行 “./scripts/test.sh” 这个脚本。不过又多了三个属

性 “group” “presentation” 和 “options”，它们分别是干什么的呢?

```
{
 "version": "2.0.0",
 "tasks": [
  {
   "label": "test shell",
   "type": "shell",
   "command": "./scripts/test.sh",
   "windows": {
    "command": ".\\scripts\\test.cmd"
   },
   "group": "test",
   "presentation": {
    "reveal": "always",
    "panel": "new"
   },
   "options": {
    "cwd": "",
    "env": {},
    "shell": {
     "executable": "bash"
    }
   }
  }
 ]
}
```

“group” 属性就是分组，我们可以通过这个属性指定这个任务被包含在哪一种分组当中。关于分组，我们有三种选择：“build” 编译生成、“test”测试和 “none”。在这个例子里，我们把它设置为了 “test”。那么，当我们在命令面板里搜索 “运行测试任务” (Run Test Task) 时，只有这个任务会被显示出来。



任务分组：构建、测试

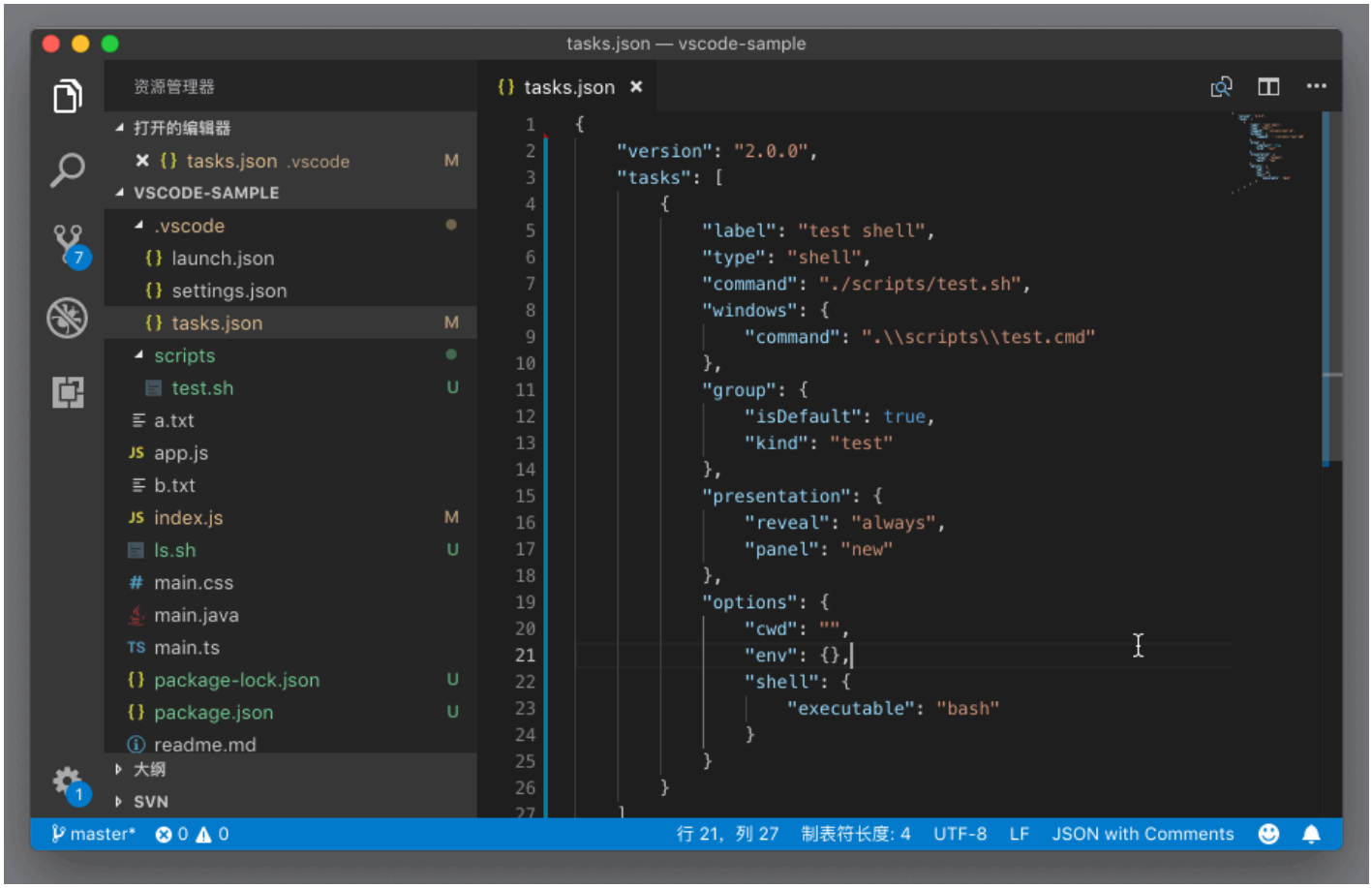
如果我们把这个分组 group 改为 “build”，那么在我们执行 “运行生成任务” （Run Build Task）时，则同样能够看到它。

分组的意思很好理解，但是你可能感觉还是不够意思，因为虽然有专门的命令去执行生成任务，或者测试任务，但是它们还是调出了一个列表让我们进行选择，多此一举，有没有办法一键运行?

当然没问题，我们只需将分组 “group” 的值改成下面这样即可。“isDefault” 代表着这条任务是不是这个分组中的默认任务，“kind” 则是代表分组。

```
"group": {
    "isDefault": true,
    "kind": "test"
   },
```

当把“group”改成以上的值后，再当我们执行 “运行测试任务” (Run Test Task) 命令时，我们会发现这条测试任务被直接执行了。



自动运行默认测试任务

而 “运行生成任务” 就更方便了，这个命令已经绑定了一组快捷键。我们只需按下 Cmd + Shift + B （Windows 上是 Ctrl + Shift + B）就可以自动运行默认的那个生成任务了（build task）。

接下里的两个属性：“presentation” 是用于控制任务运行的时候，是否要自动调出运行的界面，让我们看到结果，或者是否要新创建一个窗口执行任务；而 “options” 则是用于控制任务执行时候的几个配置，比如控制任务脚本运行的文件夹地址 “cwd”，控制环境变量 “env”，或者控制任务脚本运行的时候使用哪个 shell 环境。

你可以看到，在上面的例子里，我把 shell 环境指定为了 bash，那么这个脚本运行的时候，虽然还是使用的集成终端，但是它会使用 bash 而不是 zsh 来运行这个脚本。

小结

任务系统的知识体系相对复杂，一篇文章难以完全覆盖，所以我将其分为上下两讲来介绍，以争取对任务系统有个更全面的认知。

以上就是任务系统的上半部分知识，我们介绍了如何使用自动任务检测，如何持久化任务，如何创建自定义的任务等等。在我们进入到下一讲之前，希望你能够熟悉今天所讲的内容，这样学习下一讲的知识时，就不会云里雾里了。

精选留言

Trust me

能不能绑定UI按钮之类的

配置能不能存储到用户级别

能不能到dir根路径

2018-10-23 22:58

阿弥陀佛么么哒

配置 group 的 isDefault 的属性还是检测不到默认的呢，也没有修改过快捷键

2018-11-08 12:37

那句诺言

希望能多给点例子和参考资料，只是看还是有点不太清楚

2018-10-23 09:23