TOPICS Environments ▼

在VS Code中使用Python环境

(https://github.com/Microsoft/vscode-docs/blob/master/docs/python/environments.md)

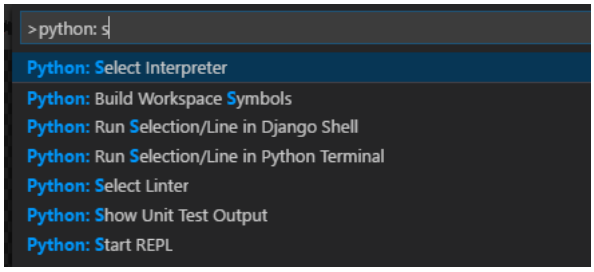
Python中的"环境"是Python程序在其中运行的上下文。环境由解释器和任意数量的已安装软件包组成。VS Code的Python扩展提供了有用的集成功能，可用于不同的环境。

> **注意**：如果要在Visual Studio Code中使用Python入门 (/docs/python/python-tutorial)，请参阅教程VS Code中的Python入门 (/docs/python/python-tutorial)。

选择并激活环境

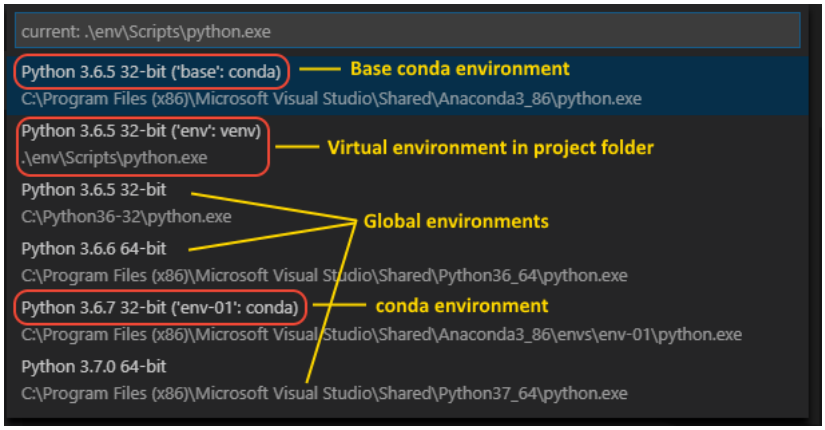
默认情况下，Python扩展会查找并使用它在系统路径中找到的第一个Python解释器。如果找不到口译员，则会发出警告。在macOS上，如果您使用的是操作系统安装的Python解释器，则扩展也会发出警告，因为您通常希望使用直接安装的解释器。无论哪种情况，都可以通过在用户设置 (/docs/getstarted/settings)中将设置 `python.disableInstallationCheck` 为来禁用这些警告。 `true` (/docs/getstarted/settings)

要选择特定的环境，请使用**Python：从命令面板中选择"解释器"**命令（ Ctrl + Shift + P ）。



您可以随时切换环境；交换环境可帮助您根据需要使用不同的解释器或库版本来测试项目的不同部分。

" **Python：选择解释器**"命令显示可用的全局环境，conda环境和虚拟环境的列表。（有关详细信息，请参见" 在扩展程序的环境中查找"部分，包括这些类型的环境之间的区别。）例如，下图显示了多个Anaconda和CPython安装以及位于其中的conda环境和虚拟环境（ `env` ）。工作区文件夹：



> **注意**：在Windows上，VS Code可能需要一些时间来检测可用的conda环境。在此过程中，您可能会在环境路径之前看到"（已缓存）"。标签表明VS Code当前正在使用该环境的缓存信息。

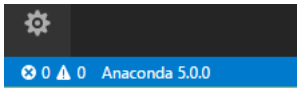
从列表中选择一个解释器，将 `python.pythonPath` 在Workspace Settings (/docs/getstarted/settings)中添加一个带有解释器路径的条目。因为该路径是工作空间设置的一部分，所以每次打开该工作空间时都应该已经选择了相同的环境。如果您想为应用程序设置默认解释器，则可以 `python.pythonPath` 在"用户设置"中手动添加一个条目。为此，请打开"命令面板"（ Ctrl + Shift + P ）并输入"**首选项**"：**打开"用户设置"**。然后 `python.pythonPath` 使用适当的解释器在用户设置的Python扩展部分中设置。

Python扩展程序使用选定的环境来运行Python代码（使用**Python：在Terminal命令中运行Python文件**），并 `.py` 在以下位置打开文件时提供语言服务（自动完成，语法检查，整理，格式化等）。编辑器，然后使用" **终端：创建新的集成终端**"命令打开终端。在后一种情况下，VS Code会自动激活所选的环境。

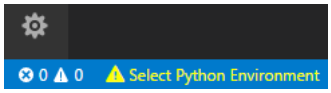
> **提示**：为防止自动激活选定的环境，请添加 `"python.terminal.activateEnvironment": false` 到 `settings.json` 文件中（可以将其放置在现有设置的兄弟位置中的任何位置）。

> **注意**：默认情况下，VS Code `python:pythonPath` 在调试代码时使用通过设置标识的解释器。您可以通过在 `pythonPath` 调试配置的属性中指定其他路径来覆盖此行为。请参阅选择调试环境。

状态栏始终显示当前的解释器。



当没有选择口译员时，状态栏也会反映出来。



无论哪种情况，单击状态栏的此区域都是**Python**的便捷快捷方式：**Select Interpreter**命令。

> **提示**：如果VS Code识别虚拟环境有任何问题，请在文档存储库中提出问题 (https://github.com/Microsoft/vscode-docs/issues)，以便我们帮助确定原因。

环境和终端窗口

使用**Python之后：选择Interpreter**，在右键单击文件并选择**Python：在Terminal中运行Python File**时，将应用该解释器。使用" **终端机：创建新的集成终端机**"命令时，环境也会自动激活，除非将 `python.terminal.activateEnvironment` 设置更改为 `false` 。

但是，从激活了某些Python环境的外壳启动VS Code不会在默认的Integrated Terminal中自动激活该环境。在运行VS Code之后，使用**终端：创建新的集成终端**命令。

> **注意：**如果将PowerShell设置为集成外壳，则无法在集成终端中自动激活conda环境。有关如何更改外壳的信息，请参见集成终端-配置 (/docs/editor/integrated-terminal#_configuration)。

您对终端中已激活的环境所做的任何更改都是持久性的。例如， `conda install <package>` 在激活了conda环境的终端上使用该软件包将软件包永久安装到该环境中。类似地， `pip install` 在激活了虚拟环境的终端中使用时，会将软件包添加到该环境中。

使用**Python更改解释器：**"**选择解释器**"命令不会影响已经打开的终端面板。因此，您可以在拆分终端中激活单独的环境：选择第一个解释器，为其创建终端，选择其他解释器，然后使用终端标题栏中的拆分按钮（ Ctrl + Shift + 5 ）。

选择调试环境

默认情况下，该 `python.pythonPath` 设置指定用于调试的Python解释器。但是，如果您 `pythonPath` 在的调试配置中有一个属性， `launch.json` 则改用该解释器。更具体地说，VS Code在确定用于调试的解释器时采用以下优先顺序：

1. `pythonPath` 所选调试配置的属性 `launch.json`
2. `python.pythonPath` 在工作区中设置 `settings.json`
3. `python.pythonPath` 在用户中设置 `settings.json`

有关调试配置的更多详细信息，请参阅调试配置 (/docs/python/debugging)。

扩展程序在哪里寻找环境

该扩展程序会在以下位置自动寻找口译员：

- 标准的安装路径，例如 `/usr/local/bin` ， `/usr/sbin` ， `/sbin` ， `c:\\python27` ， `c:\\python36` ，等。
- 虚拟环境直接位于工作区（项目）文件夹下。
- 位于由 `python.venvPath` 设置标识的文件夹中的虚拟环境（请参阅" 常规设置" (/docs/python/settings-reference#_general-settings)），其中可以包含多个虚拟环境。该扩展程序在的第一级子文件夹中查找虚拟环境 `venvPath` 。
- 位于virtualenvwrapper (https://virtualenvwrapper.readthedocs.io/) `~/.virtualenvs` 文件夹中的虚拟环境。 (https://virtualenvwrapper.readthedocs.io/)
- pyenv (https://github.com/pyenv/pyenv)安装的口译员 (https://github.com/pyenv/pyenv)。
- 位于由标识的路径中的虚拟环境 `WORKON_HOME` （由virtualenvwrapper使用 (https://virtualenvwrapper.readthedocs.io/)）。
- 包含Python解释器的Conda环境。VS Code不会显示不包含解释器的conda环境。
- 安装在口译 `.direnv` 的文件夹direnv (https://direnv.net/)工作区（项目）文件夹下。

如果Visual Studio Code不能自动找到解释器，则也可以手动指定它。

> **注意**：触发"选择解释器"流程后，将搜索工作空间文件夹的pipenv (https://pipenv.readthedocs.io/)环境。如果找到一个，则不会搜索或列出其他解释器，因为pipenv希望能够管理所有方面。

该扩展程序还加载由设置标识的环境变量定义文件 `python.envFile` 。此设置的默认值为 `${workspaceFolder}/.env` 。

全局，虚拟和conda环境

默认情况下，您安装的任何Python解释器都在其自己的**全局环境中**运行，该**环境**并不特定于任何一个项目。例如，如果仅在新命令提示符下运行 `python` （Windows）或 `python3` （macOS / Linux），则说明您正在该解释程序的全局环境中运行。因此，您安装或卸载的任何程序包都会影响全局环境以及在该上下文中运行的所有程序。

> **注意**：Python扩展版本2018.8.1及更高版本会自动更新环境。

尽管在全局环境中工作是一种入门的简便方法，但是随着时间的流逝，该环境会因为不同项目安装的许多不同软件包而变得混乱不堪。这样的混乱状况使得很难针对一组具有已知版本的特定软件包对应用程序进行全面测试，而这正是您在构建服务器或Web服务器上设置的那种环境。

因此，开发人员经常为项目创建**虚拟环境**。虚拟环境是项目中的子文件夹，其中包含特定解释器的副本。激活虚拟环境时，您安装的所有软件包仅安装在该环境的子文件夹中。然后在该环境中运行Python程序时，您会知道它仅针对那些特定的程序包运行。

> **注意**：虽然可以将虚拟环境文件夹作为工作区打开，但不建议这样做，否则可能会导致使用Python扩展程序出现问题。

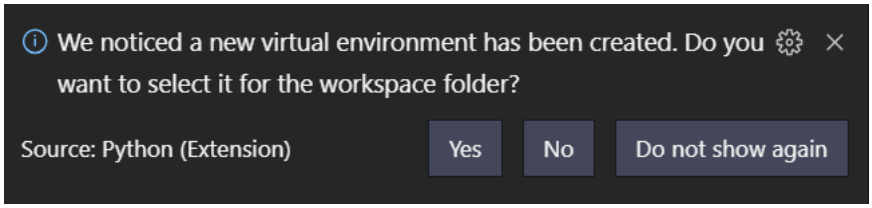
> **提示**：**conda环境**是使用 `conda` 程序包管理器创建和管理的虚拟环境。有关更多详细信息，请参见Conda环境。

要创建虚拟环境，请使用以下命令，其中“ .venv”是环境文件夹的名称：

```
# macOS/Linux
# You may need to run sudo apt-get install python3-venv first
python3 -m venv .venv

# Windows
# You can also use py -3 -m venv .venv
python -m venv .venv
```

创建新的虚拟环境时，将显示提示，允许您为工作空间选择它。



这会将Python解释器的路径从新的虚拟环境添加到您的工作区设置。然后，在安装软件包并通过Python扩展运行代码时将使用该环境。有关在项目中使用虚拟环境的示例，请参见Django教程 (/docs/python/tutorial-django)和Flask教程 (/docs/python/tutorial-flask)。

如果activate命令生成消息“ Activate.ps1没有经过数字签名。您无法在当前系统上运行此脚本。”，则需要临时更改PowerShell执行策略以允许脚本运行（请参阅“ 关于执行策略 (https://go.microsoft.com/fwlink/?LinkID=135170) ”）。 PowerShell文档）：

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process
```

> **注意**：如果您使用的Python扩展版本早于2018.10，并且在VS Code终端中创建了虚拟环境，则必须从“命令面板”中运行“ **重新加载窗口**”命令，然后使用**Python：选择**“**解释器**”以激活环境。如果VS Code识别虚拟环境有任何问题，请在文档资料库中提出问题 (https://github.com/Microsoft/vscode-docs/issues)，以便我们帮助确定原因。

> **提示**：准备将应用程序部署到其他计算机时，可以 `requirements.txt` 使用命令创建一个文件 `pip freeze > requirements.txt` （ `pip3` 在macOS / Linux上）。需求文件描述了您在虚拟环境中安装的软件包。仅使用此文件，您或其他开发人员就可以使用 `pip install -r requirements.txt` （或再次 `pip3` 在macOS / Linux上）还原那些软件包。通过使用需求文件，您无需将虚拟环境本身提交给源代码管理。

康达环境

conda环境是使用 `conda` 程序包管理器管理的Python环境（请参阅conda (http://conda.io)入门 (https://conda.io/projects/conda/en/latest/user-guide/getting-started.html)（conda.io (http://conda.io)））。Conda可以很好地创建具有相互依赖关系以及二进制包的环境。与虚拟环境（适用于项目）不同，conda环境可在任何给定计算机上全局使用。这种可用性使您可以轻松配置多个不同的conda环境，然后为任何给定项目选择合适的环境。

如前所述，Python扩展自动检测现有的conda环境，前提是该环境包含Python解释器。例如，以下命令使用Python 3.4解释器和几个库创建一个conda环境，然后VS Code在可用解释器列表中显示这些库：

```
conda create -n env-01 python=3.4 scipy=0.15.0 astroid babel
```

相反，如果您没有指定解释器（如） `conda create --name env-00` ，则环境不会出现在列表中。

有关conda (http://conda.io)命令行的更多信息，请参见Conda环境 (https://conda.io/docs/user-guide/tasks/manage-environments.html)（conda.io (http://conda.io)）。

补充笔记：

- 如果在VS Code运行时创建新的conda环境，请使用“ **重新加载窗口**”命令刷新**Python**所示的环境列表：**选择**“**解释器**”；否则，您可能看不到那里的环境。可能需要很短的时间才能出现。如果一开始看不到它，请等待15秒钟，然后再次尝试使用该命令。
- 为了确保从Shell角度正确设置环境，一种选择是使用带有已激活环境的Anaconda提示符，使用 `code .` 命令启动VS Code 。此时，您只需要使用命令面板或通过单击状态栏来选择解释器。
- 尽管VS Code的Python扩展目前尚未与conda environment.yml文件直接集成，但VS Code本身是一个出色的YAML编辑器。
- 如果默认外壳程序设置为PowerShell，则无法在VS Code集成终端中自动激活Conda环境。要更改外壳，请参阅集成终端-配置 (/docs/editor/integrated-terminal#_configuration)。
- 您可以手动指定要用于激活（版本4.4+）的conda可执行文件的路径。为此，请打开“命令面板”（ Ctrl + Shift + P ）并输入“**首选项**”：**打开**“**用户设置**”。然后 `python.condaPath` ，使用适当的路径在用户设置的Python扩展部分中设置。

手动指定翻译

如果VS Code不会自动找到您要使用的解释器，则可以在“工作区设置” `settings.json` 文件中手动设置它的路径。使用以下任何条目，您都可以将该行作为同级添加到其他现有设置中。）

首先，选择“ **文件**”（在macOS上为**Code**）> “**首选项**” > “**设置**”菜单命令（ Ctrl + , ）以打开“设置” (/docs/getstarted/settings)，然后选择“ **工作区**”。

然后执行以下任一步骤：

1. 创建或修改具有 `python.pythonPath` Python可执行文件完整路径的条目（如果 `settings.json` 直接进行编辑，请添加以下行作为设置）：

   例如：

   - 视窗：

```
"python.pythonPath": "c:/python36/python.exe",
```

- macOS / Linux：

```
"python.pythonPath": "/home/python36/python",
```

2. 您还可以使用 `python.pythonPath` 指向虚拟环境，例如：

   视窗：

   ```
   "python.pythonPath": "c:/dev/ala/venv/Scripts/python.exe",
   ```

   macOS / Linux：

   ```
   "python.pythonPath": "/home/abc/dev/ala/venv/bin/python",
   ```

3. 您可以使用语法在路径设置中使用环境变量 `${env:VARIABLE}` 。例如，如果您创建了一个名称 `PYTHON_INSTALL_LOC` 为解释器路径的变量，则可以使用以下设置值：

   ```
   "python.pythonPath": "${env:PYTHON_INSTALL_LOC}",
   ```

   > **注意**：变量替换仅在VS Code设置文件中受支持，在 `.env` 环境文件中将不起作用。

   通过使用环境变量，您可以轻松地在路径不同的操作系统之间传输项目，只需确保首先在操作系统上设置环境变量即可。

环境变量定义文件

环境变量定义文件是一个简单的文本文件，包含以形式的键值对 `environment_variable=value` ， `#` 用于注释。不支持多行值，但是值可以引用系统中或更早的文件中已定义的任何其他环境变量。有关更多信息，请参见变量替换。

默认情况下，Python扩展会查找并加载 `.env` 在当前工作空间文件夹中命名的文件，然后应用这些定义。该文件由 `"python.envFile": "${workspaceFolder}/.env"` 用户设置中的默认条目标识（请参阅常规设置 (/docs/python/settings-reference#_general-settings)）。您可以随时更改 `python.envFile` 设置以使用其他定义文件。

调试配置还包含一个 `envFile` 属性，该属性也默认为 `.env` 当前工作空间中的文件（请参见调试-设置配置选项 (/docs/python/debugging#_set-configuration-options)）。此属性使您可以轻松设置变量以进行调试，以替换默认 `.env` 文件中指定的变量。

例如，在开发Web应用程序时，您可能想轻松地在开发服务器和生产服务器之间切换。您可以为每个应用程序使用单独的定义文件，而不是直接将不同的URL和其他设置编码到应用程序中。例如：

dev.env文件

```
# dev.env - development configuration

# API endpoint
MYPROJECT_APIENDPOINT=https://my.domain.com/api/dev/

# Variables for the database
MYPROJECT_DBURL=https://my.domain.com/db/dev
MYPROJECT_DBUSER=devadmin
MYPROJECT_DBPASSWORD=!dfka**213=
```

prod.env文件

```
# prod.env - production configuration

# API endpoint
MYPROJECT_APIENDPOINT=https://my.domain.com/api/

# Variables for the database
MYPROJECT_DBURL=https://my.domain.com/db/
MYPROJECT_DBUSER=coreuser
MYPROJECT_DBPASSWORD=kKKfa98*11@
```

然后，您可以将设置 `python.envFile` 设置为 `${workspaceFolder}/prod.env` ，然后将 `envFile` 调试配置中的属性设置为 `${workspaceFolder}/dev.env` 。

> **注意**：使用多种方法指定环境变量时，请注意优先级顺序。设置（用户或工作空间） `.env` 指定的文件中包含的环境变量 `python.envFile` 将覆盖中 `envFile` 指定的变量 `launch.json` ，以及文件本身中 `env` 定义的任何变量 `launch.json` 。同样，在中 `envFile` 指定的环境变量 `launch.json` 将覆盖文件中 `env` 定义的变量 `launch.json` 。

变量替代

在定义文件中定义环境变量时，可以使用以下常规语法来使用任何现有环境变量的值：

```
<VARIABLE>=...${EXISTING_VARIABLE}...
```

其中 `...` 表示值中使用的任何其他文本。花括号是必需的。

在此语法中，适用以下规则：

- 变量按照它们在 `.env` 文件中出现的顺序进行处理，因此您可以使用文件中较早定义的任何变量。
- 单引号或双引号不影响替换值，而是包含在定义的值中。例如，如果值 `VAR1` 是 `abcedfg` ，则 `VAR2='${VAR1}'` 分配值 `'abcedfg'` 给 `VAR2` 。

- `$` 可以使用反斜杠对字符进行转义，如中所示 `\$` 。
- 您可以使用递归替换，例如 `PYTHONPATH=${PROJ_DIR}:${PYTHONPATH}` （ `PROJ_DIR` 其他环境变量在哪里）。
- 您只能使用简单替换； `${_${VAR1}_EX}` 不支持的嵌套。
- 语法不受支持的条目保持原样。

使用PYTHONPATH变量

在PYTHONPATH (https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH)环境变量指定其他地点在Python解释器应该寻找模块。在VS Code中，可以通过终端设置（terminal.integrated.env。*）和/或在 `.env` 文件中设置PYTHONPATH 。

使用终端设置时，PYTHONPATH会影响用户在终端内运行的任何工具，以及扩展对通过终端路由的用户执行的任何操作，例如调试。但是，在这种情况下，当扩展程序执行的操作没有通过终端进行路由时（例如使用linter或formatter），则此设置不会对模块查找产生影响。

使用 `.env` 文件设置PYTHONPATH时，它将影响扩展程序代表您执行的所有操作以及调试器执行的操作，但不会影响在终端中运行的工具。

如果需要，可以使用两种方法设置PYTHONPATH。

何时使用PYTHONPATH的示例是，如果您在 `src` 文件夹中包含源代码并在文件夹中进行测试 `tests` 。但是，在运行测试时， `src` 除非对相关路径进行硬编码，否则这些测试通常无法访问模块。要解决此问题，请将路径添加 `src` 到PYTHONPATH。

PYTHONPATH的值可以包含多个位置，这些位置用 `os.pathsep` 以下分隔： ；Windows上的分号（）和：Linux / macOS上的冒号（）。无效的路径将被忽略。如果发现PYTHONPATH的值未按预期运行，请确保在操作系统的位置之间使用正确的分隔符。例如，使用冒号分隔Windows上的位置，或使用分号分隔Linux / macOS上的位置会导致PYTHONPATH的值无效，该值将被忽略。

> **注意**：PYTHONPATH **未**指定Python解释器本身的路径，因此不应与该 `python.pythonPath` 设置一起使用。有关PYTHONPATH的其他信息，请阅读PYTHONPATH文档 (https://docs.python.org/3/using/cmdline.html#envvar-PYTHONPATH)。

下一步

- 编辑代码 (/docs/python/editing) -了解有关Python的自动完成，智能感知，格式化和重构的信息。
- 调试 (/docs/python/debugging) -学习在本地和远程调试Python。
- 测试 (/docs/python/testing) -配置测试环境并发现，运行和调试测试。
- 设置参考 (/docs/python/settings-reference) -在VS Code中探索与Python相关的所有设置。

2019年4月18日