TOPICS 调试 ▼

Visual Studio Code中的Python调试配置 (https://github.com/Microsoft/vscode-docs/blob/master/docs/python/debugging.md)

Python扩展支持调试多种类型的Python应用程序。有关基本调试的简短演练，请参见教程-配置和运行调试器 (/docs/python/python-tutorial#_configure-and-run-the-debugger)。另请参阅Flask教程 (/docs/python/tutorial-flask)。这两个教程都演示了诸如设置断点和单步执行代码之类的核心技能。

有关常规调试功能（例如检查变量，设置断点和其他与语言无关的活动），请查看VS Code调试 (/docs/editor/debugging)。

本文仅讨论特定于Python的那些注意事项，主要是特定于Python的调试*配置*，包括特定应用程序类型和远程调试的必要步骤。

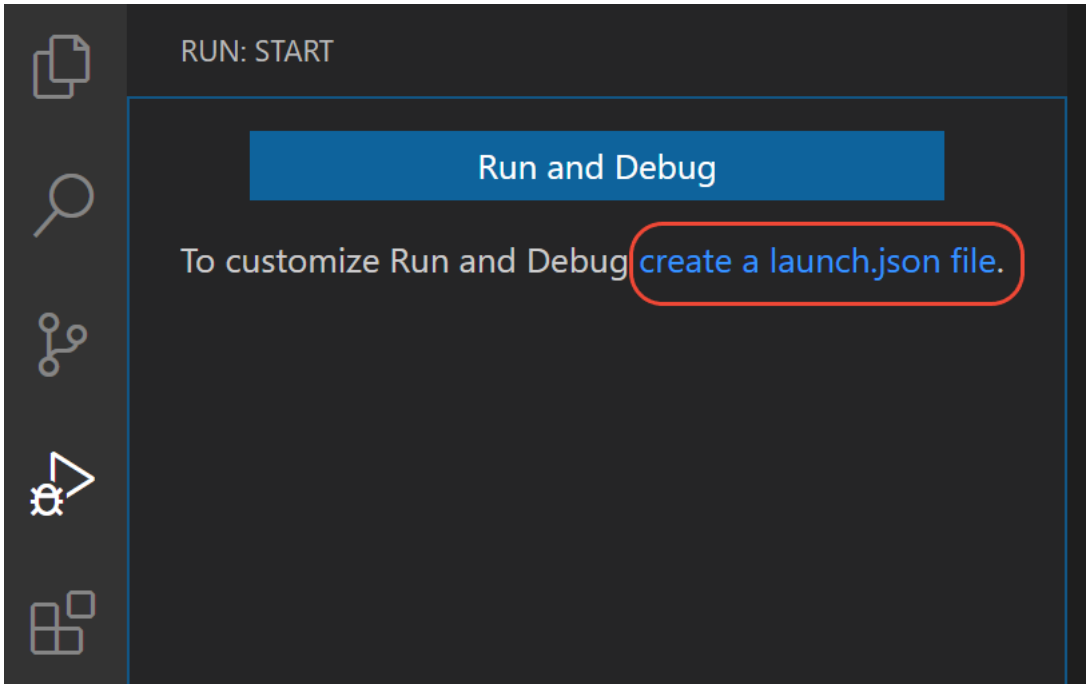
初始化配置

在调试会话期间，配置会驱动VS Code的行为。配置在 `launch.json` 存储在 `.vscode` 工作空间的文件夹中的文件中定义。

> **注意**为了更改调试配置，您的代码必须存储在文件夹中。

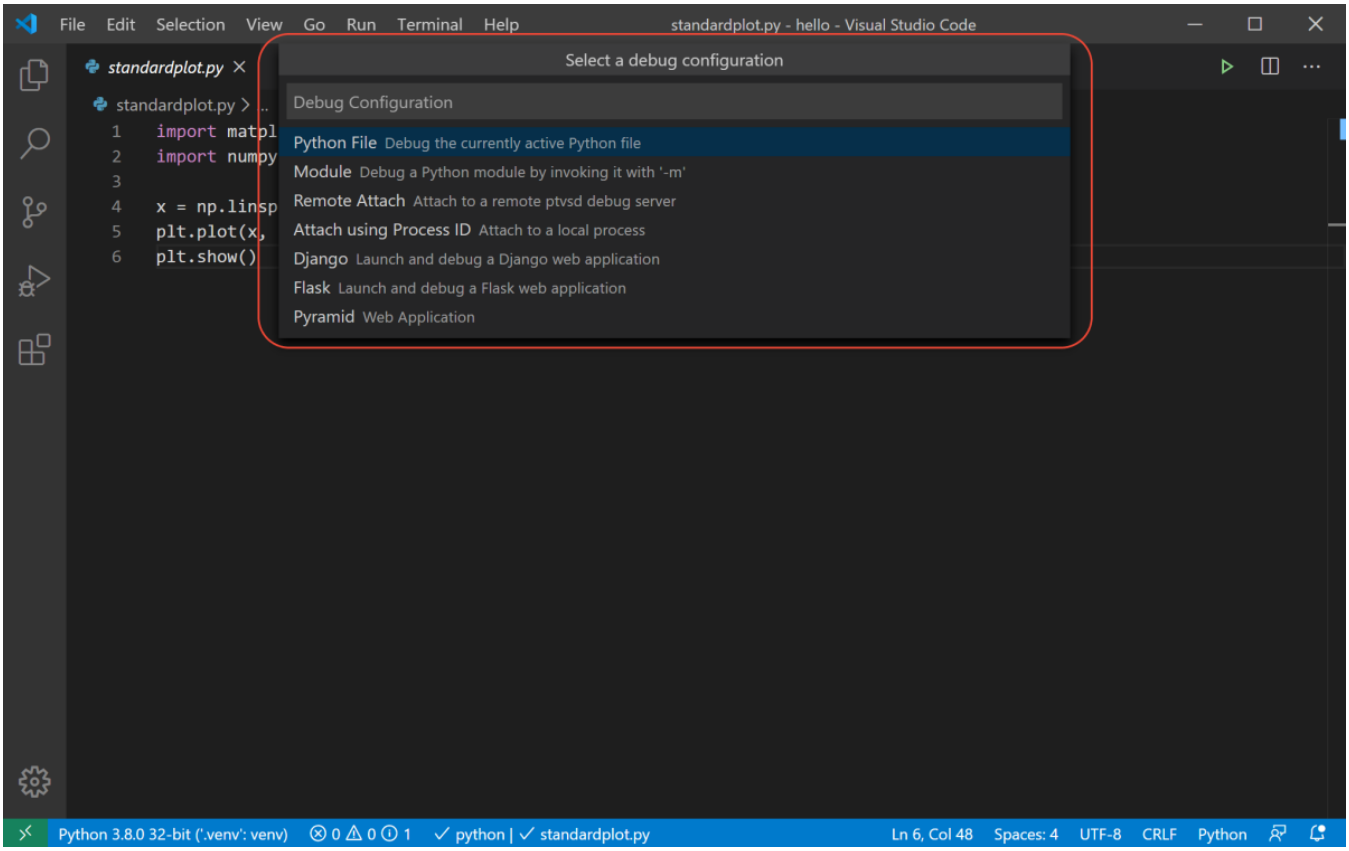
要初始化调试配置，请首先在侧栏中选择"运行"视图：

如果尚未定义任何配置，则会看到一个用于运行和调试的按钮，以及一个用于创建配置（launch.json）文件的链接：



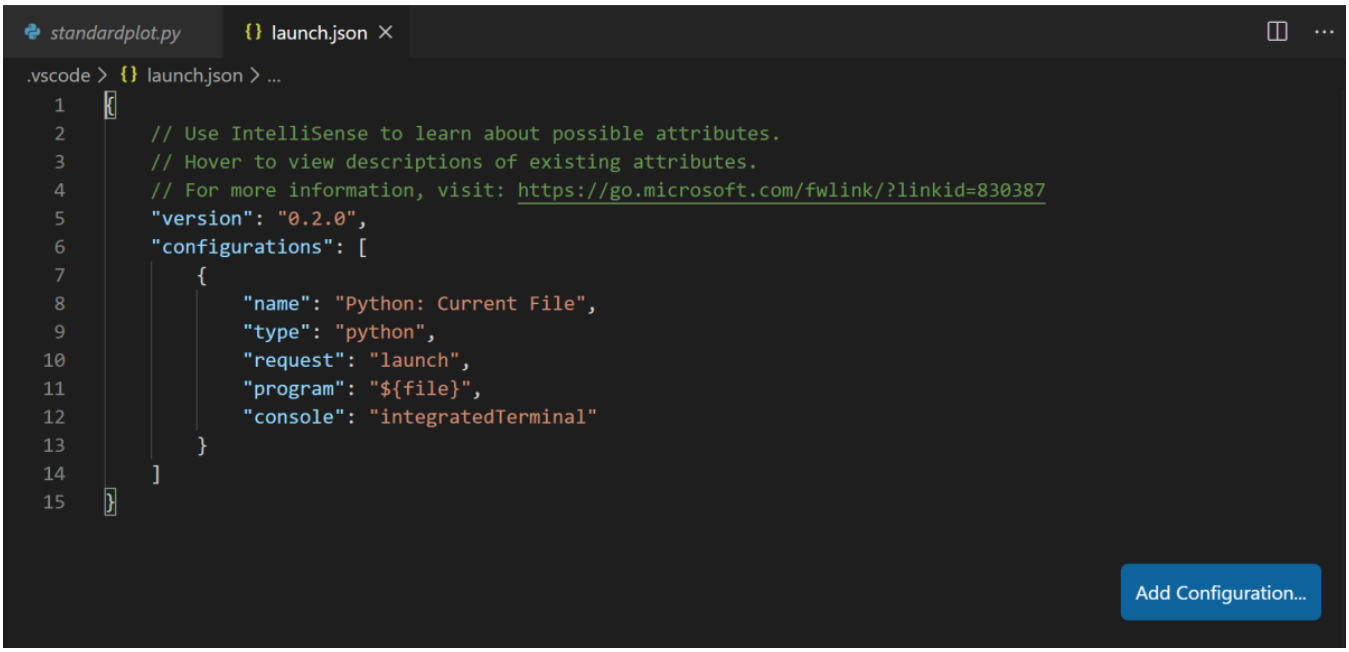
要 `launch.json` 使用Python配置生成文件，请执行以下步骤：

1. 单击**创建launch.json文件**链接（如上图所示），或使用" **运行**" >" **打开配置**"菜单命令。
2. 一个配置菜单将从Command Palette打开，您可以为打开的文件选择所需的调试配置类型。现在，在出现的" **选择调试配置**"菜单中，选择" **Python文件**"。



注意如果不存在任何配置，则通过"调试面板"，**F5**或"**运行**">"**开始调试**"启动调试会话时，也会调出调试配置菜单，但不会创建launch.json文件。

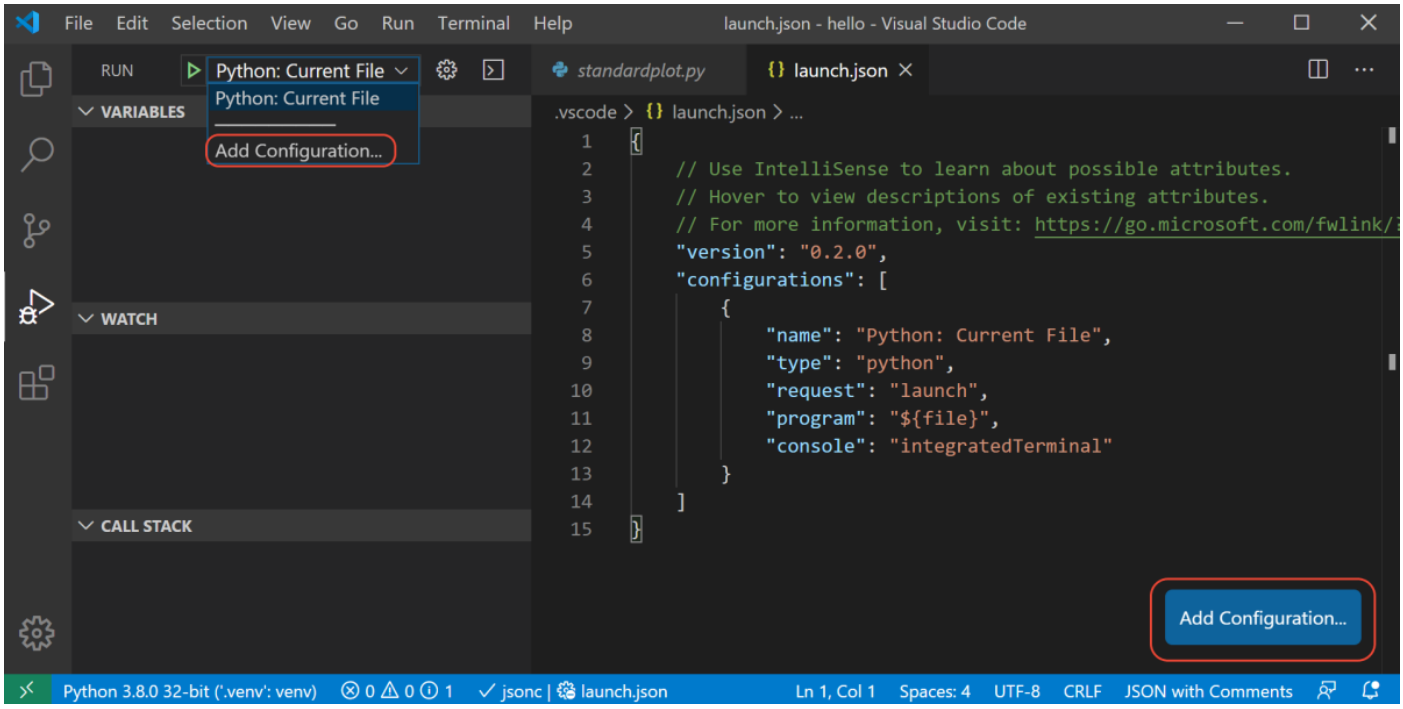
3. 然后，Python扩展程序会创建并打开一个 `launch.json` 文件，该文件包含基于您先前选择的内容的预定义配置，在本例中为**Python File**。您可以修改配置（例如，添加参数），也可以添加自定义配置。



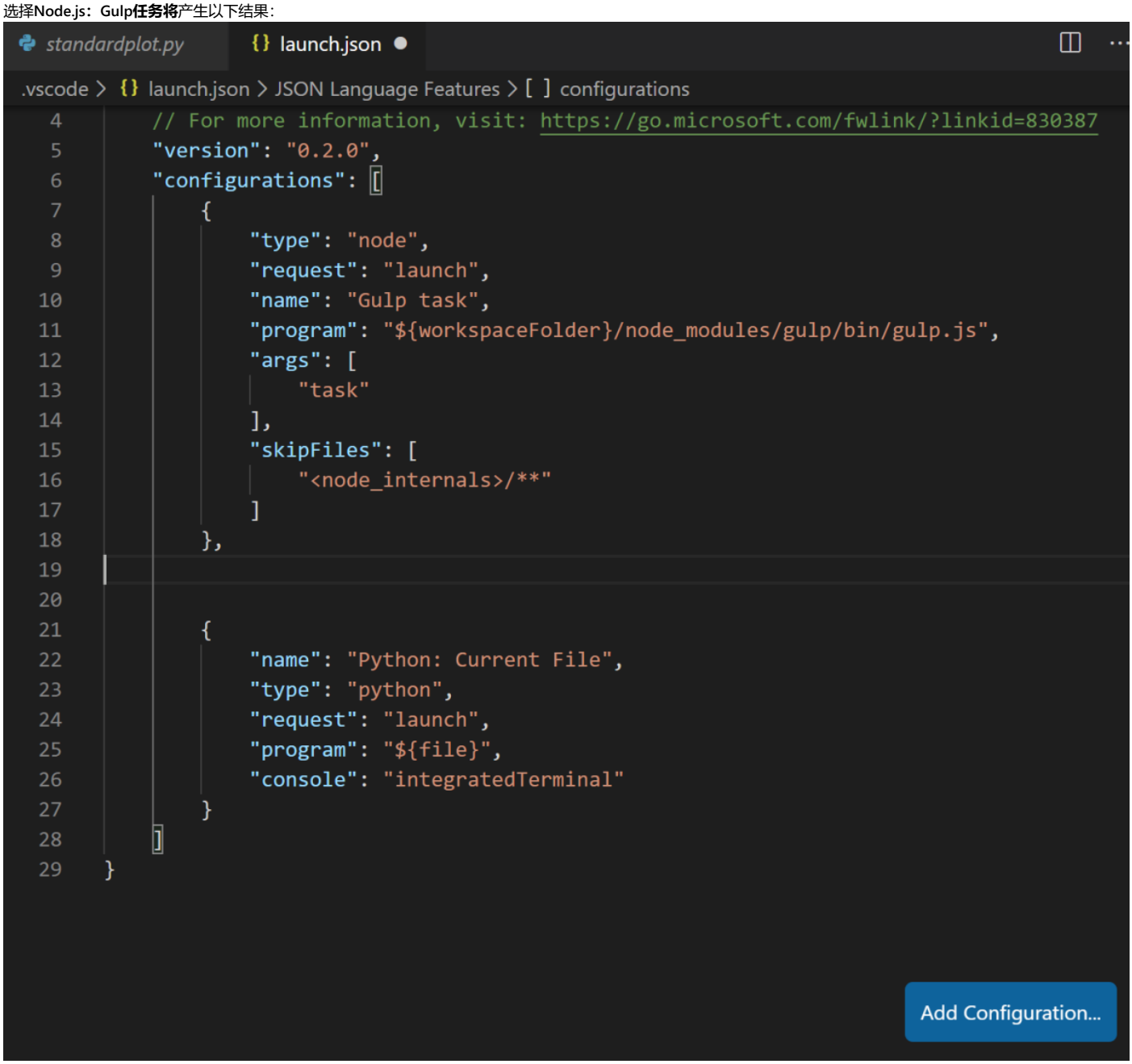
配置属性的详细信息将在本文后面的" 标准配置和选项"下进行介绍。本文的调试特定应用程序类型下还介绍了其他配置。

其他配置

默认情况下，VS Code仅显示Python扩展提供的最常见配置。您可以 `launch.json` 使用列表和编辑器中显示的" **添加配置**"命令选择其他配置 `launch.json` 。使用该命令时，VS Code会提示您所有可用配置的列表（请确保向下滚动以查看所有Python选项）：



选择Node.js：Gulp任务将产生以下结果：



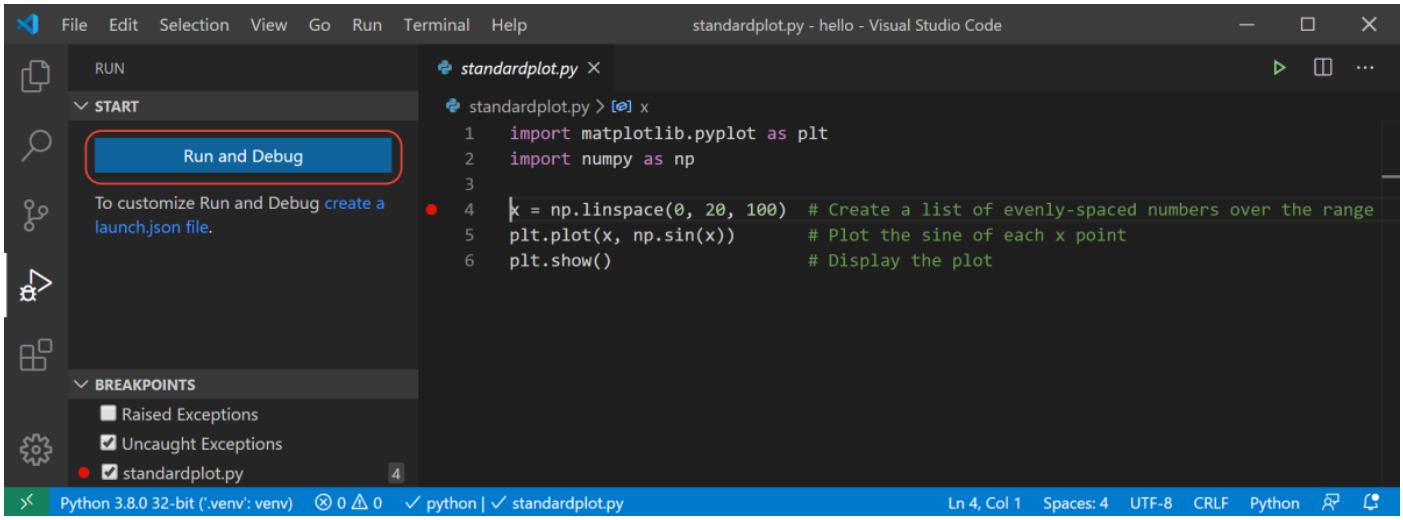
有关所有这些配置的详细信息，请参见调试特定的应用程序类型。

在调试过程中，状态栏显示当前配置和当前调试解释器。选择配置将显示一个列表，您可以从中选择其他配置：

默认情况下，调试器使用与 `python.pythonPath` VS Code其他功能相同的工作区设置。要专门使用其他解释器进行调试，请为适用的调试器配置设置 `python` in 的值 `launch.json` 。或者，在状态栏上选择命名的解释器以选择其他解释器。

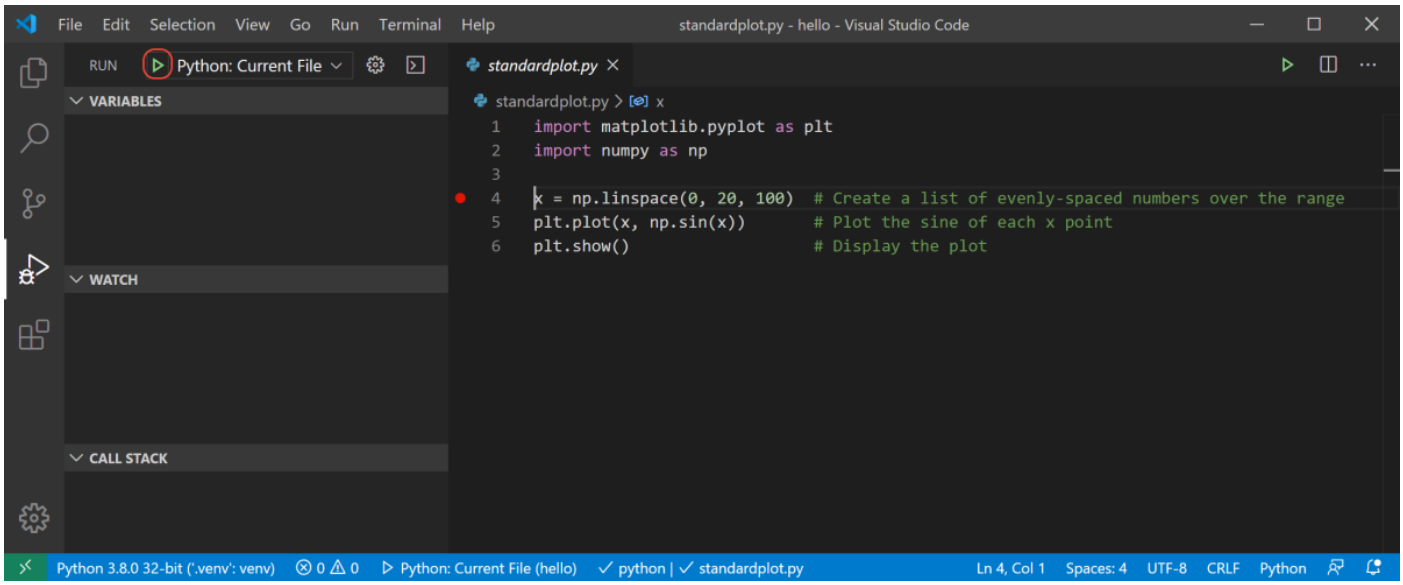
基本调试

开始调试Python文件的最简单方法是使用“ **运行**”视图，然后单击“ **运行并调试**”按钮。如果先前未设置任何配置，将显示调试选项列表。选择适当的选项以快速开始调试代码。



两种常见的选择是使用**Python文件**配置运行当前打开的Python文件，或使用**使用进程ID**附加配置将调试器附加到已经运行的进程。

有关创建和使用调试配置的信息，请参阅“ 初始化配置”和“ 其他配置”部分。添加配置后，可以从下拉列表中选择它，然后使用“ **开始调试**”按钮**启动它。**



命令行调试

调试器也可以从命令行运行。调试器命令行语法如下：

```
python -m debugpy
    --listen | --connect
    [<host>:]<port>
    [--wait-for-client]
    [--configure-<name> <value>]...
    [--log-to <path>] [--log-to-stderr]
    <filename> | -m <module> | -c <code> | --pid <pid>
    [<arg>]...
```

例如，可以从命令行使用指定的端口（5678）启动调试器，并使用以下语法启动脚本。本示例假定脚本长时间运行，并且省略了该 `--wait-for-client` 标志，这意味着脚本将不等待客户端附加。

```
python -m debugpy --listen 5678 ./myscript.py
```

然后，您将使用以下配置从VS Code Python扩展中附加。

```
{
  "name": "Python: Attach",
  "type": "python",
  "request": "attach",
  "connect": {
    "host": "localhost",
    "port": 5678
  }
}
```

> **注意**：指定host对于**listen**是可选的，默认情况下使用127.0.0.1。

如果要调试远程代码或在Docker容器中运行的代码，请在远程计算机或容器上，修改前一个CLI命令以指定主机。

```
python -m debugpy --listen 0.0.0.0:5678 ./myscript.py
```

关联的配置文件将如下所示。

```
{
  "name": "Attach",
  "type": "python",
  "request": "attach",
  "host": "remote-machine-name", // replace this with remote machine name
  "port": 5678
}
```

> **注意**：请注意，当您指定的主机值不是 `127.0.0.1` 或 `localhost` 您正在打开端口以允许从任何计算机进行访问时，这会带来安全风险。您应该确保在进行远程调试时采取了适当的安全预防措施，例如使用SSH隧道。

旗	选件	描述
- 听或**- 连接**	`[<host>:]<port>`	**必填项。**指定调试适配器服务器等待传入连接（--listen）或与正在等待传入连接的客户端（--connect）连接的主机地址和端口。与VS Code调试配置中使用的地址相同。默认情况下，主机地址为 `localhost (127.0.0.1)` 。
-等待客户	没有	**可选的。**指定在调试服务器建立连接之前，代码不应运行。此设置使您可以从代码的第一行进行调试。
-登录	`<path>`	**可选的。**指定用于保存日志的现有目录的路径。
--log-to-stderr	没有	**可选的。**启用debugpy将日志直接写入stderr。
--pid	`<pid>`	**可选的。**指定一个正在运行的进程，以将调试服务器注入到该进程中。
--configure-<名称>	`<value>`	**可选的。**设置客户端连接之前调试服务器必须知道的调试属性。这些属性可以直接在*启动*配置中使用，但必须以这种方式为*附加*配置设置。例如，如果您不希望调试服务器自动将其自身注入要附加到该进程的子进程中，请使用 `--configure-subProcess false` 。

> **注意**：`[<arg>]` 可用于将命令行参数传递给正在启动的应用程序。

通过网络连接进行调试

本地脚本调试

在某些情况下，您需要调试另一个进程在本地调用的Python脚本。例如，您可能正在调试针对特定处理作业运行不同Python脚本的Web服务器。在这种情况下，启动脚本后，您需要将VS Code调试器附加到脚本：

1. 运行VS Code，打开包含脚本的文件夹或工作区，然后 `launch.json` 为该工作区创建一个（如果还不存在）。
2. 在脚本代码中，添加以下内容并保存文件：

   ```
   import debugpy

   # 5678 is the default attach port in the VS Code debug configurations. Unless a host and port are specified, host defaults to 127.0.0.1
   debugpy.listen(5678)
   print("Waiting for debugger attach")
   debugpy.wait_for_client()
   debugpy.breakpoint()
   print('break on this line')
   ```

3. 使用**终端**打开**终端：创建新的集成终端**，它将激活脚本的选定环境。
4. 在终端中，使用安装调试包 `python -m pip install --upgrade debugpy` 。
5. 在终端中，使用脚本启动Python，例如 `python3 myscript.py` 。您应该看到代码中包含的"等待调试器附加"消息，并且脚本在 `debugpy.wait_for_client()` 调用时暂停。
6. 切换到"运行"视图，从调试器下拉列表中选择适当的配置，然后启动调试器。

7. 调试器应在 `debugpy.breakpoint()` 调用时停止，从这一点开始，您可以正常使用调试器。当然，您可以使用UI而不是使用脚本来在脚本代码中设置其他断点 `debugpy.breakpoint()` 。

使用SSH进行远程脚本调试

远程调试使您可以在VS Code上的程序在远程计算机上运行时在本地进行单步调试。无需在远程计算机上安装VS Code。为了提高安全性，在调试时，您可能希望或需要使用到远程计算机的安全连接（例如SSH）。

> **注意**：在Windows计算机上，您可能需要安装Windows 10 OpenSSH (https://docs.microsoft.com/windows-server/administration/openssh/openssh_install_firstuse)才能具有该 `ssh` 命令。

以下步骤概述了设置SSH隧道的一般过程。SSH隧道使您可以在本地计算机上工作，就好像您直接以远程方式在远程计算机上工作一样（比为公共访问而打开端口）。

在远程计算机上：

1. 通过打开 `sshd_config` 配置文件（ `/etc/ssh/` 在Linux和 `%programfiles(x86)%/openssh/etc` Windows 下均可找到）并添加或修改以下设置来启用端口转发：

   ```
   AllowTcpForwarding yes
   ```

   > **注意**：AllowTcpForwarding的默认值为yes，因此您可能不需要进行更改。

2. 如果必须添加或修改 `AllowTcpForwarding` ，请重新启动SSH服务器。在Linux / macOS上，运行 `sudo service ssh restart` ; 在Windows上，运行 `services.msc` ，找到并选择OpenSSH或 `sshd` 在服务列表中，然后选择**重新启动**。

在本地计算机上：

1. 通过运行 `ssh -2 -L sourceport:localhost:destinationport -i identityfile user@remoteaddress` 来创建SSH隧道，使用中选择的端口， `destinationport` 以及中的相应用户名和远程计算机的IP地址 `user@remoteaddress` 。例如，要使用IP地址1.2.3.4上的端口5678，命令应为 `ssh -2 -L 5678:localhost:5678 -i identityfile user@1.2.3.4` 。您可以使用该 `-i` 标志指定身份文件的路径。
2. 确认您可以在SSH会话中看到提示。
3. 在VS Code工作空间中，在 `launch.json` 文件中创建用于远程调试的配置，将端口设置为与 `ssh` 命令中使用的端口和主机匹配为 `localhost` 。您 `localhost` 在这里使用是因为您已经设置了SSH隧道。

   ```
   {
     "name": "Python: Attach",
     "type": "python",
     "request": "attach",
     "port": 5678,
     "host": "localhost",
     "pathMappings": [
       {
         "localRoot": "${workspaceFolder}", // Maps C:\Users\user1\project1
         "remoteRoot": "." // To current working directory ~/project1
       }
     ]
   }
   ```

开始调试

现在，已经为远程计算机设置了SSH隧道，您可以开始调试了。

1. 两台计算机：确保提供相同的源代码。
2. 两台计算机：安装debugpy (https://pypi.org/project/debugpy/)使用 `python -m pip install --upgrade debugpy` 您的环境中（使用时不需要虚拟环境中的一种形式，它是推荐的最佳做法）。
3. 远程计算机：有两种方法可以指定如何附加到远程进程。
   1. 在源代码中，添加以下行，并替换 `address` 为远程计算机的IP地址和端口号（此处显示的IP地址1.2.3.4仅用于说明）。

      ```
      import debugpy

      # Allow other computers to attach to debugpy at this IP address and port.
      debugpy.listen(('1.2.3.4', 5678))

      # Pause the program until a remote debugger is attached
      debugpy.wait_for_client()
      ```

      使用的IP地址 `listen` 应该是远程计算机的专用IP地址。然后，您可以正常启动程序，使其暂停直到调试器连接。

   2. 通过debugpy启动远程进程，例如：

      ```
      python3 -m debugpy --listen 1.2.3.4:5678 --wait-for-client -m myproject
      ```

      这将 `myproject` 使用来启动软件包 `python3` ，并使用远程计算机的专用IP地址作为 `1.2.3.4` 端口并监听端口 `5678` （您也可以通过指定文件路径而不是使用来启动远程Python进程 `-m` ，例如使用 `./hello.py` ）。
4. 本地计算机：**仅当您如上所述在远程计算机上修改了源代码时**，才在源代码中添加在远程计算机上添加的相同代码的注释掉的副本。添加这些行可确保两台计算机上的源代码逐行匹配。

```
#import debugpy

# Allow other computers to attach to debugpy at this IP address and port.
#debugpy.listen(('1.2.3.4', 5678))

# Pause the program until a remote debugger is attached
#debugpy.wait_for_client()
```

5. 本地计算机：切换到VS Code中的"运行"视图，选择**Python：附加**配置
6. 本地计算机：在要开始调试的代码中设置一个断点。
7. 本地计算机：使用修改后的**Python：附加**配置和"开始调试"按钮，启动VS Code调试器。VS Code应该在您本地设置的断点处停止，从而使您可以单步执行代码，检查变量并执行所有其他调试操作。您在**调试控制台**中输入的表达式也将在远程计算机上运行。

   从 `print` 语句输出到stdout的文本出现在两台计算机上。但是，其他输出（例如来自matplotlib之类的软件包中的图形图）仅出现在远程计算机上。
8. 在远程调试期间，调试工具栏显示如下：

   在此工具栏上，断开连接按钮（ Shift + F5 ）停止调试器，并允许远程程序运行完成。重新启动按钮（ Ctrl + Shift + F5 ）重新启动本地计算机上的调试器，但不会**不**重新启动远程程序。仅当您已经重新启动了远程程序并需要重新连接调试器时，才使用重新启动按钮。

设置配置选项

首次创建时 `launch.json` ，有两种标准配置可以在编辑器中的集成终端（在VS Code内）或外部终端（在VS Code内）中运行活动文件：

```
{
    "name": "Python: Current File (Integrated Terminal)",
    "type": "python",
    "request": "launch",
    "program": "${file}",
    "console": "integratedTerminal"
},
{
    "name": "Python: Current File (External Terminal)",
    "type": "python",
    "request": "launch",
    "program": "${file}",
    "console": "externalTerminal"
}
```

以下各节介绍了特定的设置。您还可以添加 `args` 标准配置中未包含的其他设置，例如，。

> **提示**：在项目中创建运行特定启动文件的配置通常很有帮助。例如，如果要在启动调试器时始终 `startup.py` 使用参数 `--port 1593` 启动，请如下创建配置条目：

```
 {
     "name": "Python: startup.py",
     "type": "python",
     "request": "launch",
     "program": "${workspaceFolder}/startup.py",
     "args" : ["--port", "1593"]
 },
```

`name`

提供出现在" VS代码"下拉列表中的调试配置的名称。

`type`

标识要使用的调试器的类型；将此设置保留 `python` 为Python代码。

`request`

指定开始调试的方式：

- `launch` ：在中指定的文件上启动调试器 `program`
- `attach` ：将调试器附加到已经运行的进程。有关示例，请参见远程调试。

`program`

提供python程序的入口模块（启动文件）的标准路径。价值 `${file}` ，往往在默认配置中使用，使用编辑器中的当前活动的文件。通过指定特定的启动文件，无论打开了哪些文件，始终可以确保使用相同的入口点启动程序。例如：

```
"program": "/Users/Me/Projects/PokemonGo-Bot/pokemongo_bot/event_handlers/__init__.py",
```

您还可以依赖于工作空间根目录中的相对路径。例如，如果根是， `/Users/Me/Projects/PokemonGo-Bot` 则可以使用以下命令：

```
"program": "${workspaceFolder}/pokemongo_bot/event_handlers/__init__.py",
```

`python`

指向用于调试的Python解释器的完整路径。

如果未指定，则此设置默认为设置中标识的解释器 `python.pythonPath` ，等效于使用value `${config:python.pythonPath}` 。要使用其他解释器，请在 `python` 调试配置的属性中指定其路径。

另外，您可以使用在每个平台上定义的自定义环境变量来包含要使用的Python解释器的完整路径，因此不需要其他文件夹路径。

如果需要将参数传递给Python解释器，则可以使用语法 `"python": ["<path>", "<arg>",...]` 。

`args`

指定要传递给Python程序的参数。用空格分隔的参数字符串的每个元素都应包含在引号中，例如：

```
"args": ["--quiet", "--norepeat", "--port", "1593"],
```

`stopOnEntry`

设置 `true` 为时，在要调试的程序的第一行中断调试器。如果省略（默认）或设置为 `false` ，则调试器将程序运行到第一个断点。

`console`

指定只要 `redirectOutput` 不更改默认设置，程序输出的显示方式。

值	显示输出的位置
`"internalConsole"`	**VS Code调试控制台。**如果 `redirectOutput` 设置为False，则不显示任何输出。
`"integratedTerminal"` （默认）	VS Code集成终端 (/docs/editor/integrated-terminal)。如果 `redirectOutput` 设置为True，则输出还将显示在调试控制台中。
`"externalTerminal"`	**单独的控制台窗口。**如果 `redirectOutput` 设置为True，则输出还将显示在调试控制台中。

`cwd`

指定调试器的当前工作目录，该目录是代码中使用的任何相对路径的基本文件夹。如果省略，则默认为 `${workspaceFolder}` （以VS Code打开的文件夹）。

例如，say `${workspaceFolder}` 包含一个包含的 `py_code` 文件夹 `app.py` ，一个包含的文件 `data` 夹 `salaries.csv` 。如果在上启动调试器 `py_code/app.py` ，则数据文件的相对路径会根据以下值而变化 `cwd` ：

电脑	数据文件的相对路径
省略或 `${workspaceFolder}`	`data/salaries.csv`
`${workspaceFolder}/py_code`	`../data/salaries.csv`
`${workspaceFolder}/data`	`salaries.csv`

`redirectOutput`

如果省略或设置为 `true` （internalConsole的默认值），则使调试器将程序中的所有输出打印到VS Code调试输出窗口中。如果设置为 `false` （IntegratedTerminal和externalTerminal的默认值），则程序输出不会显示在调试器输出窗口中。

使用 `"console": "integratedTerminal"` 或时通常会禁用此选项， `"console": "externalTerminal"` 因为无需在调试控制台中复制输出。

`justMyCode`

如果省略或设置为 `true` （默认值），则仅将调试限制为用户编写的代码。设置为 `false` 还启用标准库功能的调试。

`django`

设置 `true` 为时，将激活特定于Django Web框架的调试功能。

`sudo`

设置为 `true` 并与一起使用时 `"console": "externalTerminal"` ，允许调试需要提升的应用程序。必须使用外部控制台来捕获密码。

`pyramid`

设置 `true` 为时，请确保使用必要的 `pserve` 命令 (https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/startup.html?highlight=pserve)启动Pyramid应用程序。

`env`

为调试器进程设置可选的环境变量，而不是调试器始终继承的系统环境变量。这些变量的值必须以字符串形式输入。

`envFile`

包含环境变量定义的文件的可选路径。请参阅配置Python环境-环境变量定义文件 (/docs/python/environments#_environment-variable-definitions-file)。

`gevent`

如果设置为 `true` ，则启用调试gevent猴子修补的代码 (http://www.gevent.org/intro.html)。

断点和对数点

Python扩展支持调试代码的断点 (/docs/editor/debugging#_breakpoints)和日志 (/docs/editor/debugging#_logpoints)点 (/docs/editor/debugging#_breakpoints)。有关基本调试和使用断点的简短演练，请参见教程-配置和运行调试器 (/docs/python/python-tutorial#_configure-and-run-the-debugger)。

条件断点

也可以将断点设置为根据表达式，命中数或两者的组合触发。Python扩展支持命中计数，该计数是整数，以及==，>，> =，<，<=和%运算符后面的整数。例如，您可以通过将命中次数设置为来将断点设置为在5次出现后触发。 `>5` 有关更多信息，请参见主要VS Code调试文章中的条件断点 (/docs/editor/debugging#_conditional-breakpoints)。

在代码中调用断点

在Python代码中，您可以 `debugpy.breakpoint()` 在调试会话期间的任何要暂停调试器的位置进行调用。

断点验证

Python扩展会自动检测在不可执行的行（例如 `pass` 语句或多行语句的中间）上设置的断点。在这种情况下，运行调试器会将断点移至最近的有效行，以确保代码执行在该点停止。

调试特定的应用程序类型

配置下拉菜单为常规应用类型提供了多种不同的选项：

组态	描述
连接	请参阅上一节中的远程调试。
Django的	指定 `"program": "${workspaceFolder}/manage.py"` ， `"args": ["runserver", "--noreload"]` 和 `"console": "integratedTerminal"` 。还添加了 `"django": true` 用于调试Django HTML模板的功能。请注意，调试时无法自动重新加载Django应用。
烧瓶	请参阅下面的烧瓶调试。
Gevent	添加 `"gevent": true` 到标准的集成终端配置。
金字塔	删除 `program` ，添加 `"args": ["${workspaceFolder}/development.ini"]` ，添加 `"jinja": true` 以启用模板调试，并添加 `"pyramid": true` 以确保使用必要的 `pserve` 命令 (https://docs.pylonsproject.org/projects/pyramid/en/latest/narr/startup.html?highlight=pserve)启动程序。
cra草	指定 `"module": "scrapy"` 并添加 `"args": ["crawl", "specs", "-o", "bikes.json"]` 。
沃森	指定 `"program": "${workspaceFolder}/console.py"` 和 `"args": ["dev", "runserver", "--noreload=True"]` 。

远程调试和Google App Engine也需要特定的步骤。有关调试测试（包括鼻子测试 (/docs/python/testing)）的详细信息，请参见测试 (/docs/python/testing)。

要调试需要管理员权限的应用，请使用 `"console": "externalTerminal"` 和 `"sudo": "True"` 。

烧瓶调试

```
{
    "name": "Python: Flask",
    "type": "python",
    "request": "launch",
    "module": "flask",
    "env": {
        "FLASK_APP": "app.py"
    },
    "args": [
        "run",
        "--no-debugger",
        "--no-reload"
    ],
    "jinja": true
},
```

如您所见，此配置指定 `"env": {"FLASK_APP": "app.py"}` 和 `"args": ["run", "--no-debugger","--no-reload"]` 。使用该 `"module": "flask"` 属性代替 `program` 。（您可能会 `"FLASK_APP": "${workspaceFolder}/app.py"` 在 `env` 属性中看到，在这种情况下，将配置修改为仅引用文件名。否则，您可能会看到“无法导入模块C”错误，其中C是驱动器号。）

该 `"jinja": true` 设置还允许调试Flask的默认Jinja模板引擎。

如果要在开发模式下运行Flask的开发服务器，请使用以下配置：

```
{
    "name": "Python: Flask (development mode)",
    "type": "python",
    "request": "launch",
    "module": "flask",
    "env": {
        "FLASK_APP": "app.py",
        "FLASK_ENV": "development"
    },
    "args": [
        "run"
    ],
    "jinja": true
},
```

故障排除

调试器可能无法运行的原因有很多。调试控制台通常会显示特定的原因，但是有两个特定的原因如下：

- python可执行文件的路径不正确：请检查 `python.pythonPath` 用户设置中的值。
- 监视窗口中有无效的表达式：从监视窗口中清除所有表达式，然后重新启动调试器。
- 如果您使用的是使用本机线程API（例如Win32 `CreateThread` 函数而不是Python线程API）的多线程应用程序，则目前有必要在您要调试的任何文件的顶部都包含以下源代码：

  ```
  import debugpy
  debugpy.debug_this_thread()
  ```

下一步

- Python环境 (/docs/python/environments) -控制使用哪个Python解释器进行编辑和调试。
- 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。
- 常规调试 (/docs/editor/debugging) -了解VS Code的调试功能。

5/6/2020