
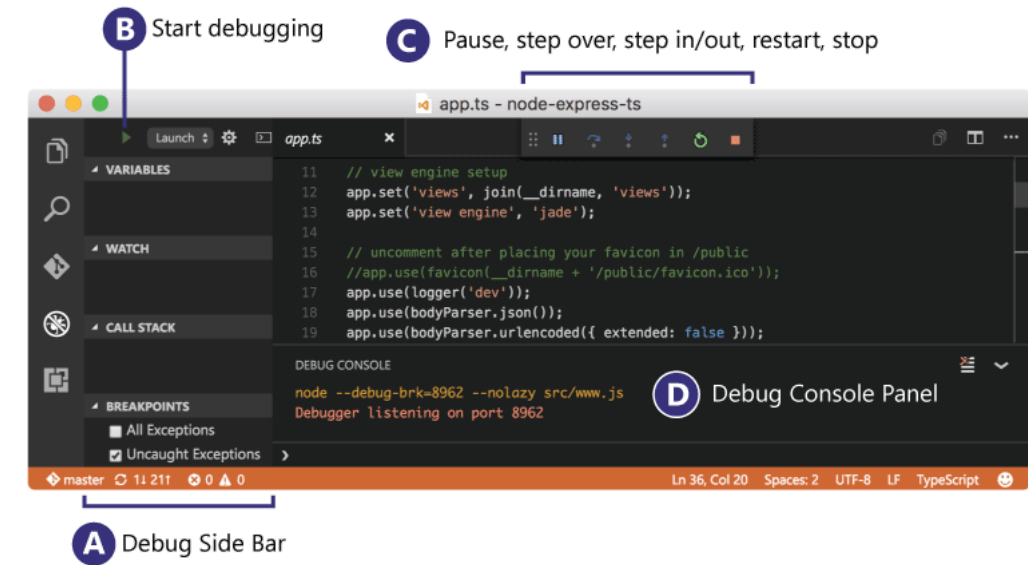


TOPICS Debugging

调试

 (https://github.com/Microsoft/vscode-docs/blob/master/docs/editor/debugging.md)

Visual Studio Code的主要功能之一是其强大的调试支持。VS Code的内置调试器有助于加速您的编辑，编译和调试循环。







调试器扩展

VS Code具有对Node.js (https://nodejs.org/)运行时的内置调试支持，并且可以调试JavaScript，TypeScript或任何其他可转换为JavaScript的语言。

要调试其他语言和运行时（包括PHP (https://marketplace.visualstudio.com/items?itemName=felixfbecker.php-debug)，Ruby (https://marketplace.visualstudio.com/items?itemName=rebornix.Ruby)，Go (https://marketplace.visualstudio.com/items?itemName=golang.go)，C# (https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)，Python (https://marketplace.visualstudio.com/items?itemName=ms-python.python)，C++ (https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools)，PowerShell (https://marketplace.visualstudio.com/items?itemName=ms-vscode.PowerShell)和许多其他语言 (https://marketplace.visualstudio.com/search?term=debug&target=VSCode&category=Debuggers&sortBy=Relevance))，请在我们的VS Code Marketplace中 (https://marketplace.visualstudio.com/vscode/Debuggers)查找 Debuggers 扩展 (/docs/editor/extension-gallery)，或在顶层“运行”菜单中选择“安装其他调试器”。(https://marketplace.visualstudio.com/vscode/Debuggers)

以下是一些流行的扩展，其中包括调试支持：

 Python ms-python 21.8M Linting, Debugging (multi-threaded, remote), Inte...	 C/C++ ms-vscode 12.3M C/C++ IntelliSense, debugging, and code browsing.	 C# ms-dotnettools 8.0M C# for Visual Studio Code (powered by OmniSharp).	 Debugger for Chrome msjsdiag 5.9M Debug your JavaScript code in the Chrome browser,...
(https://marketplace.visualstudio.com/items?itemName=ms-python.python)	(https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools)	(https://marketplace.visualstudio.com/items?itemName=ms-dotnettools.csharp)	(https://marketplace.visualstudio.com/items?itemName=msjsdiag.debugger-for-chrome)

提示：上面显示的扩展名是动态查询的。选择上方的扩展程序磁贴以阅读说明并查看，以确定最适合您的扩展程序。

开始调试

以下文档基于内置的Node.js (https://nodejs.org/)调试器，但是大多数概念和功能也适用于其他调试器。

在阅读有关调试的信息之前，首先创建一个示例Node.js应用程序会很有帮助。您可以按照Node.js演练 (/docs/nodejs/nodejs-tutorial)来安装Node.js并创建一个简单的“Hello World” JavaScript应用程序（app.js）。设置完简单的应用程序后，此页面将带您进入VS Code调试功能。

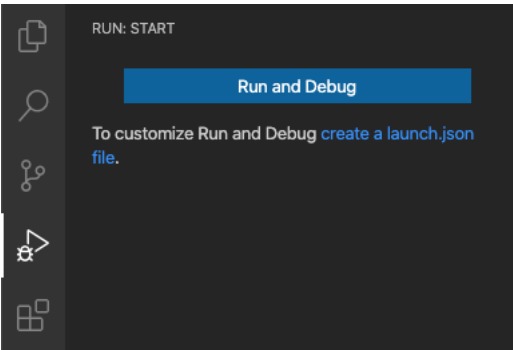
运行视图

要调出“运行”视图，请在“VS代码”侧面的活动栏中选择“运行”图标。您也可以使用键盘快捷键 **Ctrl + Shift + d**。



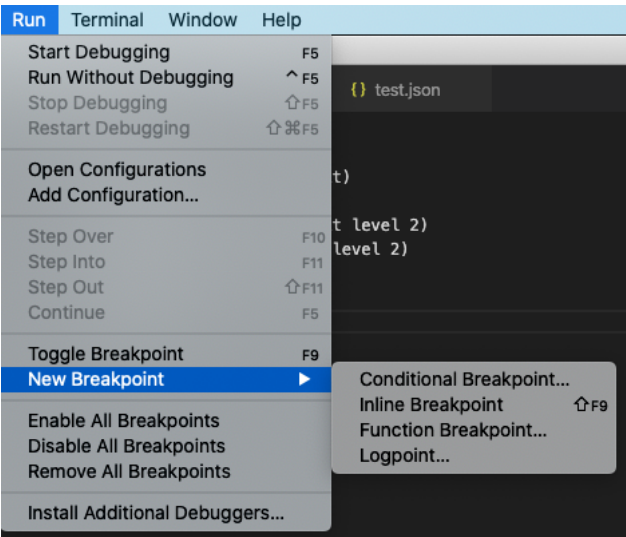
“运行”视图显示与运行和调试有关的所有信息，并在顶部带有调试命令和配置设置的栏。

如果尚未配置运行和调试（尚未 `launch.json` 创建），我们将显示“运行开始”视图。



运行菜单

顶层“运行”菜单包含最常见的运行和调试命令：

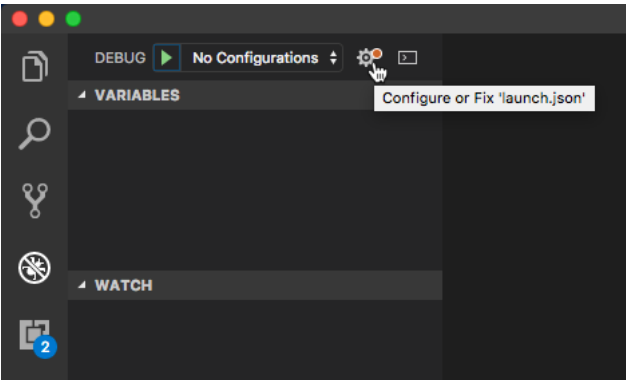


启动配置

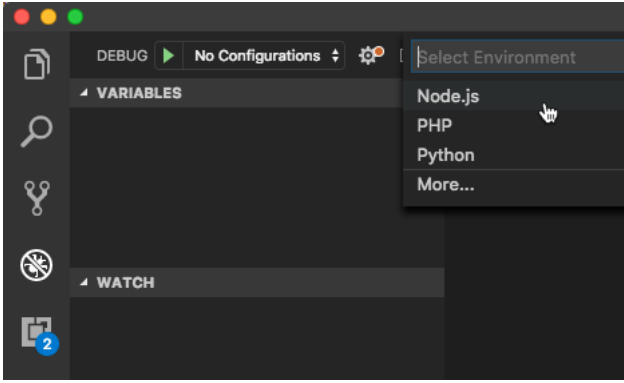
要在VS Code中运行或调试一个简单的应用程序，请按 `F5` 键，VS Code会尝试运行您当前处于活动状态的文件。

但是，对于大多数调试方案而言，创建启动配置文件是有好处的，因为它使您可以配置和保存调试设置详细信息。VS Code将调试配置信息保留在工作空间 `launch.json` 中的 `.vscode` 文件夹（项目根文件夹）中或用户设置 (`/docs/editor/debugging#_global-launch-configuration`)或工作空间设置中的文件中 (`/docs/editor/multi-root-workspaces#_workspace-launch-configurations`)。

要创建 `launch.json` 文件，请在VS Code中打开项目文件夹（`File > Open Folder`），然后在Run视图顶部栏上选择Configure gear图标。



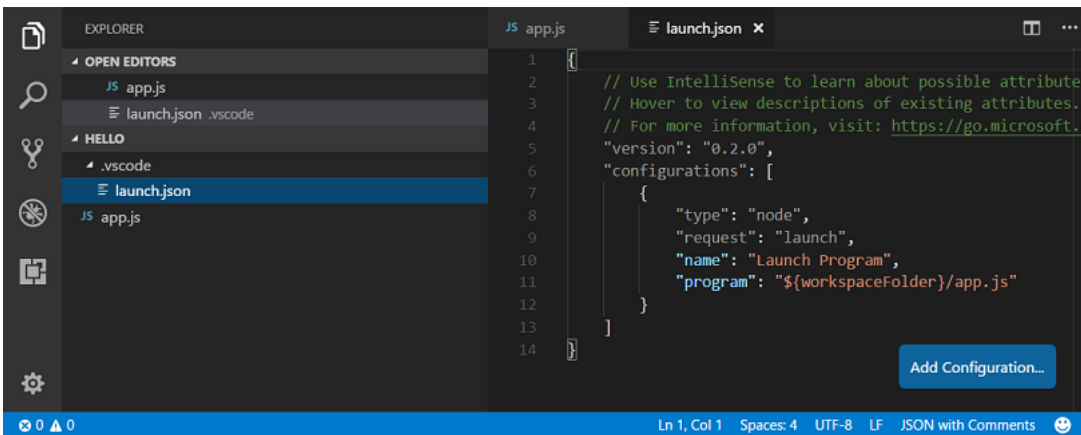
VS Code将尝试自动检测您的调试环境，但是如果失败，则必须手动选择它：



这是为Node.js调试生成的启动配置：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${file}"
    }
  ]
}
```

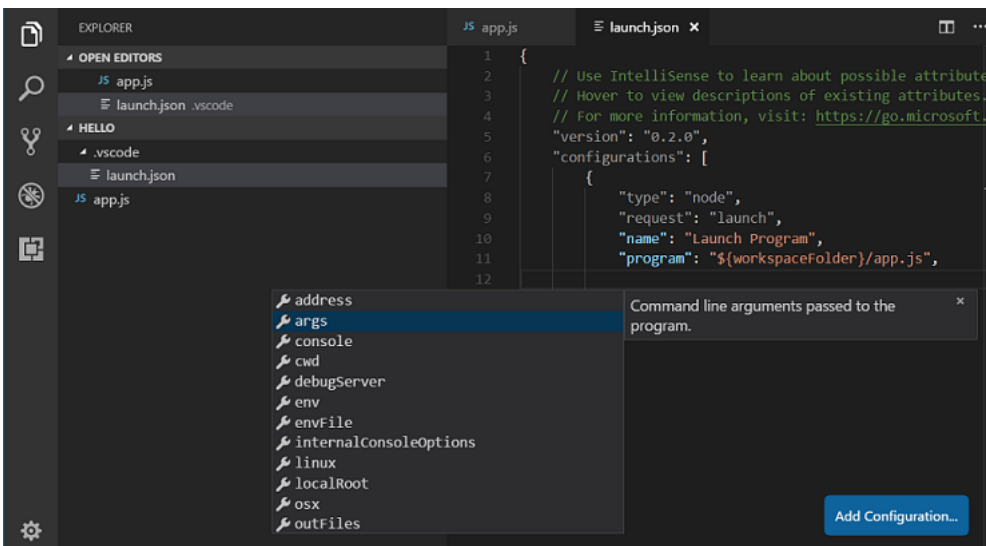
如果返回到“文件资源管理器”视图（Ctrl + Shift + E），您将看到VS Code创建了一个 .vscode 文件夹并将该 launch.json 文件添加到您的工作区中。



注意：即使您没有在VS Code中打开文件夹，也可以调试简单的应用程序，但是无法管理启动配置和设置高级调试。如果没有打开文件夹，“VS代码状态栏”将显示为紫色。

请注意，启动配置中可用的属性因调试器而异。您可以使用IntelliSense建议（Ctrl + Space）来找出特定调试器存在的属性。悬停帮助也可用于所有属性。

不要假定一个调试器可用的属性也自动适用于其他调试器。如果在启动配置中看到绿色的波浪形，请将鼠标悬停在它们上方以了解问题所在，并在启动调试会话之前尝试修复它们。



查看所有自动生成的值，并确保它们对于您的项目和调试环境有意义。

启动与附加配置

在VS Code中，有两种核心调试模式，**Launch**和**Attach**，它们处理两种不同的工作流程和开发人员细分。根据您的工作流程，知道哪种配置类型适合您的项目可能会造成混淆。

如果您来自浏览器开发人员工具背景，则可能不习惯“从工具中启动”，因为您的浏览器实例已经打开。打开DevTools时，只需将 DevTools **附加**到打开的浏览器选项卡上。另一方面，如果您来自服务器或桌面背景，则让您的编辑器为您**启动**进程是很正常的，并且您的编辑器会自动将其调试器附加到新启动的进程。

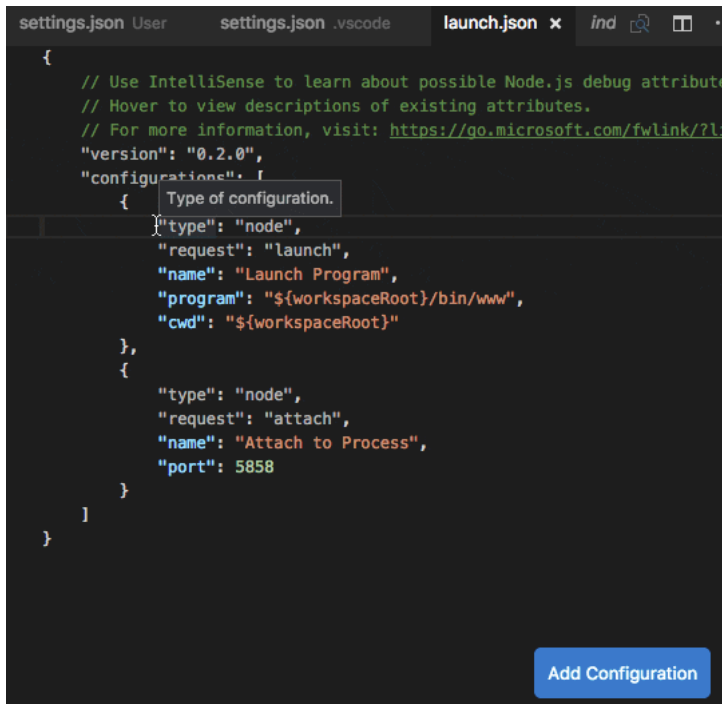
解释**启动**和**附加**区别的最好的方法是将**启动**配置视为**在将** VS Code附加到应用程序**之前**如何在调试模式下启动应用程序的方法，而**附加**配置则是如何连接VS Code的方法的方法调试器**已运行**的应用程序或进程。

VS Code调试器通常支持以调试模式启动程序或以调试模式附加到已运行的程序。根据请求（`attach` 或 `launch`），需要不同的属性，VS Code的 `launch.json` 验证和建议应对此有所帮助。

添加新配置

要将新配置添加到现有 `launch.json`，请使用以下技术之一：

- 如果您的光标位于配置数组中，请使用IntelliSense。
- 按“**添加配置**”按钮以在阵列的开头调用代码段IntelliSense。
- 在“运行”菜单中选择“**添加配置**”选项。

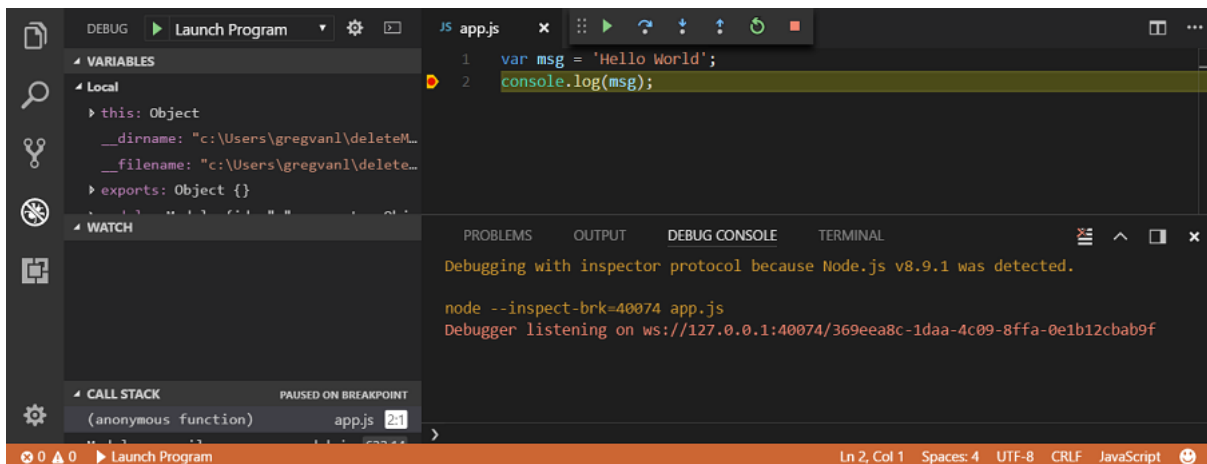


VS Code还支持复合启动配置，以便同时启动多个配置。有关更多详细信息，请阅读本节。

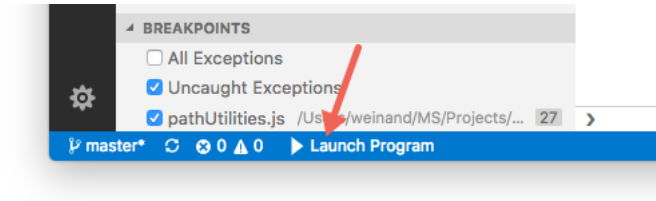
为了启动调试会话，请首先在“运行”视图中使用“**配置**”下拉列表选择名为“**启动程序**”的配置。设置启动配置后，请使用 F5 启动调试会话。

或者，您可以通过“**命令面板**”（Ctrl + Shift + P）来运行配置，方法是在“**调试**”上进行筛选：**选择并开始调试**或键入 `'debug'`，然后选择要调试的配置。

调试会话开始后，将立即显示DEBUG CONSOLE面板并显示调试输出，并且状态栏会更改颜色（默认颜色主题为橙色）：



此外，**调试状态**将显示在状态栏中，显示活动的调试配置。通过选择调试状态，用户可以更改活动的启动配置并开始调试，而无需打开“运行”视图。



调试动作

调试会话开始后，“调试”工具栏将出现在编辑器的顶部。



- 继续/暂停 F5
- 越过 F10
- 进入 F11
- 移出 Shift + F11
- 重新启动 Ctrl + Shift + F5
- 停止 Shift + F5

提示： 使用设置 `debug.toolBarLocation` 来控制调试工具栏的位置。它可以是“运行”视图的默认值 `floating`，也可以是。一个调试工具栏可以水平，也拖索到编辑区。 `docked hidden floating`

运行方式

除了调试程序外，VS Code还支持**运行程序**。Ctrl: F5 触发“**调试：运行（不调试开始）**”操作，并使用当前选择的启动配置。“运行”模式支持许多启动配置属性。VS Code 在程序运行时维护调试会话，然后按“**停止**”按钮将终止程序。

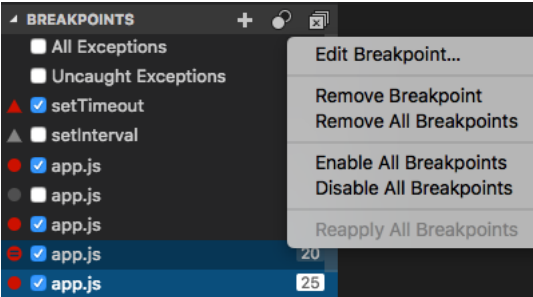
提示： “运行”操作始终可用，但并非所有调试器扩展都支持“运行”。在这种情况下，“运行”将与“调试”相同。

断点

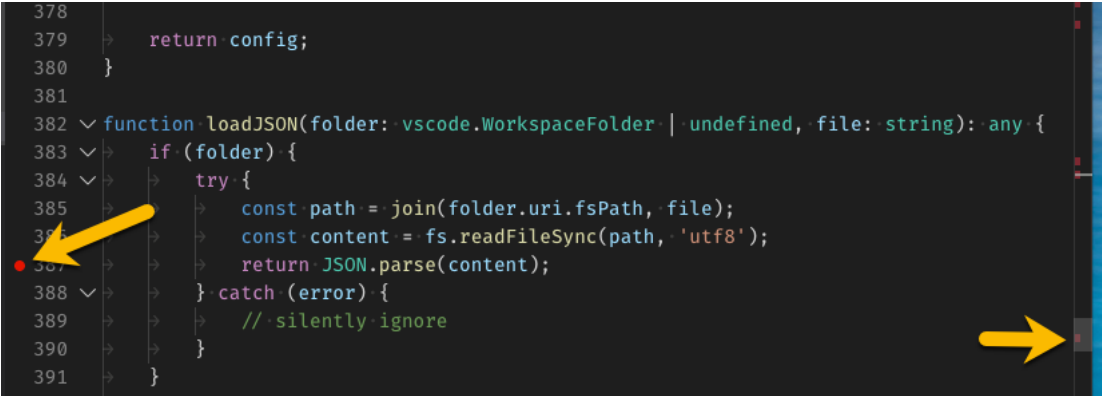
可以通过单击**编辑器边距**或在当前行上使用 F9 来切换断点。可以在“运行”视图的“**BREAKPOINTS**”部分中进行更精细的断点控制（启用/禁用/重新应用）。

- 编辑器边距中的断点通常显示为红色实心圆。
- 禁用的断点有一个填充的灰色圆圈。
- 当调试会话开始时，无法在调试器中注册的断点将变为灰色空心圆。如果在运行没有实时编辑支持的调试会话时编辑源，则可能会发生同样的情况。

“**重新应用所有断点**”命令将所有断点再次设置到其原始位置。如果您的调试环境是尚未执行的源代码中的“惰性”和“错位”断点，这将很有帮助。



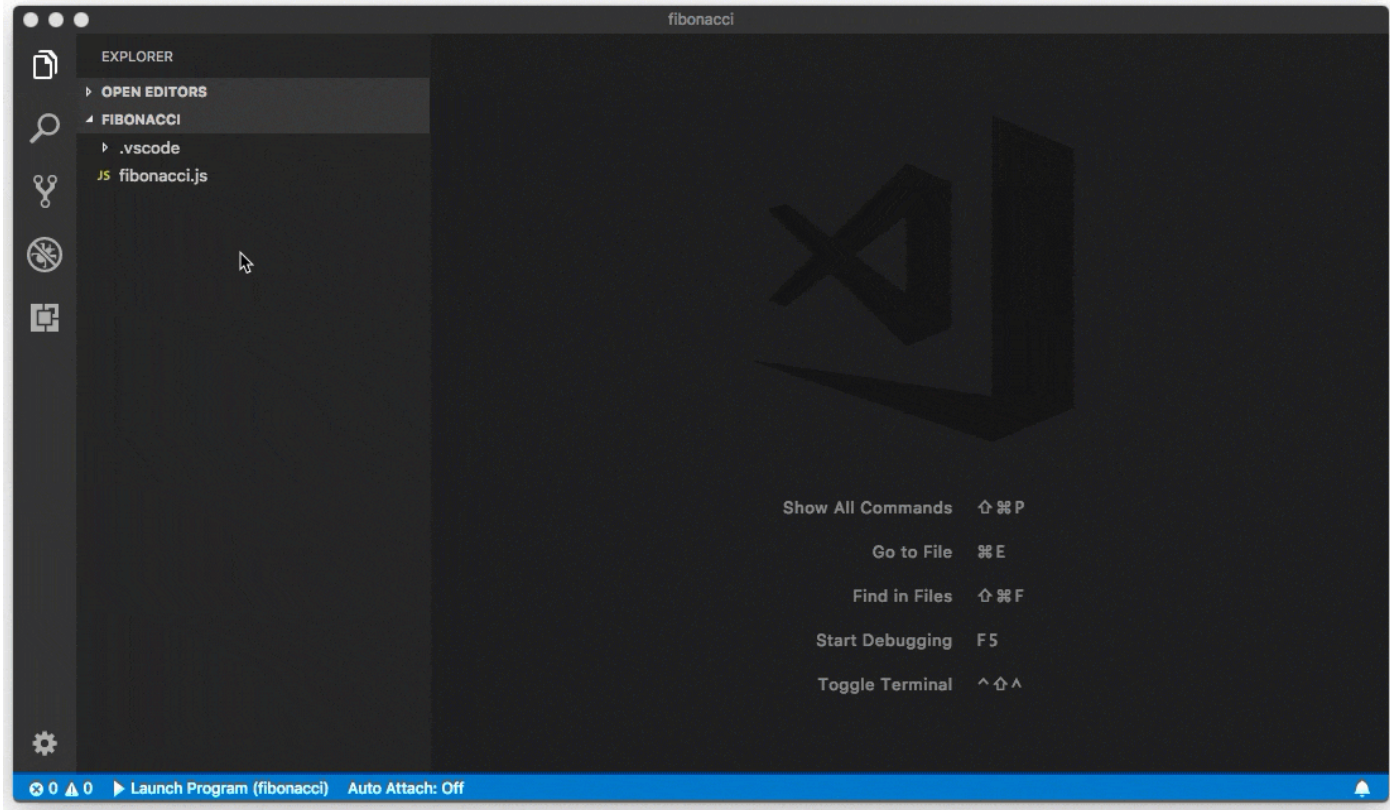
通过启用以下设置，可以选择在编辑器的概述标尺中显示断点 `debug.showBreakpointsInOverviewRuler`：



日志点

Logpoint是断点的变体，它不会“闯入”调试器，而是将消息记录到控制台。日志点对于在调试无法暂停或停止的生产服务器时注入日志特别有用。

Logpoint由“菱形”形状的图标表示。日志消息是纯文本，但可以包含要在花括号（'{}'）中计算的表达式。

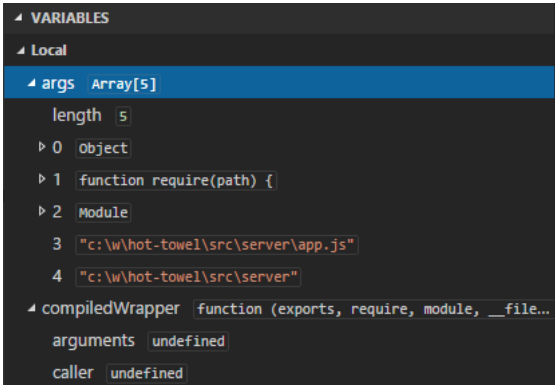


与常规断点一样，可以启用或禁用对数点，也可以通过条件和/或命中数来控制它们。

注意：VS Code的内置Node.js调试器支持日志点，但其他调试扩展也可以实现。在Python中 (/docs/python/python-tutorial)和Java的 (/docs/java/java-tutorial)扩展，例如，支持Logpoints。

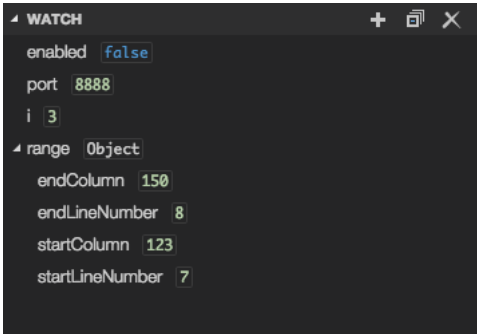
资料检查

可以在“运行”视图的“变量”部分中检查变量，也可以将其悬停在编辑器中的源上进行检查。变量值和表达式评估相对于“调用堆栈”部分中的选定堆栈帧。

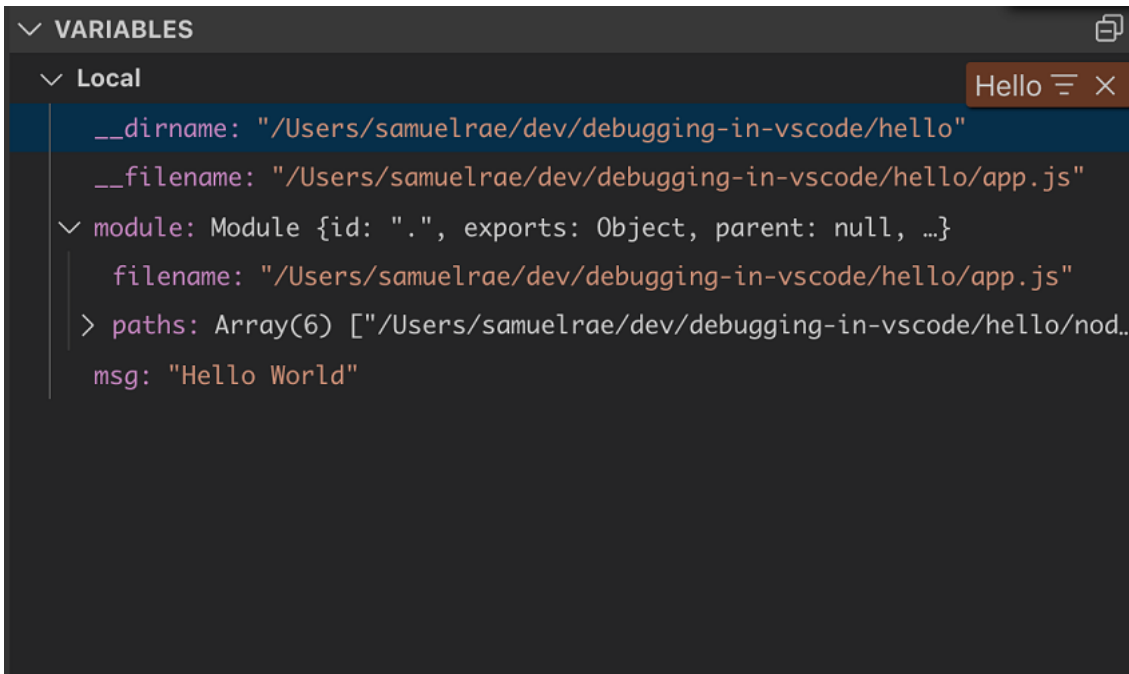


可以使用变量的上下文菜单中的“设置值”操作来修改变量值。

变量和表达式也可以进行评估，并在运行视图的查看手表节。

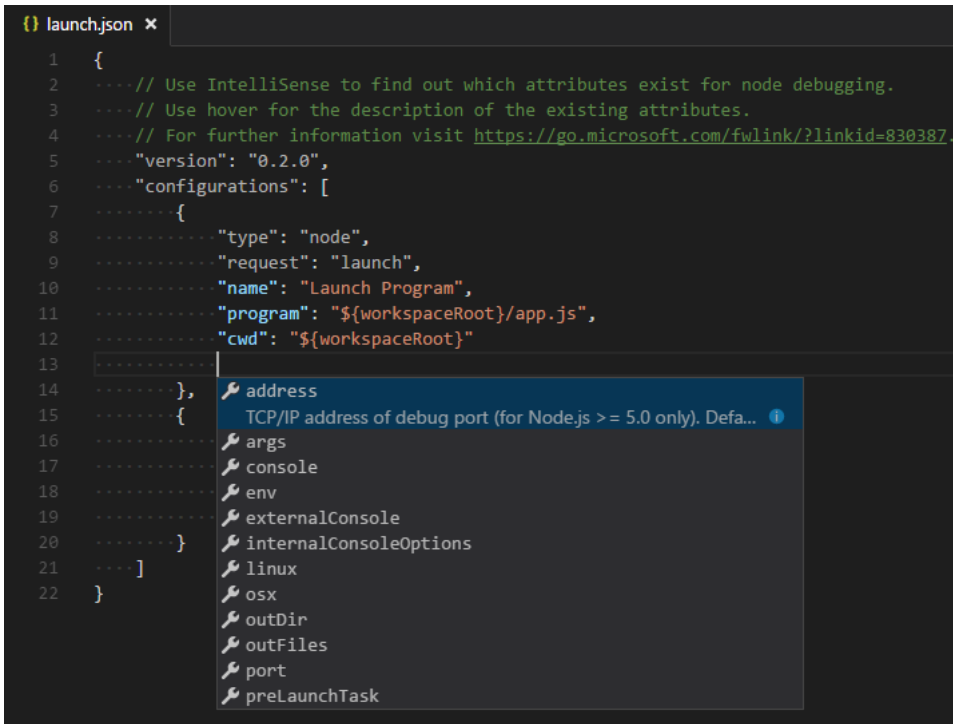


当焦点位于“变量”部分时，可以通过键入来过滤变量名称和值



Launch.json属性

有许多 launch.json 属性可帮助支持不同的调试器和调试方案。如上所述，一旦为属性指定了值，就可以使用IntelliSense（Ctrl + Space）查看可用属性的列表 type。



对于每个启动配置，必须具有以下属性：

- type -用于此启动配置的调试器的类型。每安装调试扩展引入一个类型： node 用于内置节点调试器，例如，或 php 与 go 对PHP和围棋扩展。
- request -此启动配置的请求类型。当前， launch 并且 attach 受支持。
- name -在调试启动配置下拉列表中显示的易于阅读的名称。

这是所有启动配置可用的一些可选属性：

- presentation -使用 order， group 和 hidden 在属性 presentation 可以排序，在调试配置下拉菜单，并在调试组，和隐藏的构造和化合物快速挑选对象。
- preLaunchTask -要在调试会话开始之前启动任务，请将此属性设置为task.json (/docs/editor/tasks)（在工作区的 .vscode 文件夹中）指定的任务名称。或者，可以将其设置 \${defaultBuildTask} 为使用默认的构建任务。
- postDebugTask -要在调试会话结束时启动任务，请将此属性设置为task.json (/docs/editor/tasks)（在工作区的 .vscode 文件夹中）指定的任务名称。
- internalConsoleOptions -此属性控制调试会话期间“调试控制台”面板的可见性。
- debugServer - 仅适用于调试扩展作者：此属性允许您连接到指定的端口，而不必启动调试适配器。
- serverReadyAction -如果要调试的程序在调试控制台或集成终端上输出特定消息时，要在Web浏览器中打开URL。有关详细信息，请参见下面的“调试服务器程序时自动打开URI” (https://code.visualstudio.com/docs/editor/debugging#_remote-debugging)部分。

许多调试器支持以下一些属性：

- program -启动调试器时要运行的可执行文件或文件
- args -传递给程序的参数进行调试

- `env` -环境变量（该值 `null` 可用于“取消定义”变量）
- `cwd` -当前工作目录，用于查找依赖关系和其他文件
- `port` -连接到正在运行的进程时的端口
- `stopOnEntry` -程序启动时立即中断
- `console` -什么样的控制台来使用，例如 `internalConsole`，`integratedTerminal` 或 `externalTerminal`

变量替代

VS Code可以将常用路径和其他值用作变量，并支持字符串中的变量替换 `launch.json`。这意味着您不必在调试配置中使用绝对路径。例如，`${workspaceFolder}` 给出工作区文件夹的根路径，`${file}` 在活动编辑器中打开的文件以及 `${env:Name}` 环境变量“名称”。您可以在“变量参考”中 (</docs/editor/variables-reference>)或通过在 `launch.json` 字符串属性内调用IntelliSense 来查看预定义变量的完整列表。

```
{
  "type": "node",
  "request": "launch",
  "name": "Launch Program",
  "program": "${workspaceFolder}/app.js",
  "cwd": "${workspaceFolder}",
  "args": ["${env:USERNAME}"]
}
```

平台特定的属性

`Launch.json` 支持定义值（例如，要传递给程序的参数），这些值取决于运行调试器的操作系统。为此，将平台特定的文字放入 `launch.json` 文件中，并在该文字中指定相应的属性。

下面是“args”在Windows上以不同方式传递给程序的示例：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${workspaceFolder}/node_modules/gulp/bin/gulpfile.js",
      "args": ["myFolder/path/app.js"],
      "windows": {
        "args": ["myFolder\\path\\app.js"]
      }
    }
  ]
}
```

有效的操作属性适用“windows”于Windows，“linux”Linux和“osx” macOS。在特定于操作系统的范围内定义的属性将覆盖在全局范围内定义的属性。

在下面的示例中，调试程序始终在macOS上**停止进入**：

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${workspaceFolder}/node_modules/gulp/bin/gulpfile.js",
      "stopOnEntry": true,
      "osx": {
        "stopOnEntry": false
      }
    }
  ]
}
```

全局启动配置

VS Code支持“launch”在用户设置 (</docs/getstarted/settings>)内添加对象。“launch”然后，此配置将在您的工作空间之间共享。例如：

```
"launch": {
  "version": "0.2.0",
  "configurations": [{
    "type": "node",
    "request": "launch",
    "name": "Launch Program",
    "program": "${file}"
  }]
}
```

提示：如果工作空间包含“launch.json”，则将忽略全局启动配置。

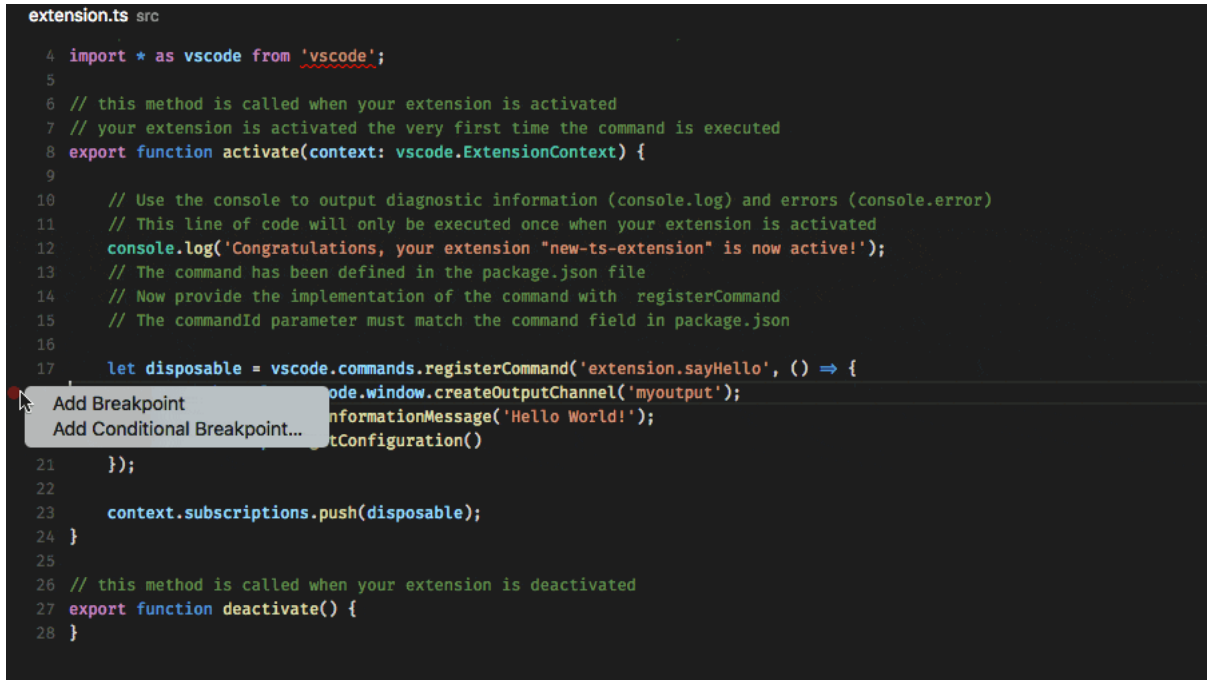
高级断点主题

条件断点

强大的VS Code调试功能是能够根据表达式，命中数或两者的组合来设置条件。

- **表达式条件**：只要表达式计算为真，断点就会被命中。
- **命中计数**：“命中计数”控制断点需要多少次才能“中断”执行。在调试器扩展中，是否尊重“命中计数”以及表达式的确切语法是不同的。

创建断点（使用“**添加条件断点**”操作）或修改现有的断点（使用“**编辑断点**”操作）时，可以添加条件和/或命中数。在这两种情况下，都会打开带有下拉菜单的嵌入式文本框，您可以在其中输入表达式：



```
extension.ts src
4 import * as vscode from 'vscode';
5
6 // this method is called when your extension is activated
7 // your extension is activated the very first time the command is executed
8 export function activate(context: vscode.ExtensionContext) {
9
10     // Use the console to output diagnostic information (console.log) and errors (console.error)
11     // This line of code will only be executed once when your extension is activated
12     console.log('Congratulations, your extension "new-ts-extension" is now active!');
13     // The command has been defined in the package.json file
14     // Now provide the implementation of the command with registerCommand
15     // The commandId parameter must match the command field in package.json
16
17     let disposable = vscode.commands.registerCommand('extension.sayHello', () => {
18         // This line of code will only be executed once when your extension is activated
19         // The command has been defined in the package.json file
20         // Now provide the implementation of the command with registerCommand
21         // The commandId parameter must match the command field in package.json
22         code.window.createOutputChannel('myoutput');
23         vscode.window.showInformationMessage('Hello World!');
24     });
25
26     context.subscriptions.push(disposable);
27 }
28
29 // this method is called when your extension is deactivated
30 export function deactivate() {}
31 }
```

如果调试器不支持条件断点，则将缺少“**添加条件断点**”操作。

内联断点

仅当执行到达与内联断点关联的列时，内联断点才会命中。当调试在一行中包含多个语句的精简代码时，这特别有用。

可以在调试会话期间使用 Shift + F9 或通过上下文菜单设置内联断点。内联断点在编辑器中内联显示。

内联断点也可以有条件。通过编辑器左边缘的上下文菜单，可以在一行上编辑多个断点。

功能断点

调试器可以通过指定函数名称来支持创建断点，而不是直接在源代码中放置断点。这在源不可用但功能名称已知的情况下很有用。

通过按BREAKPOINTS部分标题中的+按钮并输入函数名称来创建函数断点。函数断点在“BREAKPOINTS”部分显示为红色三角形。

数据断点

如果调试器支持数据断点，则可以在VARIABLES视图图中对其进行设置，并在基础变量的值更改时被命中。数据断点在“BREAKPOINTS”部分显示为红色的六边形。

调试控制台REPL

可以使用Debug Console REPL（Read-Eval-Print Loop (https://en.wikipedia.org/wiki/Read%E2%80%93Eval%E2%80%93Print_loop））功能对表达式求值。要打开调试控制台，请使用“**调试**”窗格顶部的“**调试控制台**”操作，或使用“**查看：调试控制台**”命令（Ctrl + Shift + Y）。按下Enter键后，将对表达式求值，并且在您键入时，Debug Console REPL将显示建议。如果需要输入多行，请在两行之间使用Shift + Enter，然后使用Enter发送所有行以进行评估。调试控制台输入使用活动编辑器的模式，这意味着调试控制台输入支持语法着色，缩进，自动关闭引号和其他语言功能。

```
DEBUG CONSOLE
node --debug-brk=43743 --no-lazy fib.js
Debugger listening on port 43743
89

enabled
false
7 + 8
15
range
Object
  child: Object
    endLineNumber: 8
    startColumn: 123
    startLineNumber: 7
    text: "11"
    text6: "11"
    text: "lineContext.getTokenEndIndex(tokenIndex) + 1"
    fib(15)
    987
> |
```

注意：您必须处于运行的调试会话中，才能使用Debug Console REPL。

重定向到调试目标的输入/输出

重定向输入/输出是特定于调试器/运行时的，因此VS Code并没有适用于所有调试器的内置解决方案。

您可能要考虑以下两种方法：

1. 在终端或命令提示符中手动启动程序进行调试（“调试目标”），并根据需要重定向输入/输出。确保将适当的命令行选项传递给调试目标，以便调试器可以附加到该调试目标。创建并运行一个附加到调试目标的“附加”调试配置。
2. 如果您使用的调试器扩展可以在VS Code的Integrated Terminal（或外部终端）中运行调试目标，则可以尝试将Shell重定向语法（例如“<”或“>”）作为参数传递。

这是一个示例 launch.json 配置：

```
{
  "name": "launch program that reads a file from stdin",
  "type": "node",
  "request": "launch",
  "program": "program.js",
  "console": "integratedTerminal",
  "args": ["<", "in.txt"]
}
```

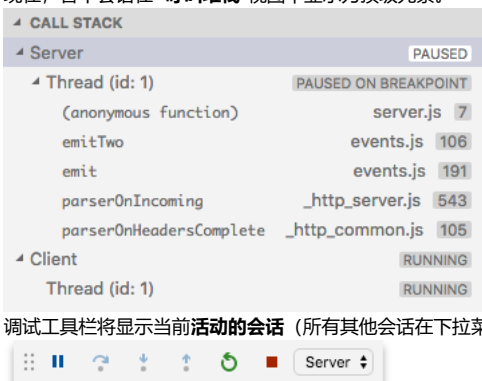
这种方法要求“<”语法通过调试器扩展名传递，并且在集成终端中未经修改就结束。

多目标调试

对于涉及多个过程（例如，客户端和服务器的）复杂方案，VS Code支持多目标调试。

使用多目标调试很简单：启动第一个调试会话后，就可以启动另一个会话。一旦第二个会话启动并运行，VS Code UI就会切换到多目标模式。

- 现在，各个会话在“呼叫堆栈”视图中显示为顶级元素。



- 调试工具栏将显示当前活动的会话（所有其他会话在下拉菜单中均可用）。

- 调试操作（例如，调试工具栏中的所有操作）在活动会话上执行。可以通过使用调试工具栏中的下拉菜单或通过选择其他元素来更改活动会话。

复合发射配置

启动多个调试会话的另一种方法是使用复合启动配置。复合启动配置列出了应并行启动的两个或多个启动配置的名称。preLaunchTask 可以选择指定一个在单个调试会话开始之前运行的a。

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Server",
      "program": "${workspaceFolder}/server.js"
    },
    {
      "type": "node",
      "request": "launch",
      "name": "Client",
      "program": "${workspaceFolder}/client.js"
    }
  ],
  "compounds": [
    {
      "name": "Server/Client",
      "configurations": ["Server", "Client"],
      "preLaunchTask": "${defaultBuildTask}"
    }
  ]
}
```

复合启动配置显示在启动配置下拉菜单中。

远程调试

VS Code本身并不支持远程调试：这是您正在使用的调试扩展的功能，您应查阅Marketplace中 (<https://marketplace.visualstudio.com/search?target=VSCode&category=Debuggers&sortBy=Downloads>)的扩展页面以获取支持和详细信息。

但是，有一个例外：VS Code中包含的Node.js调试器支持远程调试。请参阅Node.js调试 (/docs/nodejs/nodejs-debugging#_remote-debugging)主题以了解如何进行配置。

调试服务器程序时自动打开URI

开发Web程序通常需要在Web浏览器中打开特定的URL，以便在调试器中访问服务器代码。VS Code具有内置功能“**serverReadyAction**”以自动执行此任务。

这是一个简单的Node.js Express (<https://expressjs.com>)应用程序的示例：

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('Hello World!');
});

app.listen(3000, function() {
  console.log('Example app listening on port 3000!');
});
```

该应用程序首先为“/”URL安装一个“Hello World”处理程序，然后开始在端口3000上侦听HTTP连接。该端口在调试控制台中宣布，通常开发人员现在将<http://localhost:3000> 在其浏览器应用程序中键入内容。

该**serverReadyAction**特性使得它可以将结构化的属性添加 **serverReadyAction** 到任何推出的配置，并选择“动作”来进行：

```
{
  "type": "node",
  "request": "launch",
  "name": "Launch Program",
  "program": "${workspaceFolder}/app.js",

  "serverReadyAction": {
    "pattern": "listening on port ([0-9]+)",
    "uriFormat": "http://localhost:%s",
    "action": "openExternally"
  }
}
```

在这里，该 **pattern** 属性描述了用于匹配宣布端口的程序输出字符串的正则表达式。端口号的模式放在括号中，以便可以用作正则表达式捕获组。在此示例中，我们仅提取端口号，但是也可以提取完整的URI。

该 **uriFormat** 属性描述如何将端口号转换为URI。第一个 **%s** 被匹配模式的第一个捕获组替换。

然后，使用为URI方案配置的标准应用程序，在VS Code外部（“外部”）打开结果URI。

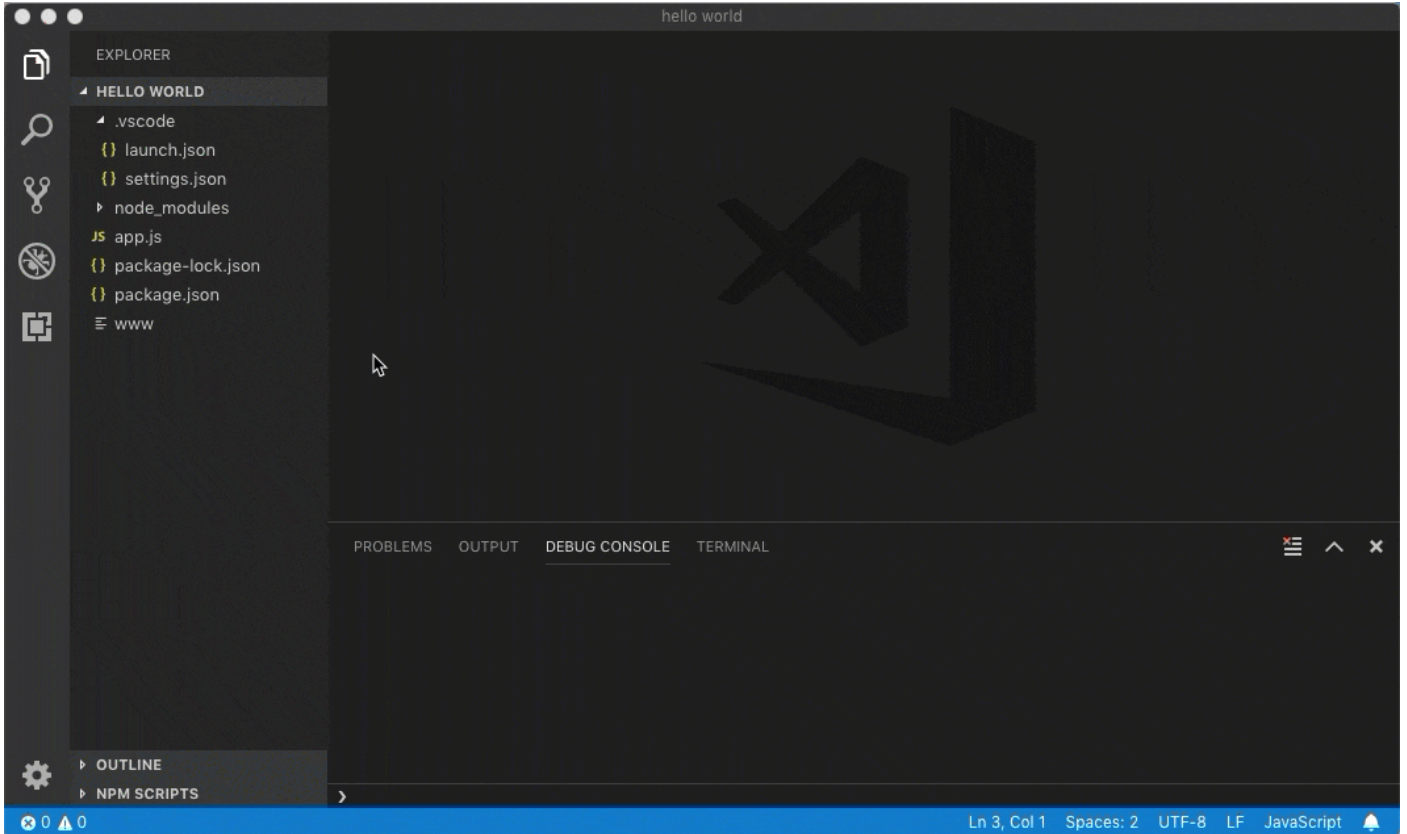
或者，**action** 可以将设置为 **debugWithChrome**。在这种情况下，VS Code为URI启动Chrome调试会话（这需要安装Debugger for Chrome (<https://marketplace.visualstudio.com/items?itemName=msjsdiag.debugger-for-chrome>)扩展程序）。在此模式下，**webRoot** 可以添加传递到Chrome调试会话的属性。

为了简化起见，大多数属性是可选的，我们使用以下后备值：

- **pattern**: "listening on.* ([https://\\S+|\[0-9\]+](https://\\S+|[0-9]+))" 与常用消息“正在侦听端口3000”或“现在正在侦听: <https://localhost:5001> (<https://localhost:5001>)”相匹配。

- `uriFormat`: "http://localhost:%s"
- `webRoot`: "\${workspaceFolder}"

这是正在运行的`serverReadyAction`功能:



下一步

要了解VS Code的Node.js调试支持, 请看一下:

- [Node.js- \(/docs/nodejs/nodejs-debugging\)](#)描述了VS Code中包含的Node.js调试器。

要查看有关Node.js调试基础知识的教程, 请观看以下视频:

- [简介视频-调试 \(/docs/introvideos/debugging\)](#) -展示调试 (/docs/introvideos/debugging)的基础知识。
- [Node.js调试入门 \(https://www.youtube.com/watch?v=2oFKNL7vYV8\)](https://www.youtube.com/watch?v=2oFKNL7vYV8) -显示如何将调试器附加到正在运行的Node.js进程。

要了解VS Code的任务运行支持, 请访问:

- [任务 \(/docs/editor/tasks\)](#) -描述如何使用Gulp, Grunt和Jake运行任务, 以及如何显示错误和警告。

要编写自己的调试器扩展, 请访问:

- [调试器扩展 \(/api/extension-guides/debugger-extension\)](#) -使用模拟样本来说明创建VS Code调试扩展所需的步骤。

常见问题

受支持的调试方案是什么?

Linux, macOS和Windows均支持使用VS Code开箱即用调试基于Node.js的应用程序。市场中提供的VS Code扩展 (<https://marketplace.visualstudio.com/vscode/Debuggers?sortBy=Downloads>)支持许多其他方案。

我在“运行”视图下拉列表中没有看到任何启动配置。怎么了?

最常见的问题是未设置 `launch.json` 文件或该文件中存在语法错误。另外, 您可能需要打开一个文件夹, 因为无文件夹调试不支持启动配置。

该文档对您有帮助吗?

☐ 是 ☐ 没有