

# 与调试器交互

您可以使用Qt Creator**调试模式**在调试时检查应用程序的状态。可以通过多种方法与调试器交互，其中包括：

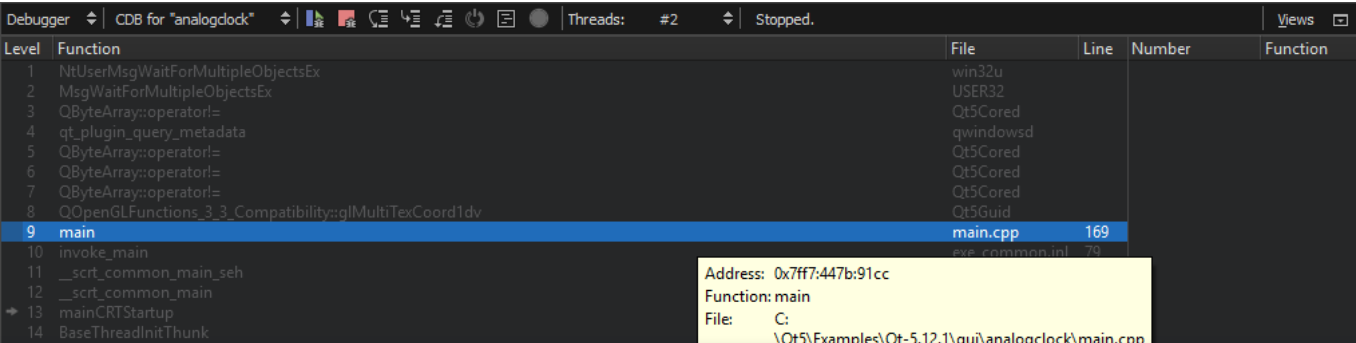
- › 逐行或逐条指令地完成程序。
- › 中断正在运行的程序。
- › 设置断点。
- › 检查调用堆栈的内容。
- › 检查和修改局部变量和全局变量的内容。
- › 检查和修改调试程序的寄存器和内存内容。
- › 检查已加载的共享库的列表。
- › 反汇编代码段。

Qt Creator以清晰简洁的方式显示本机调试器提供的原始信息，目的是在不失去本机调试器功能的情况下尽可能简化调试过程。

除了堆栈视图、局部变量和表达式视图、寄存器等提供的通用IDE功能外，Qt Creator还包括使调试基于Qt的应用程序变得容易的功能。调试器插件了解几个Qt类的内部布局，例如**QString**，Qt容器，最重要的是**QObject**（以及从中派生的类），以及C++标准库的大多数容器和一些GCC扩展。这种更深入的理解用于以有用的方式呈现此类类的对象。

## 使用调试器

在调试模式下，可以使用多个视图与正在**调试**的程序进行交互。视图的可用性取决于您是调试 C++ 还是 QML。默认情况下显示常用视图，隐藏很少使用的视图。若要更改默认设置，请选择“**视图>视图**”，然后选择要显示或隐藏的视图。或者，可以从任何可见调试器视图的标题栏的上下文菜单中启用或禁用视图。



您可以将Qt Creator中的视图拖放到屏幕上的新位置。视图的大小和位置将保存以供将来的会话使用。选择“**视图>视图>重置为默认布局**”，将视图重置为其原始大小和位置。

若要节省屏幕上的空间，请选择“**视图>视图>自动隐藏视图标题栏**”。

若要显示和隐藏视图中的列，请在上下文菜单中切换“**显示列**”。

一旦程序开始在调试器的控制下运行，它的行为和执行将照常进行。可以通过选择“**调试>中断**”来中断正在运行的C++程序。命中断点时，程序会自动中断。

程序停止后，Qt Creator：

- › 检索表示程序当前位置的调用堆栈的数据。
- › 检索局部变量的内容。
- › 检查**表达式**。
- › 更新**寄存器**、**模块**和**反汇编器**视图（如果要调试基于 C++ 的应用程序）。

可以使用**调试**模式视图更详细地检查数据。

您可以使用以下键盘快捷键：

- › 若要完成调试，请按Shift+F5。
- › 若要完整地执行一行代码，请按F10（在 macOS 上按 Command+Shift+O）。
- › 若要单步执行函数或子函数，请按F11（在 macOS 上按 Command+Shift+I）。
- › 要离开当前函数或子函数，请按 Shift+F11（在 macOS 上按 Command+Shift+T）。
- › 若要继续运行该程序，请按F5。
- › 要运行到包含光标的行，请按Ctrl+F10（在 macOS 上按 Shift+F8）。
- › 若要在单步执行嵌套函数时运行到所选函数，请按Ctrl+F6。

也可以继续执行程序，直到当前函数完成或跳转到当前函数中的任意位置。

## 单步执行代码

使用 GDB 作为调试后端时，可以将多个步骤压缩为一个步骤，以减少调试的干扰。有关详细信息，请参阅[指定 GDB 设置](#)。

扩展的 GDB 设置提供了在代码中向后退步的选项，但应谨慎使用此选项，因为它在 GDB 端速度慢且不稳定。有关详细信息，请参阅[指定扩展 GDB 设置](#)。

## 自定义调试视图

您可以通过在**首选项>调试器**中指定设置来更改调试视图的外观和行为。例如，您可以：

- › 在调试视图中使用交替的行颜色。
- › 采用主编辑器中的字体大小更改。
- › 在调试时在主编辑器中显示工具提示。
- › 关闭临时源和内存视图，并在调试器退出时切换到以前使用的Qt Creator模式。
- › 当调试的应用程序中断时，将Qt Creator置于前台。


# 设置断点


可以将断点与以下项相关联：

- › 源代码文件和行
- › 功能
- › 地址
- › 引发和捕获异常
- › 执行和分叉进程
- › 执行某些系统调用
- › 程序运行时特定地址的内存块中的更改
- › 发射 QML 信号
- › 抛出 JavaScript 异常

在某些条件下，可以限制断点对程序的中断。

断点有两种类型：和。未认领的断点表示中断调试程序的任务，并在以后将控制权传递给您。它有两种状态：和。unclaimedclaimedpendingimplanted

未声明的断点存储为会话的一部分，并且独立于程序是否正在调试而存在。当它们引用代码中的位置时，它们会在“断点**预设**”视图和编辑器中使用（**未认领的断点**）图标列出。

Breakpoint Preset							
Debuggee	Function	File	Line	Address	Condition	Ignore	Threads
	-	...\quickcontrols2\gallery\gallery.qml	155				(all)


当调试器启动时，调试后端从可能由调试程序处理的一组未认领断点中识别断点，并声明这些断点以供自己独占使用。声明的断点列在正在运行的调试器的“**断点**”视图中。此视图仅在调试器运行时存在。

当调试器声明断点时，未声明的断点将从“断点**预设**”视图中消失，在“断点”视图中显示为挂起**断点**。

在不同时间，会尝试将挂起的断点植入调试的进程。成功完全植入可能会创建一个或多个植入断点，每个断点都与调试断点中的实际地址相关联。例如，植入还可能将编辑器中的断点标记从空行移动到为其生成实际代码的下一行。植入的断点图标没有沙漏覆盖。

调试器结束时，其声明的断点（挂起和植入）将返回到未声明状态，并重新出现在“**断点预设**”视图中。

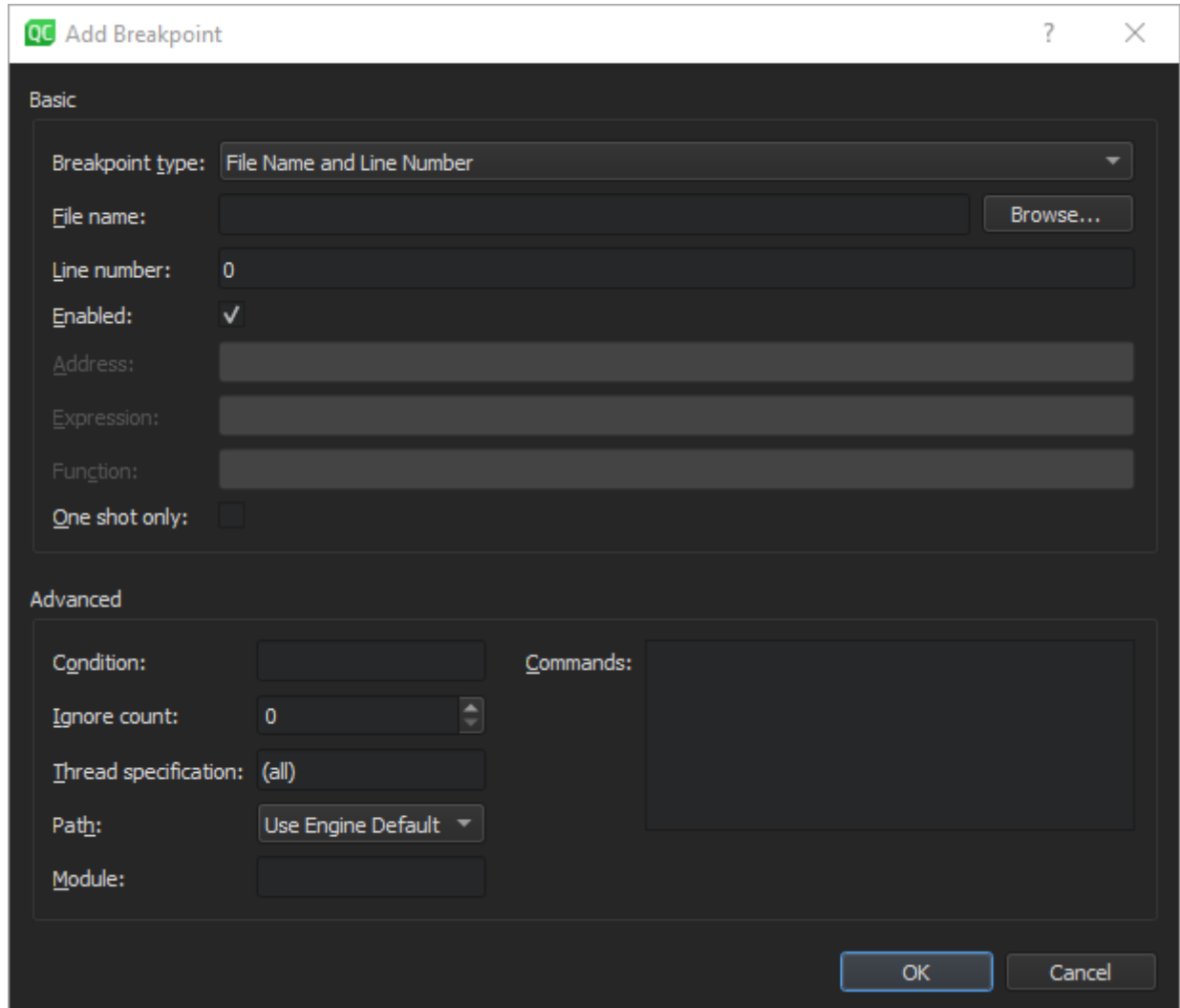
如果在执行调试的程序期间命中植入的断点，控制权将传递回给您。然后，您可以检查中断程序的状态，或者逐行或连续继续执行。

Number	Function	File	Line	Address	Condition	Ignore	Threads
 2	-	f:\dd\vc\tools\crt\vcstartup\src\startup\exe_main.cpp	17	0x7ff7447b9e3d			(all)

## 添加断点

- › 在代码编辑器中，单击左边距或在您希望程序停止的特定行上按F9（在 macOS 上为 F8）。
- › 在“断点预设”视图或“断点”视图中：
  - › 双击视图的空白部分。
  - › 右键单击视图，然后在上下文菜单中选择“添加断点”。

2. 在“断点类型”字段中，选择程序代码中希望程序停止的位置。要指定的其他选项取决于所选位置。



3. 在“条件”字段中，如果**条件**的计算结果为 true，则设置在断点处停止之前要评估的条件。
4. 在“忽略”字段中，指定在程序停止之前忽略断点的次数。
5. 在“命令”字段中，指定程序停止时要执行的**命令**；一行上的一个命令。GDB 按指定的顺序执行命令。

## 指定断点设置

可以在“**编辑**>**首选项**”>“**调试器**”中指定断点的设置。有关更多信息，请参见[指定调试器设置](#)。

若要在断点中使用完整的绝对路径，请选中“**使用完整的绝对路径设置断点**”复选框。

GDB 和 CDB 允许在未生成代码的源代码行上设置断点。在这种情况下，断点将移动到实际生成代码的下一个源代码行。若要通过在源代码编辑器中移动断点标记来反映此类临时更改，请选择“GDB>**调整断点位置**”或“CDB>**更正断点位置**”。

使用 GDB 作为后端时，可以使用 Python 扩展普通的 GDB 断点类。选择“GDB>**使用伪邮件跟踪点**”。

可以在某些函数上自动添加断点以捕获错误和警告消息。有关详细信息，请参阅[指定 CDB 设置](#)和[指定扩展 GDB 设置](#)。

有关断点的详细信息，请参阅 GDB 文档中的[断点](#)、[观察点](#)和[捕获点](#)。

## 移动断点

移动断点：

- › 将断点标记拖放到文本编辑器中的另一行。
- › 在“断点预设”视图或“断点”视图中，选择“**编辑所选断点**”，然后在“行号”字段中设置行号。

## 删除断点

删除断点：

- › 单击文本编辑器中的断点标记。
- › 在“断点预设”视图或“**断点**”视图中：
  - › 选择断点，然后按 **Delete** 键。
  - › 在上下文菜单中选择“删除选定的断点”、“删除**选定的断点**”或“删除文件的断点”。

## 启用和禁用断点

要暂时禁用断点而不删除断点并丢失条件和命令等关联数据，请执行以下操作：

- › 右键单击文本编辑器中的断点标记，然后选择“**禁用断点**”。
- › 选择包含断点的行，然后按 **Ctrl+F9**（在 macOS 上按 **Ctrl+F8**）。
- › 在“断点预设”视图或“**断点**”视图中：
  - › 选择断点，然后按 **空格键**。
  - › 在上下文菜单中选择“**禁用断点**”。

文本编辑器和视图中的空心断点图标表示禁用的断点。若要重新启用断点，请使用上述任一方法。

除了数据断点之外，断点在重新启动调试的程序时将保持其启用或禁用状态。

## 设置数据断点

在指定地址读取或写入数据时，**数据断点**将停止程序。

在地址处设置数据断点：

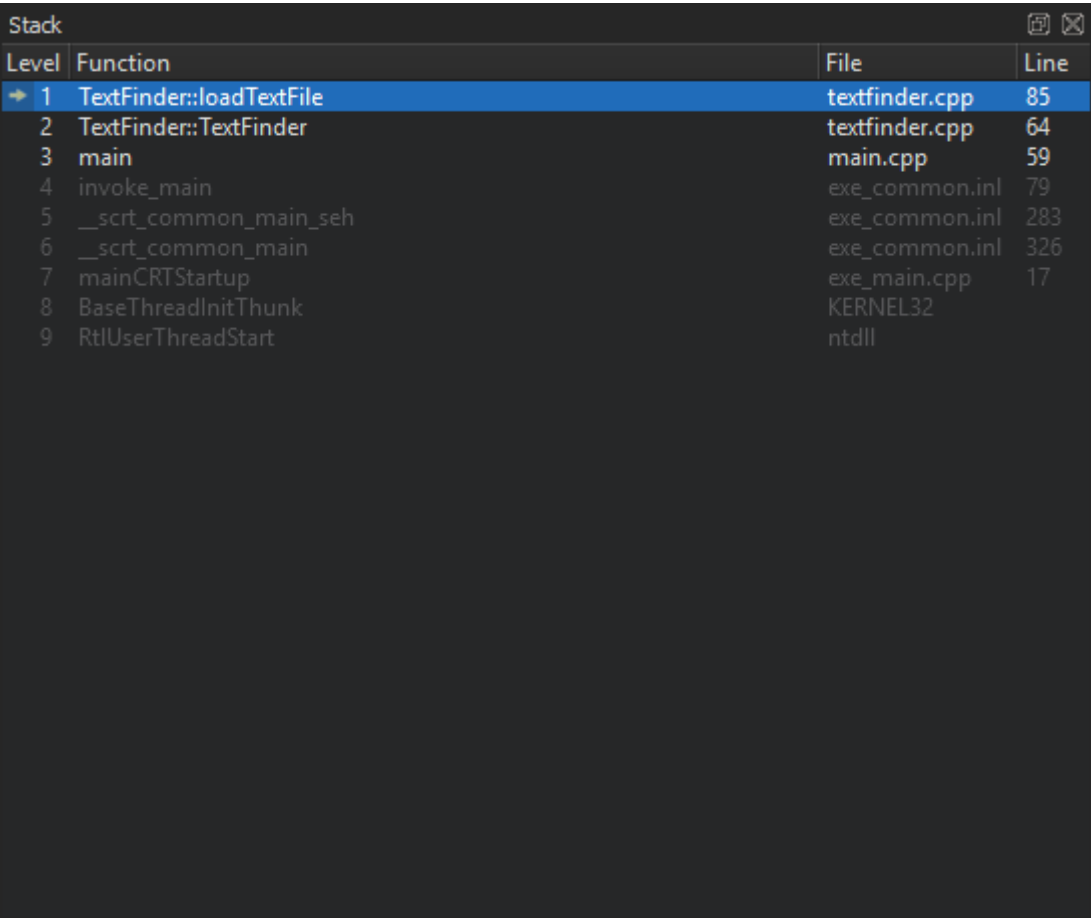
1. 在“断点预设”或“断点”视图中，选择上下文菜单中的“**添加断点**”。
2. 在**断点类型**字段中，选择**在固定地址访问数据时中断**。
3. 在地址字段中，指定内存块的**地址**。
4. 选择“**确定**”。

如果地址显示在“**局部变量**”或“**表达式**”视图中，则可以在上下文菜单中选择“**在对象的地址处添加数据断点**”以设置数据断点。

当调试的程序退出时，数据断点将被禁用，因为它们的地址不太可能在下次程序启动时保持不变。如果确实需要，

## 查看调用堆栈跟踪

当正在调试的程序中断时，Qt Creator 会将指向当前位置的嵌套函数调用显示为调用堆栈跟踪。此堆栈跟踪是从调用堆栈帧构建的，每个调用堆栈帧表示一个特定的函数。对于每个函数，Qt Creator 都会尝试检索相应源文件的文件名和行号。此数据显示在**堆栈**视图中。



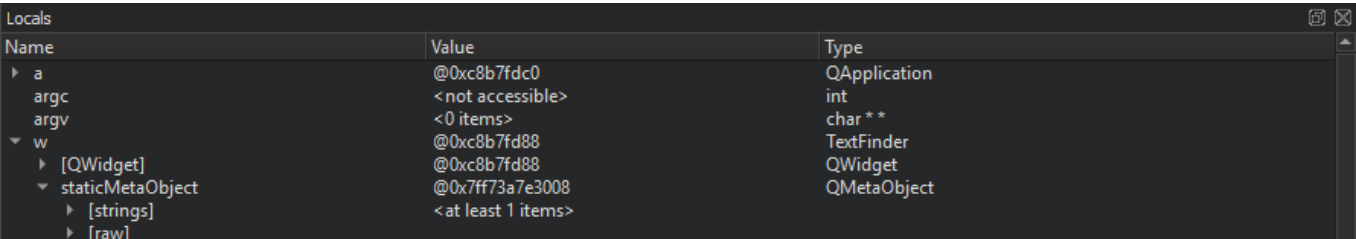
由于导致当前位置的调用堆栈可能产生或经过没有调试信息的代码，因此并非所有堆栈帧都具有相应的源位置。没有相应源位置的堆栈帧在**“堆栈”**视图中灰显。

如果单击具有已知源位置的帧，文本编辑器将跳转到相应的位置并更新**“局部变量”**和**“表达式”**视图，使程序在进入函数之前似乎被中断了。

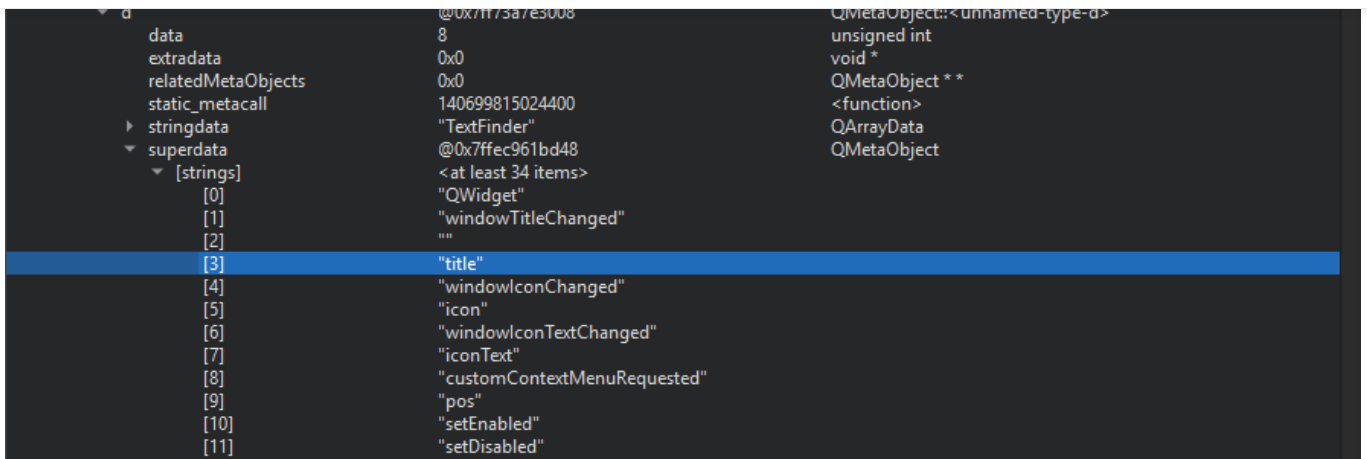
要找出哪个 QML 文件导致 Qt Quick 2 应用程序崩溃，请在堆栈视图的上下文菜单中选择**加载 QML 堆栈**。调试器尝试从停止的可执行文件中检索 JavaScript 堆栈，并将帧附加到 C++ 帧（如果找到任何帧）。您可以单击 QML 堆栈中的帧以在编辑器中打开 QML 文件。

## 局部变量和函数参数

“局部变量”视图由**“局部变量”**窗格和**“返回值”**窗格（空时隐藏）组成。





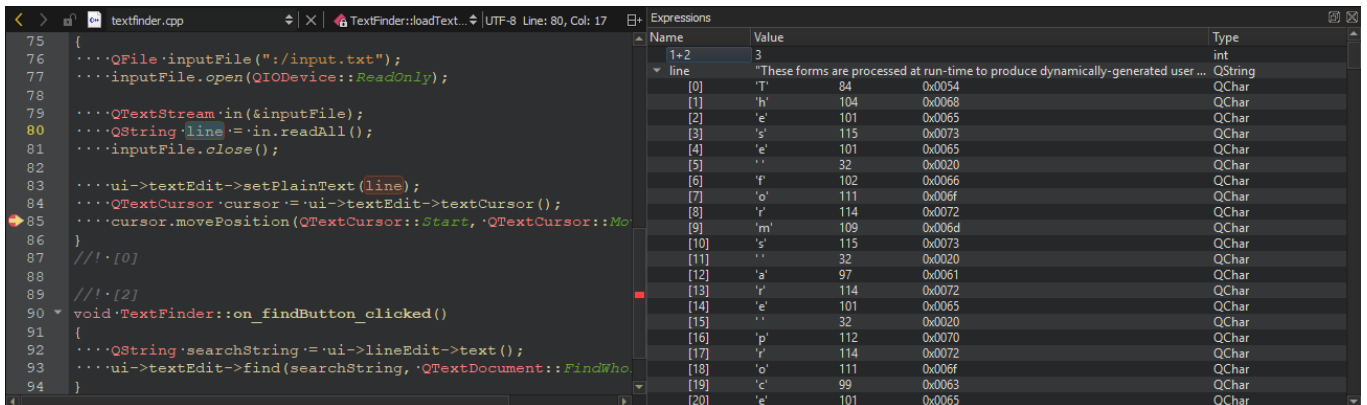


每当程序在调试器的控制下停止时，它都会检索有关最顶层堆栈帧的信息，并将其显示在“**局部变量**”视图中。“**局部变量**”窗格显示有关该帧中函数的参数以及局部变量的信息。如果调试器中的最后一个操作是在按 Shift+F11 后从函数返回的，则“**返回值**”窗格将显示函数返回的值。

使用 GDB 时，可以指定是显示对象的动态类型还是静态类型。在上下文菜单中选择**使用动态对象类型进行显示**。请记住，选择动态类型可能会较慢。

## 计算表达式

要计算算术表达式或函数调用的值，请使用表达式视图中的**表达式**计算器。要插入新的表达式计算器，请双击“**表达式**”或“**局部变量**”视图的空白部分，或者从上下文菜单中选择“**添加新表达式计算器**”，或者从代码编辑器中拖放**表达式**。



**注意：**表达式计算器功能强大，但会显著减慢调试器操作的速度。建议不要过度使用它们，并尽快删除不需要的表达式计算器。

每当前帧更改时，都会重新计算表达式计算器。请注意，每次都会调用表达式中使用的函数，即使它们有副作用。

QML 调试器可以评估 JavaScript 表达式。

GDB、LLDB 和 CDB 支持简单 C 和 C++ 表达式的计算。仅当函数实际编译到调试的可执行文件或可执行文件使用的库中时，才能调用函数。最值得注意的是，内联函数（如标准容器的大多数实现）通常不可用。

operator[]

使用 GDB 或 LLDB 作为后端时，可以使用特殊的范围语法通过一个表达式显示多个值。formis 的子表达式被拆分为一系列单独计算的表达式。foo[a..b]foo[a], ..., foo[b]

**注意：**

GDB和LLDB，以及Qt Creator的调试器，也适用于Linux和macOS上的优化构建。优化可能会导致指令重新排序或删除某些局部变量，从而导致“**局部变量**和**表达式**”视图显示意外数据。

GCC 提供的调试信息不包含有关变量初始化时间的足够信息。因此，Qt Creator无法判断局部变量的内容是否包含“真实数据”或“初始噪声”。如果QObject显示为未初始化，则其值报告为**不在范围内**。但是，并非所有未初始化的对象都可以被识别为这样。

**注意：** 计算的表达式集将保存在会话中。

## 检查基本Qt对象

“**局部变量**”和“**表达式**”视图还提供了对调试器最强大功能的访问：全面显示属于Qt基本对象的数据。例如，在QObject的情况下，您看到的不是指向某个私有数据结构的指针，而是子项、信号和插槽的列表。

同样，Qt Creator的调试器不显示许多指针和整数，而是有序地显示QHash或QMap的内容。此外，调试器显示QFileInfo的访问数据，并提供对QVariant 真实内容的访问。

在“**局部变量**”或“**表达式**”视图中单击鼠标右键以打开上下文菜单，该菜单提供了用于查看数据的其他选项。可用选项取决于当前项的类型，由**调试帮助程序**提供。通常，类似字符串的数据（如 and）提供多种编码，以及使用完整编辑器窗口的可能性。类似映射的数据（如、和）提供了一个紧凑的选项，使用键的列，从而简洁地显示带有短键（如数字或短字符串）的容器。例如，若要展开QMap的所有值，请选择“**更改值显示格式>压缩**”。QByteArraystd::stringQMapQHashstd::mapname

可以使用“**局部变量**和**表达式**”视图更改简单数据类型的变量的内容，例如,,,以及程序中断时。为此，请单击“值”列，使用就地编辑器修改值，然后按Enter（或Return）。intfloatQStringstd::string

要更改QVector或值的完整内容，请在主条目的“值”列中键入所有以逗号分隔的值。std::vector

您可以在显示此信息的主编辑器中启用工具提示。有关详细信息，请参阅**调试时在工具提示中查看变量的值**。

## 直接与本机调试器交互

在某些情况下，直接与本机调试器的命令行交互很方便。在Qt Creator中，您可以使用**调试器日志**视图的左窗格来实现此目的。按Ctrl+Enter时，文本光标下的行的内容将直接发送到本机调试器。或者，您可以使用视图底部的线条编辑。输出显示在“**调试器日志**”视图的右窗格中。

**注意：** 通常，您不需要此功能，因为Qt Creator为您提供了更好的方法来处理任务。例如，可以在“**表达式**”视图中计算表达式，而不是从命令行使用GDB命令。print

## 调试基于C++的应用程序

以下各节介绍仅适用于调试C++的其他调试函数。

### 从命令行启动调试器

您可以从命令行使用Qt Creator 调试器界面。若要将其附加到正在运行的进程，请将进程ID 指定为选项的参数。要检查核心文件，请指定文件名。Qt Creator执行所有必要的步骤，例如搜索属于核心文件的二进制文件。



```
> C:\qtcreator\bin>qtcreator -debug 2000
> C:\qtcreator\bin>qtcreator -debug core=core.2000
> C:\qtcreator\bin>qtcreator -debug some.exe,core=core
> C:\qtcreator\bin>qtcreator -debug server=some.dot.com:4251
```

有关更多信息，请参见[使用命令行选项](#)。

## 在 macOS 中逐步进入框架

在 macOS 中，外部库通常内置于所谓的框架中，其中可能包含库的发布版本和调试版本。在 macOS 桌面上运行应用程序时，默认情况下使用框架的发行版本。若要单步执行框架，请在项目运行设置中选择“**使用框架的调试版本**”选项。

## 查看线程

如果多线程程序中断，则可以使用“线程”视图或调试器状态栏中名为“**线程**”的组合框从一个线程切换到另一个**线程**。**堆栈**视图会相应地自行调整。

## 查看模块

“模块”视图显示调试器插件包含的有关正在调试的应用程序中包含的**模块**的信息。模块是 Windows 中的动态链接库（），Linux 中的共享对象（）和 macOS 中的动态共享库（）。.dll.so.dylib

此外，该视图显示模块中的符号，并指示每个模块的加载位置。

右键单击视图以打开上下文菜单，其中包含以下菜单项：

- > 更新模块列表
- > 加载模块的符号
- > 检查模块
- > 编辑模块文件
- > 在模块中显示符号
- > 显示模块之间的依赖关系（仅限 Windows）

默认情况下，“**模块**”视图处于隐藏状态。

使用 CDB 作为调试后端时，可以指定在加载或卸载应用程序模块时调试器应中断。若要为指定的模块启用中断，请选择“**编辑>首选项>调试器>CDB**”。有关详细信息，请参阅[指定 CDB 设置](#)。

## 查看源文件

“源文件”视图列出了项目中包含的所有**源文件**。如果无法单步执行指令，则可以检查源文件是否实际上是项目的一部分，或者它是否在其他地方编译。该视图显示文件系统中每个文件的路径。

右键单击视图以打开上下文菜单，其中包含用于重新加载数据和打开文件的菜单项。

若要使调试器能够在与生成库的位置不同的位置使用源代码树的副本时单步执行代码并显示源代码，可以将源路径映射到目标路径。有关更多信息，请参阅[映射源路径](#)。

默认情况下，“**源文件**”视图处于隐藏状态。

“反汇编程序”视图对于用于检查单个指令（如单步执行和单步执行）的低级命令非常有用。默认情况下，“反汇编程序”视图处于隐藏状态。

若要访问“反汇编程序”视图，请在调试器运行时选中“调试”>“按指令操作”。或者，单击调试器工具栏上的（按指令操作）工具按钮。

默认情况下，GDB 显示 AT&T 样式反汇编。要切换到英特尔样式，请选择编辑>首选项>调试器>GDB>使用英特尔样式反汇编。

### 查看和编辑寄存器状态

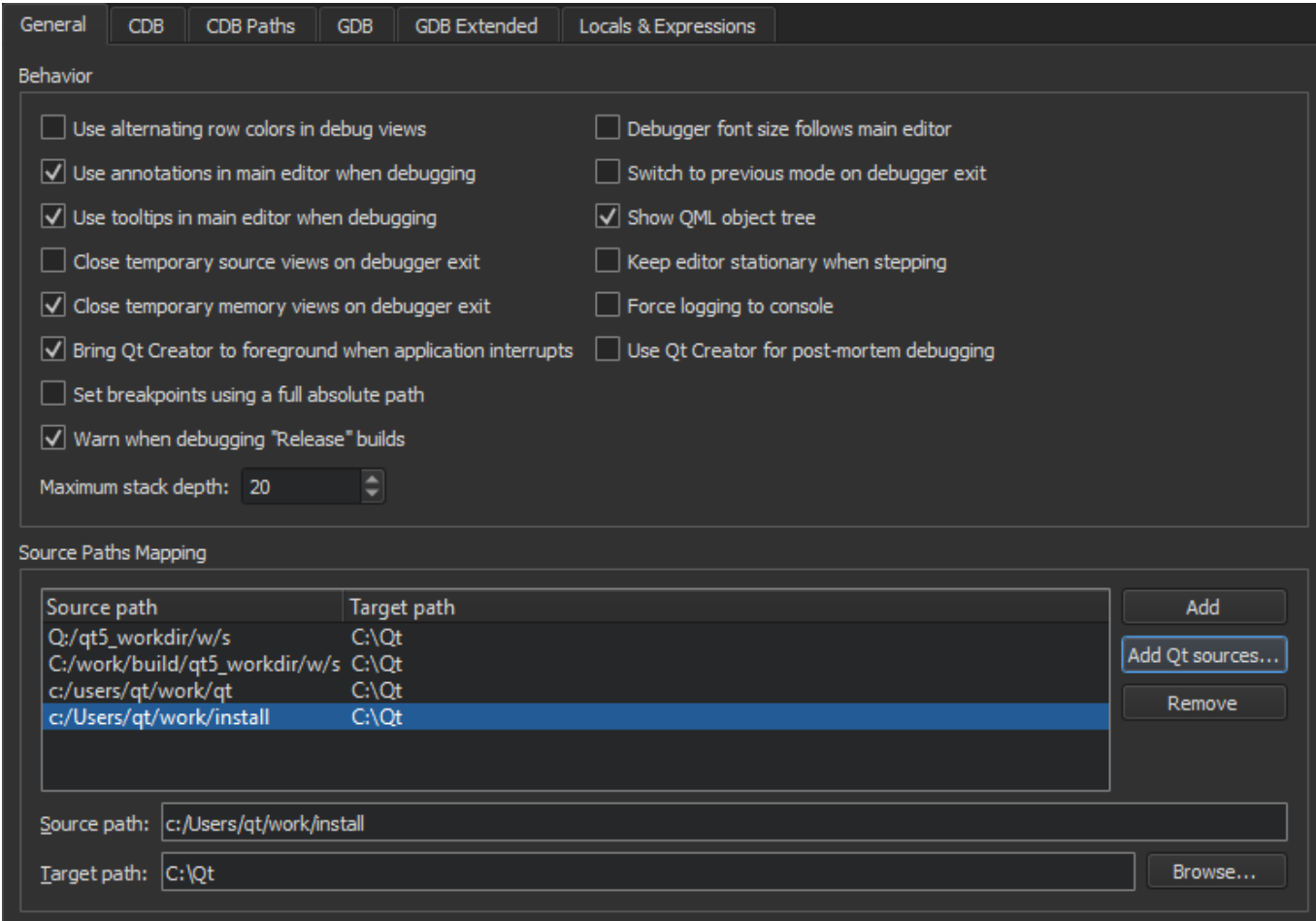
“寄存器”视图显示 CPU 寄存器的当前状态。根据 CPU 类型，将有不同的寄存器可用。最近更改的寄存器值以红色突出显示，空寄存器值以及前导零显示为灰色。

此外，可以在程序停止时编辑寄存器的内容。这适用于通用和特殊用途寄存器。寄存器可以在标准精简视图中编辑，或者如果寄存器显示为展开，则可以在其特定部分进行编辑。

默认情况下，“寄存器”视图处于隐藏状态。

## 指定调试器设置

若要指定用于管理调试器进程的设置，请选择“编辑>首选项>调试器”。在“常规”选项卡中，可以指定所有调试器通用的设置。



可以自定义调试视图的外观和行为，指定断点的设置，以及将源路径映射到目标路径。

## 映射源路径

若要使调试器能够在与生成库的位置不同的位置使用源代码树的副本时单步执行代码并显示源代码，可以将源路径映射到目标路径。

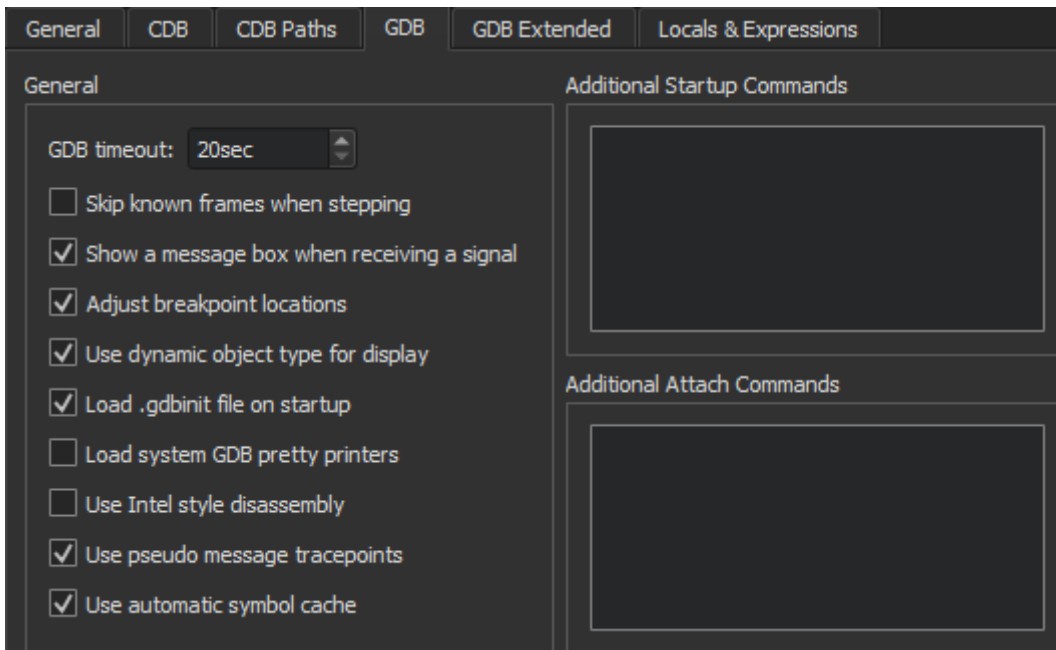
要自动将源路径映射到尚未修补的Qt版本，请选择添加Qt源并浏览到Qt源文件的位置。

手动将源路径映射到目标路径：

1. 在“**源路径映射**”中，选择“**添加**”以将条目添加到路径列表。
2. 在“**源路径**”字段中，指定调试器报告的可执行文件的调试信息中的**源路径**。
3. 在“**目标路径**”字段中，指定源树在本地计算机上的实际位置。

## 指定 GDB 设置

若要指定用于管理 GDB 进程的设置，请选择“**编辑>首选项>调试器>GDB**”。



要指定终止无响应的 GDB 进程的超时，请在**GDB 超时**字段中设置等待的秒数。对于大多数应用程序来说，默认值 20 秒应该足够了，但如果加载大型库或列出源文件所需的时间比在慢速计算机上花费的时间长得多，则应增加该值。

若要将多个步骤压缩为一个步骤，以便在单步执行代码时减少干扰调试，请选中“**单步执行时跳过已知帧**”复选框。例如，跳过原子参考计数代码，信号发射的单步进入最终直接进入与其连接的插槽中。

若要在调试期间应用程序收到信号（如 SIGSEGV）后立即显示消息框，请选中“**接收信号时显示消息框**”复选框。

GDB 允许在未生成代码的源代码行上设置断点。在这种情况下，断点将移动到实际生成代码的下一个源代码行。若要通过在源代码编辑器中移动断点标记来反映此类临时更改，请选中“**调整断点位置**”复选框。

若要指定是显示对象的动态类型还是静态类型，请选中“**使用动态对象类型进行显示**”复选框。请记住，选择动态类型可能会较慢。

若要允许在调试器启动时读取用户的默认 .gdbinit 文件，请选中“**启动时加载 .gdbinit 文件**”复选框。

若要使用系统中安装的默认 GDB 漂亮打印机或链接到应用程序使用的库的默认 GDB 漂亮打印机，请选中“**加载**

若要让 GDB 自动将其符号索引的副本保存在磁盘上的缓存中，并在将来加载同一二进制文件时从该缓存中检索该副本，请选中“**使用自动符号缓存**”复选框。

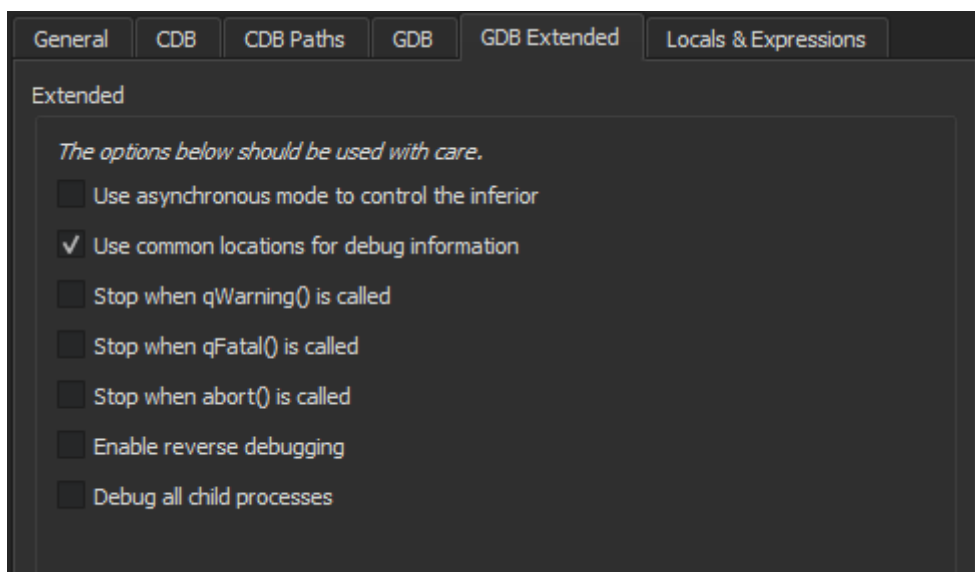
要在启动 GDB 之后、启动或附加调试程序之前以及初始化调试帮助程序之前执行 GDB 命令，请在“**其他启动命令**”字段中输入这些命令。

要在 GDB 成功连接到远程目标后执行 GDB 命令，请在“**其他连接命令**”字段中输入这些命令。您可以在此处添加命令以进一步设置目标，例如 `asor. monitor resetload`

要执行简单的 Python 命令，请为它们添加前缀。要执行跨多行的 Python 命令序列，请在单独的行前面加上块，并在单独的行后附加。要执行任意 Python 脚本，请使用。 `pythonpythonendpython`  
`execfile('/path/to/script.py')`

## 指定扩展 GDB 设置

若要指定 GDB 的扩展设置，请选择“**编辑>首选项>调试器>GDB 扩展**”。这些设置允许访问 GDB 的高级或实验功能。启用它们可能会对调试体验产生负面影响，因此请谨慎使用它们。



要使用异步模式控制下级，请选中相应的复选框。

若要将公共路径添加到调试信息的位置（例如，在启动 GDB 时），请选中“**使用调试信息的常用位置**”复选框。 `/usr/src/debug`

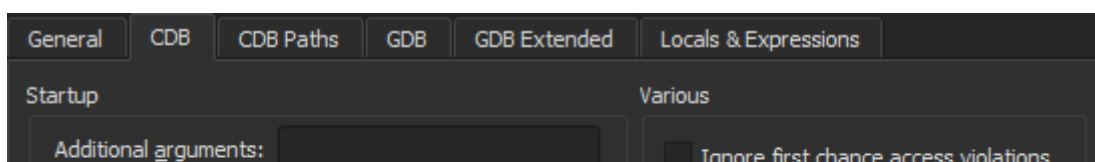
要在调用 `oris` 时停止，请选中相应的复选框。 `qWarningqFatalabort`

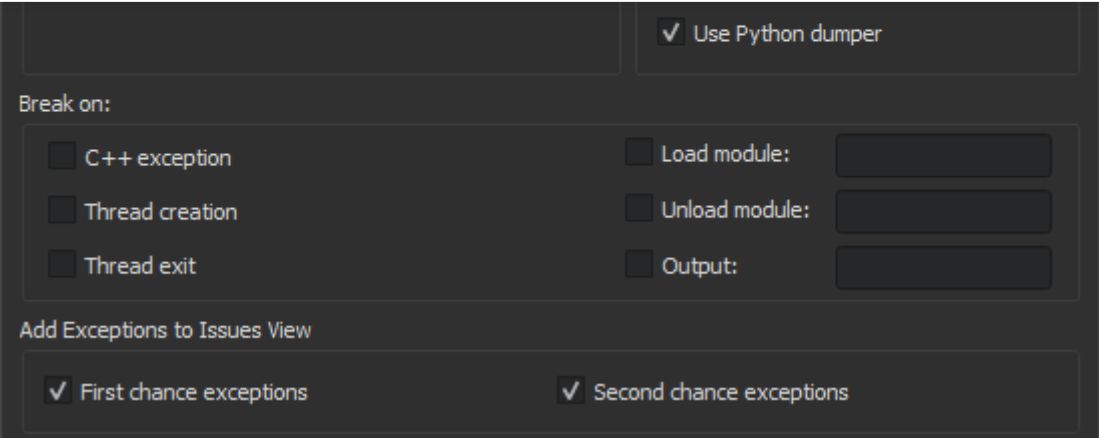
若要启用后退，请选中“**启用反向调试**”复选框。此功能在 GDB 端非常缓慢且不稳定。在系统调用上倒退时，它会表现出不可预测的行为，并且很可能会破坏调试会话。

若要在分叉后继续调试所有子进程，请选中“**调试所有子进程**”复选框。

## 指定 CDB 设置

若要指定用于管理 CDB 进程的设置，请选择“**编辑>首选项>调试器>CDB**”。





您可以在**其他参数**字段中指定用于启动 CDB 的其他参数。

如果控制台应用程序在配置的控制台中无法正常运行，并且后续连接失败，则可以使用 CDB 的本机控制台诊断。  
[Qt创建者手册8.0.2](#)  
Topics >

若要在函数上自动添加断点，请选中“**调用 CrtCbgrReport () 时停止**”复选框。例如，这会捕获由以下原因引起的运行时错误消息。CrtCbgrReport()assert()

在“中断日期”组中，指定调试器是否应在C++异常、线程创建或退出、加载或卸载指定的**应用程序模块**时或指定的输出上**中断**。

若要禁用访问冲突异常时的首次机会中断，请选中“**忽略首次机会访问冲突**”复选框。第二次出现访问冲突将中断到调试器中。

CDB 允许在注释中或未生成代码的源代码行上设置断点。在这种情况下，断点将移动到实际生成代码的下一个源代码行。若要通过在源代码编辑器中移动断点标记来反映此类临时更改，请选中“**更正断点位置**”复选框。有关详细信息，请参阅[设置断点](#)。

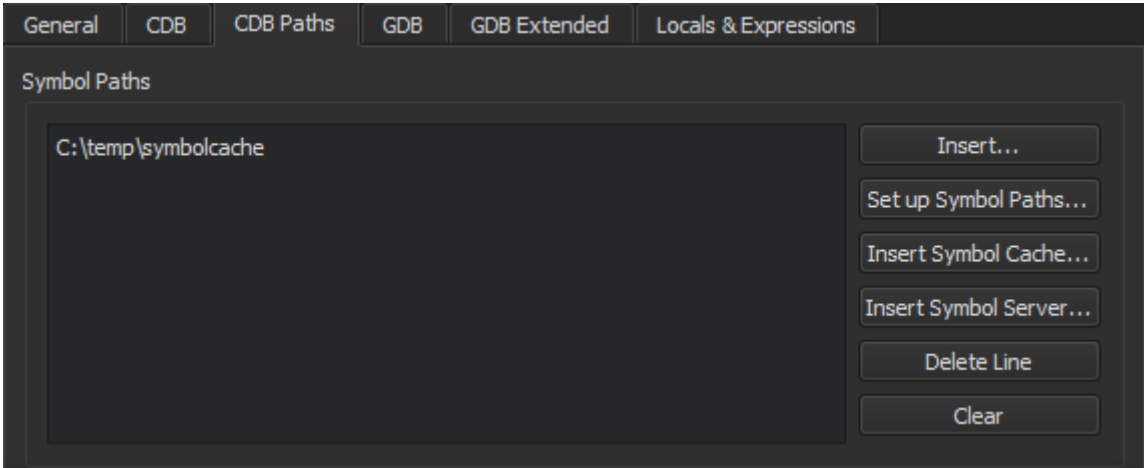
若要使用 Python 转储程序类提供的抽象层创建“**局部变量**”和“**表达式**”视图中显示的数据项的说明，请选中“**使用 Python 转储程序**”复选框。有关更多信息，请参见[调试帮助程序实现](#)。

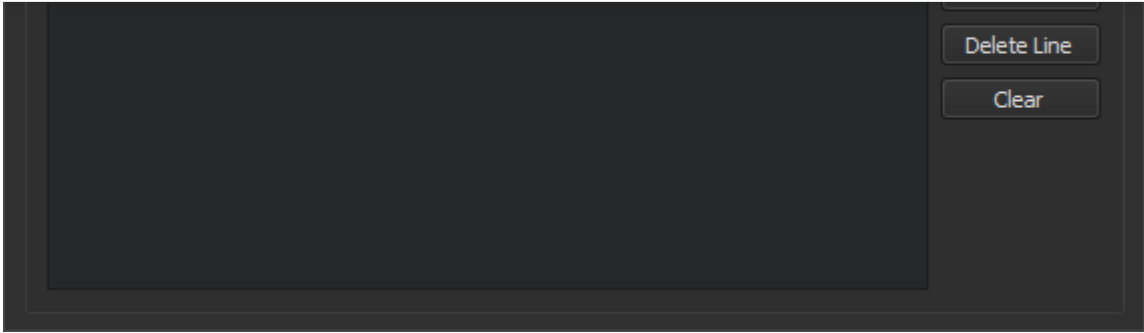
若要在“问题”中显示有关首次机会和第二次机会例外的信息，请选中“**向问题视图添加例外**”组中的复选框。

## 在 Windows 上设置 CDB 路径

若要获取用于调试 Windows 应用程序的操作系统库的调试信息，请将 Microsoft 提供的符号服务器添加到调试器的符号搜索路径中：

1. 选择“**编辑>首选项>调试器>CDB 路径**”。





2. 在“符号路径”组中，选择“插入”。
3. 选择要存储缓存信息的目录。使用临时目录中的子文件夹，例如。C:\temp\symbolcache
4. 选择“确定”。

注意：在网络连接速度较慢的情况下，填充缓存可能需要很长时间。

若要使用源服务器基础结构直接从版本控制或 Web 提取缺少的源文件，请在“源路径”字段中输入以下字符串：  
srv\*

< 启动调试器

使用调试帮助程序 >

©2022 Qt有限公司 此处包含的文档贡献的版权归 他们各自的所有者。此处提供的文档根据自由软件基金会发布的GNU 自由文档许可证版本 1.3的条款进行许可。Qt和相应的徽标是Qt有限公司在芬兰和/或全球其他国家的商标。所有其他商 标均为财产 其各自所有者。



联系我们

公司

- 关于我们
- 投资者
- 编辑部
- 职业
- 办公地点

发牌

- 条款和条件
- 开源
- 常见问题

支持

- 支持服务
- 专业服务

对于客户

- 支持中心
- 下载





训练

客户成功案例

社区

为Qt做贡献

论坛

维基

下载

市场

© 2022 Qt公司

[反馈](#) [登录](#)