

Q 搜索

使用调试帮助程序

结构化数据（如对象、或类型）作为树的一部分显示在“**局部变量**”和“**表达式**”视图中。要访问对象的子结构，请展开树节点。子结构按其内存中的顺序显示，除非选择了上下文菜单中的“**按字母顺序对类和结构的成员进行排序**”选项。classstructunion

同样，指针显示为树项，其中单个子项表示指针的目标。如果选择了上下文菜单项“**自动取消引用指针**”，则指针和目标将合并为一个条目，显示指针的名称和类型以及目标的值。

这种标准表示对于检查简单结构来说已经足够好了，但它通常不能对更复杂的结构（例如或关联容器）提供足够的洞察力。这些项目在内部由指针的复杂排列表示，通常经过高度优化，部分数据既不能通过子结构也不能通过指针直接访问。QObjects

为了让用户也能轻松访问这些项目，Qt Creator使用了称为**调试助手**的Python脚本。调试帮助程序始终自动使用。若要强制显示结构的纯 C 类显示，请选择“**编辑>首选项>调试器>局部变量和表达式**”，然后取消选中“**使用调试帮助程序**”复选框。这仍将使用 Python 脚本，但会生成更多基本输出。要强制对单个对象或给定类型的所有对象进行普通显示，请从上下文菜单中选择相应的选项。

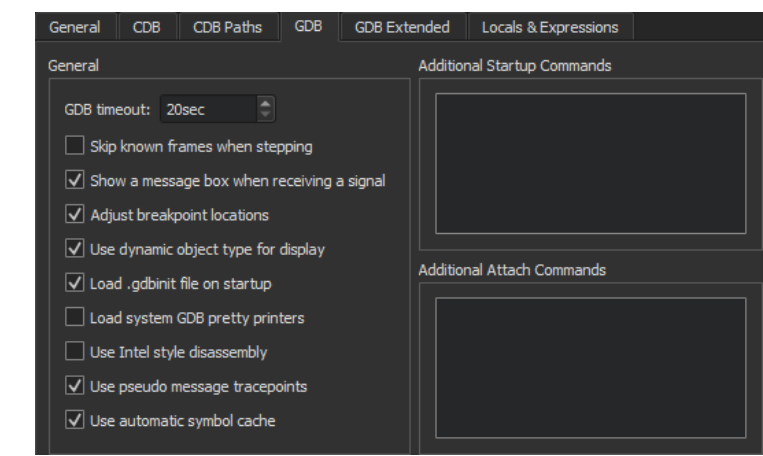
Qt Creator附带了200多个最流行的Qt类的调试帮助程序，标准C++容器和智能指针，涵盖了开箱即用的C++应用程序开发人员的常规需求。

扩展调试帮助程序

Qt Creator使用Python脚本将原始内存内容和类型信息数据从本机调试器后端（目前支持GDB，LLDB和CDB）转换为在“**局部变量**”和“**表达式**”视图中呈现给用户的形式。

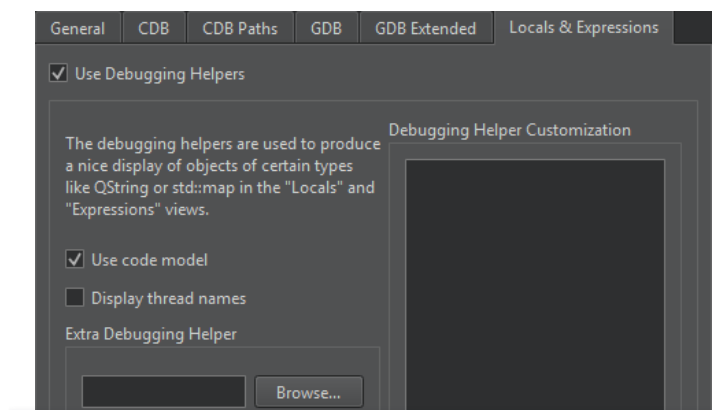
与GDB漂亮的**打印机**和LLDB的数据**格式化程序**不同，Qt Creator的调试助手独立于本机调试后端。也就是说，相同的代码可以与 Linux 上的 GDB、macOS 上的 LLDB 和 Windows 上的 CDB 一起使用，或者任何其他平台上至少有一个受支持的后端可用。

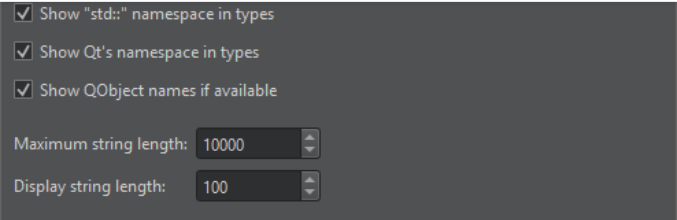
若要使用系统中安装的默认 GDB 漂亮打印机或链接到应用程序使用的库的默认 GDB 漂亮打印机，请选择**首选项>调试器>GDB>加载系统 GDB 漂亮打印机**。有关详细信息，请参阅**指定 GDB 设置**。



自定义内置调试帮助程序

可以在加载并完全初始化内置调试帮助程序后执行命令。要加载其他调试帮助程序或修改现有调试帮助程序，请选择**Edit>Preferences>Debugger>Locals & Expressions**，然后在“**调试帮助程序自定义**”字段中输入命令。





如果在使用 GDB 时收到有关接收信号的错误消息，则可以指定**GDB 命令**来处理信号。例如，您可以告诉 GDB 忽略信号，如果您收到以下错误消息：SIGSTOPThe inferior stopped because it received a signal from the operating system. Signal name: SIGSTOP

若要阻止 GDB 处理信号，请将以下命令添加到“**调试帮助程序自定义**”字段：SIGSTOP

```
handle SIGSTOP nopass
handle SIGSTOP nostop
```

若要在调试期间应用程序收到信号后立即显示消息框，请选择“**编辑>首选项**”>“**调试器>GDB>接收信号时显示消息框**”。

添加自定义调试帮助程序

要为自己的类型添加调试帮助程序，不需要编译，只需添加几行 Python。这些脚本可以同时处理多个版本的Qt或您自己的库。

若要为自定义类型添加调试帮助程序，请将调试帮助程序实现添加到本机调试器的启动文件（例如，或），或直接在Edit>Preferences>Debugger>GDB 中的“**其他启动命令**”中指定它们。~/.gdbinit~/lldbinit

要开始为您自己的数据类型实现调试帮助程序，您可以将它们的实现放入Qt安装或独立Qt Creator安装中的文件中。在macOS上，该文件被捆绑到Qt Creator应用程序包中，它位于文件夹中。share/qtcreator/debugger/personaltypes.pyContents/resources/debugger

该文件包含一个示例实现：personaltypes.py

```
# def qdump__MapNode(d, value):
#     d.putValue("This is the value column contents")
#     d.putExpandable()
#     if d.isExpanded():
#         with Children(d):
#             # Compact simple case.
#             d.putSubItem("key", value["key"])
#             # Same effect, with more customization possibilities.
#             with SubItem(d, "data")
#                 d.putItem("data", value["data"])
```

添加调试帮助程序：

1. 打开文件进行编辑。例如，如果您的Qt安装位于Windows上的目录中，请查看。在 macOS 上，查看。
share/qtcreator/debugger/personaltypes.pyQt5C:\Qt5\Tools\QtCreator\share\qtcreator\debuggerQt5\Qt Creator.app/Contents/resources/debugger
2. 将转储程序实现添加到文件末尾。有关实现调试帮助程序的详细信息，请参阅以下部分。personaltypes.py
3. 为了防止在更新Qt Creator安装时（例如，在更新Qt安装时）被覆盖，请将其复制到文件系统中Qt Creator安装之外的安全位置，并在 Edit>Preferences>Debugger>Locals & Expressions>Extra Debugging Helper中指定该位置。personaltypes.py

当您在Qt Creator中启动调试会话或从**调试器日志**视图的上下文菜单中选择重新**加载调试助手**时，将自动选取自定义调试帮助程序。personaltypes.py

调试帮助程序概述

调试帮助程序的实现通常由单个 Python 函数组成，该函数需要命名，其中是要检查的类或类模板。请注意，示波器分辨率运算符替换为双下划线：。嵌套命名空间是可能的。模板参数不用于函数名称的构造。qdump__NS__FooNS::Foo::__

例子：

- › 为类型实现调试帮助程序的函数的名称是。namespace Project { template<typename T> struct Foo {... } }qdump__Project__Foo
- › 为类型实现调试帮助程序的函数的名称是。std::__1::vector<T>::iteratorqdump__std____1__vector__iterator

Qt Creator的调试器插件会在您想要显示此类对象时调用此函数。该函数传递以下参数：

- › d类型，一个包含当前设置的对象，并提供用于构建表示“**局部变量**”和“**表达式**”视图一部分的对象的工具。Dumper
- › value类型，包装GDB。值或lldb。SBValue.Value

该函数必须向 Dumper 对象提供某些信息，这些信息用于在“**局部变量**”和“**表达式**”视图中构建对象及其子项的显示。qdump__*

例：

```
def qdump__QFiniteStack(d, value):
    alloc = value["_alloc"].integer()
```

```
d.putArrayData(value["_array"], size, value.type[0])
```

注意：要创建可与 LLDB 和 GDB 后端一起使用的转储程序函数，请避免直接访问 `and`命名空间，而是改用类的函数。`gdb.*lldb.*Dumper`

若要在调试帮助程序中获取对象的基本实例，请使用函数或以下示例代码：`value.base()`

```
def qdump__A(d, value):
    t = value.members(True)[0].type
    dptr, base_v = value.split('p{%s}' % t.name)
    d.putItem(base_v)
```

调试帮助程序可以设置为在类型名称与正则表达式匹配时调用。为此，调试帮助程序的函数名称必须以（带有两个下划线字符）开头。此外，该函数需要具有第三个参数，该参数使用默认值调用，该值指定类型名称应匹配的正则表达式。`qdump__regex`

例如，Nim 0.12 编译器为其编译的所有泛型序列分配人工名称（如 `and`）。要在 Qt Creator 中可视化这些内容，可以使用以下调试助手：`TY1TY2`

```
def qdump__NimGenericSequence__(d, value, regex = "^TY.*$"):
    size = value["Sup"]["len"]
    base = value["data"].dereference()
    typeobj = base.dereference().type
    d.putArrayData(base, size, typeobj)
```

调试帮助程序实现

调试帮助程序以类似于 GDB/MI 和 JSON 的格式创建所显示数据项的说明。

对于“**局部变量**”和“**表达式**”视图中的每一行，需要创建如下所示的字符串并将其引导到调试器插件。

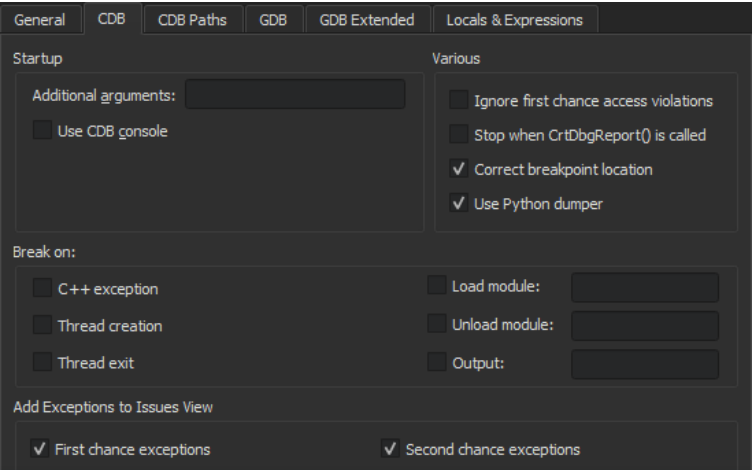
```
{ iname='some internal name',          # optional
  address='object address in memory',  # optional
  name='contents of the name column',  # optional
  value='contents of the value column',
  type='contents of the type column',
  numchild='number of children',       # zero/nonzero is sufficient
  children=[                          # only needed if item is expanded in view
    {iname='internal name of first child',
      },
    {iname='internal name of second child',
      },
  ],
}
```

字段的值是对象的内部名称，它由以点分隔的标识符列表组成，对应于对象在视图中的表示位置。如果不存在，则通过连接父对象的点和序列号来生成它。
`inameiname`

字段的值显示在视图的“**名称**”列中。如果未指定，则使用括号中的简单数字。`name`

由于格式不能保证稳定，因此强烈建议不要直接生成连线格式，而是使用 Python Dumper 类提供的抽象层，特别是类本身和抽象。这些提供了一个完整的框架来处理 `theandfields`，处理简单类型、引用、指针、枚举以及已知和未知结构的子项，以及一些处理常见情况的方便函数。`DumperDumper:ValueDumper:Typeinameaddr`

使用 CDB 作为调试器后端时，可以通过选择“**编辑>首选项>调试器>CDB>使用 Python 转储程序**”来启用 Python 转储程序。



该类包含通用簿记、低级和便利函数。Dumper

类的成员函数如下: Dumper

- › putItem(self, value)- 直接处理基本类型、引用、指针和枚举的主函数，迭代复合类型的基类和类成员，并在适当时调用函数。qdump__*
- › putIntItem(self, name, value)- 相当于:

```
with SubItem(self, name):
    self.putValue(value)
    self.putType("int")
```

- › putBoolItem(self, name, value)- 相当于:

```
with SubItem(self, name):
    self.putValue(value)
    self.putType("bool")
```

- › putCallItem(self, name, value, func, *args)- 使用本机调试器后端将函数调用放在value指定的值上，并输出结果项。func

本机调用非常强大，例如，可以利用调试过程中的现有调试或日志记录工具。但是，由于以下原因，它们只能在受控环境中使用，并且仅在没有其他方法访问数据时使用：

- › 直接执行代码是危险的。它以调试进程的权限运行本机代码，不仅可能损坏调试的进程，还可能访问磁盘和网络。
- › 检查核心文件时无法执行调用。
- › 在调试器中设置和执行调用的成本很高。

- › putArrayData(self, address, itemCount, type)- 创建由位于的类数组对象的类型指定的子项数。itemCounttypeaddress
- › putSubItem(self, component, value)- 相当于:

```
with SubItem(self, component):
    self.putItem(value)
```

捕获嵌套函数调用引发的异常，并将生成的所有输出替换为以下输出: putItem

```
except RuntimeError:
    d.put('value="<invalid>",type="<unknown>",numchild="0",')
```

- › put(self, value)- 直接追加到输出字符串的低级函数。这也是追加输出的最快方法。
- › putField(self, name, value)- 附加字段。name='value'
- › childRange(self)- 返回当前作用域中指定的子项范围。Children
- › putItemCount(self, count)- 将字段追加到输出。value='< %d items>'
- › putName(self, name)- 附加字段。name=''
- › putType(self, type, priority=0)- 追加字段，除非类型与父项的默认子类型一致，或者已为具有更高值的当前项调用。type=''putTypepriority
- › putBetterType(self, type)- 覆盖上次记录的内容。type
- › putExpandable(self)- 宣布当前值存在子项。默认值为无子项。
- › putNumChild(self, numchild)- 宣布当前值的子项存在 (>0) 或不存在。numchild
- › putValue(self, value, encoding = None)- 附加文件，可以选择后跟字段。需要可转换为完全由字母数字值组成的字符串。该参数可用于指定编码，以防必须以某种方式对实际值进行编码以满足仅字母数字要求。参数要么是 formwhere of any,,,, 的字符串，要么是 给出对象的基本组件的大小（如果它不是隐含的），并且指定值是否应该在显示中用引号括起来。
value='valueencoding='valueencodingencodingcodec:itemsize:quotequoteclatin1utf8utf16ucs4intfloatitemsizecodequote'

例:

```
# Safe transport of quirky data. Put quotes around the result.
d.putValue(d.hexencode("ABC\DEF"), "utf8:1:1")
```

- › putStringValue(self, value)- 对QString进行编码并使用正确的设置进行调用。putValueencoding
- › putByteArrayValue(self, value)- 对QByteArray进行编码，并使用正确的设置进行调用。putValueencoding
- › isExpanded(self)- 检查当前项目是否在视图中展开。
- › createType(self, pattern, size = None)- 创建一个对象。确切的操作取决于。Dumper.Typepattern
- › 如果匹配已知类型的名称，则返回描述此类型的对象。patternDumper.Type

- › q- 有符号 8 字节整数值
- › Q- 无符号8字节整数值
- › i- 有符号 4 字节整数值
- › I- 无符号4字节整数值
- › h- 有符号 2 字节整数值
- › H- 无符号2字节整数值
- › b- 有符号 1 字节整数值
- › B- 无符号 1 字节整数值
- › d- 8 字节 IEEE 754 浮点值
- › f- 4 字节 IEEE 754 浮点值
- › p- 指针，即根据目标架构大小合适的无符号整数值
- › @- 合适的填充。大小由前面和后面的字段以及目标体系结构决定
- › <n>s- <n> 字节的 blob，隐含对齐为 1
- › <typename>- 由名称确定的合适大小和合适对齐方式的斑点Dumper.Typetypename

自卸车类型类

Theclass 描述一段数据的类型，通常是C++类或结构、指向结构的指针或基元类型，如整型或浮点型。Dumper.Type

类型对象（即类的实例）可以由本机调试器后端创建，通常是通过计算内置到调试的二进制文件中或随调试的二进制文件一起提供的调试信息，或者由调试帮助程序动态创建。Dumper.Type

Qt Creator使用这种可能性为大多数Qt类即时提供类型信息，消除了使用Qt的调试版本进行对象自省的需要。

类的成员函数如下：Dumper.Type

- › name- 此类型的字符串名称，或者类型是匿名的。None
- › size(self)- 返回此类型的对象的大小（以字节为单位）。
- › bitsize(self)- 返回此类型的对象的大小（以位为单位）。
- › alignment(self)- 返回此类型对象所需的对齐方式（以字节为单位）。
- › deference(self)- 返回指针类型的取消引用类型，否则返回。None

Qt创建者手册8.0.2
Topics >

- › stripTypedefs(self)- 如果此类型是别名，则返回基础类型。
- › templateArgument(self, position, numeric = False)- 返回位于模板化类型的模板参数。Ifis 将参数作为整数值返回。positionnumericTrue
- › fields(self)- 返回描述此类型的基类和数据成员的列表。Dumper:Fields

自卸车场类

类描述类型对象的基类或数据成员。Dumper.Field

该类的成员函数和属性如下：Dumper.Field

- › isBaseClass- 区分基类和数据成员。
- › fieldType(self)- 返回此基类或数据成员的类型。
- › parentType(self)- 返回所属类型。
- › bitsize(self)- 返回此字段的大小（以位为单位）。
- › bitpos(self)- 返回此字段在拥有类型中的偏移量（以位为单位）。

Dumper.Value Class

类描述一段数据，例如C++类或基元数据类型的实例。它还可用于描述在内存中没有直接表示形式的人工项目，例如文件内容、非连续对象或集合。Dumper.Value

阿哈斯一直是一个关联。值的实际数据的两个主要表示形式是：Dumper.ValueDumper.Type

- › 遵循 Python 缓冲区协议的 Python 对象，例如 Python 或对象。应与此值类型的大小匹配。memoryviewbytesize()
- › 一个整数值，表示指向当前地址空间中对象的开头的指针。对象的大小由其类型给出。size()

为 ais 创建调试器帮助程序时，通常不需要了解 ais 的内部表示形式。Dumper.Value

该类的成员函数和属性如下：Dumper.Value

- › integer(self)- 将此值的解释返回为合适大小的有符号整数值。
- › pointer(self)- 将此值作为当前地址空间中的指针返回的解释。
- › members(self, includeBases)- 返回表示此值的基本对象和数据成员的对象列表。Dumper.Value
- › dereference(self)- 对于描述指针的值，返回取消引用的值，否则返回。None
- › cast(self, type)- 返回与此值具有相同数据但类型的值。type
- › address(self)- 如果此值由当前地址空间中的连续区域组成，则返回该值的地址，否则返回此值。None

› `dynamicTypeName(self)`- 如果这是基类对象，则尝试检索此值的动态类型的名称。返回如果不可能。None

子项和子项类

如果数据未初始化或损坏，尝试创建子项可能会导致错误。要在这种情况下正常恢复，请使用上下文 *管理器* 创建嵌套项。ChildrenSubItem

构造函数使用一个强制参数和几个可选参数。强制参数引用当前对象。可选参数可用于指定子项数，每个子项具有类型和孙项。指定 `lfis` 时，仅显示多个子项。在倾倒容器内容物时应使用此方法，否则可能需要很长时间。参数和可用于减少儿童倾倒地产生的数据量。如果第 `n` 个子项的地址等于 `addrBase + n * addrStep`，则将禁止打印第 `n` 个子项的地址打印。`Children__init__(self, dumper, numChild = 1, childType = None, childNumChild = None, maxNumChild = None, addrBase = None, addrStep = None)DumpernumChildchildType_childNumChild_maxNumChildaddrBaseaddrStep`

例：



```
if d.isExpanded():
    with Children(d):
        with SubItem(d):
            d.putName("key")
            d.putItem(key)
        with SubItem(d):
            d.putName("value")
            d.putItem(value)
```

请注意，这可以更方便地编写为：

```
d.putNumChild(2)
if d.isExpanded():
    with Children(d):
        d.putSubItem("key", key)
        d.putSubItem("value", value)
```



The Qt Company



联系我们

公司

关于我们

投资者

编辑部

职业

办公地点

支持

支持服务

专业服务

合作伙伴

训练

社区

为Qt做贡献

论坛

维基

下载

发牌

条款和条件

开源

常见问题

对于客户

支持中心

下载

Qt登录

联系我们

客户成功案例

