(1条消息)GDB查看变量 - junecau的专栏 - CSDN博客

在你调试程序时，当程序被停住时，你可以使用print命令（简写命令为p），或是同义命令inspect来查看当前程序的运行数据。print命令的格式是：
print
print /
是表达式，是你所调试的程序的语言的表达式（GDB可以调试多种编程语言），是输出的格式，比如，如果要把表达式按16进制的格式输出，那么就是/x。

一、表达式
print和许多GDB的命令一样，可以接受一个表达式，GDB会根据当前的程序运行的数据来计算这个表达式，既然是表达式，那么就可以是当前程序运行中的const常量、变量、函数等内容。可惜的是GDB不能使用你在程序中所定义的宏。
表达式的语法应该是当前所调试的语言的语法，由于C/C++是一种大众型的语言，所以，本文中的例子都是关于C/C++的。（而关于用GDB调试其它语言的章节，我将在后面介绍）
在表达式中，有几种GDB所支持的操作符，它们可以用在任何一种语言中。
@
是一个和数组有关的操作符，在后面会有更详细的说明。
::
指定一个在文件或是一个函数中的变量。
{}
表示一个指向内存地址的类型为type的一个对象。

二、程序变量
在GDB中，你可以随时查看以下三种变量的值：
1、全局变量（所有文件可见的）
2、静态全局变量（当前文件可见的）
3、局部变量（当前Scope可见的）
如果你的局部变量和全局变量发生冲突（也就是重名），一般情况下是局部变量会隐藏全局变量，也就是说，如果一个全局变量和一个函数中的局部变量同名时，如果当前停止点在函数中，用print显示出的变量的值会是函数中的局部变量的值。如果此时你想查看全局变量的值时，你可以使用“::”操作符：
file::variable
function::variable
可以通过这种形式指定你所想查看的变量，是哪个文件中的或是哪个函数中的。例如，查看文件f2.c中的全局变量x的值：

gdb) p 'f2.c'::x
当然，“::”操作符会和C++中的发生冲突，GDB能自动识别“::” 是否C++的操作符，所以你不必担心在调试C++程序时会出现异常。
另外，需要注意的是，如果你的程序编译时开启了优化选项，那么在用GDB调试被优化过的程序时，可能会发生某些变量不能访问，或是取值错误码的情况。这个是很正常的，因为优化程序会删改你的程序，整理你程序的语句顺序，剔除一些无意义的变量等，所以在GDB调试这种程序时，运行时的指令和你所编写指令就有不一样，也就会出现你所想象不到的结果。对付这种情况时，需要在编译程序时关闭编译优化。一般来说，几乎所有的编译器都支持编译优化的开关，例如，GNU 的C/C++编译器GCC，你可以使用“-gstabs”选项来解决这个问题。关于编译器的参数，还请查看编译器的使用说明文档。

三、数组
有时候，你需要查看一段连续的内存空间的值。比如数组的一段，或是动态分配的数据的大小。你可以使用GDB的“@”操作符，“@”的左边是第一个内存的地址的值，“@”的右边则你你想查看内存的长度。例如，你的程序中有这样的语句：
int *array = (int *) malloc (len * sizeof (int));
于是，在GDB调试过程中，你可以以如下命令显示出这个动态数组的取值：
p *array@len
@的左边是数组的首地址的值，也就是变量array所指向的内容，右边则是数据的长度，其保存在变量len中，其输出结果，大约是下面这个样子的：
(gdb) p *array@len
$1 = {2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40}
如果是静态数组的话，可以直接用print数组名，就可以显示数组中所有数据的内容了。

四、输出格式
一般来说，GDB会根据变量的类型输出变量的值。但你也可以自定义GDB的输出的格式。例如，你想输出一个整数的十六进制，或是二进制来查看这个整型变量的中的位的情况。要做到这样，你可以使用GDB的数据显示格式：
x 按十六进制格式显示变量。
d 按十进制格式显示变量。
u 按十六进制格式显示无符号整型。
o 按八进制格式显示变量。
t 按二进制格式显示变量。
a 按十六进制格式显示变量。
c 按字符格式显示变量。
f 按浮点数格式显示变量。
(gdb) p i
$21 = 101
(gdb) p/a i
$22 = 0x65

```
(gdb) p/c i
$23 = 101 'e'
(gdb) p/f i
$24 = 1.41531145e-43
(gdb) p/x i
$25 = 0x65
(gdb) p/t i
$26 = 1100101
```

五、查看内存

你可以使用examine命令（简写是x）来查看内存地址中的值。x命令的语法如下所示：

x/

n、f、u是可选的参数。

n 是一个正整数，表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。

f 表示显示的格式，参见上面。如果地址所指的是字符串，那么格式可以是s，如果地十是指令地址，那么格式可以是i。

u 表示从当前地址往后请求的字节数，如果不指定的话，GDB默认是4个bytes。u参数可以用下面的字符来代替，b表示单字节，h表示双字节，w表示四字节，g表示八字节。当我们指定了字节长度后，GDB会从指内存定的内存地址开始，读写指定字节，并把其当作一个值取出来。

表示一个内存地址。

n/f/u三个参数可以一起使用。例如：

命令：x/3uh 0x54320 表示，从内存地址0x54320读取内容，h表示以双字节为一个单位，3表示三个单位，u表示按十六进制显示。

六、自动显示

你可以设置一些自动显示的变量，当程序停住时，或是在你单步跟踪时，这些变量会自动显示。相关的GDB命令是display。

display

display/

display/

expr是一个表达式，fmt表示显示的格式，addr表示内存地址，当你用display设定好了一个或多个表达式后，只要你的程序被停下来，GDB会自动显示你所设置的这些表达式的值。

格式i和s同样被display支持，一个非常有用的命令是：

display/i $pc

$pc是GDB的环境变量，表示着指令的地址，/i则表示输出格式为机器指令码，也就是汇编。于是当程序停下后，就会出现源代码和机器指令码相对应的情形，这是一个很有意思的功能。

下面是一些和display相关的GDB命令：

undisplay

delete display

删除自动显示，dnums意为所设置好了的自动显式的编号。如果要同时删除

几个，编号可以用空格分隔，如果要删除一个范围内的编号，可以用减号表示（如：2-5）
disable display
enable display
disable和enalbe不删除自动显示的设置，而只是让其失效和恢复。
info display
查看display设置的自动显示的信息。GDB会打出一张表格，向你报告当然调试中设置了多少个自动显示设置，其中包括，设置的编号，表达式，是否enable。
七、设置显示选项
GDB中关于显示的选项比较多，这里我只例举大多数常用的选项。
set print address
set print address on
打开地址输出，当程序显示函数信息时，GDB会显出函数的参数地址。系统默认为打开的，如：
(gdb) f
#0 set_quotes (lq=0x34c78 "<<", rq=0x34c88 ">>")
at input.c:530
530 if (lquote != def_lquote)

set print address off
关闭函数的参数地址显示，如：
(gdb) set print addr off
(gdb) f
#0 set_quotes (lq="<<", rq=">>") at input.c:530
530 if (lquote != def_lquote)
show print address
查看当前地址显示选项是否打开。
set print array
set print array on
打开数组显示，打开后当数组显示时，每个元素占一行，如果不打开的话，每个元素则以逗号分隔。这个选项默认是关闭的。与之相关的两个命令如下，我就不再多说了。
set print array off
show print array
set print elements
这个选项主要是设置数组的，如果你的数组太大了，那么就可以指定一个来指定数据显示的最大长度，当到达这个长度时，GDB就不再往下显示了。如果设置为0，则表示不限制。
show print elements
查看print elements的选项信息。
set print null-stop
如果打开了这个选项，那么当显示字符串时，遇到结束符则停止显示。这个选项默认为off。
set print pretty on
如果打开printf pretty这个选项，那么当GDB显示结构体时会比较漂亮。如：

```
$1 = {
next = 0x0,
flags = {
sweet = 1,
sour = 1
},
meat = 0x54 "Pork"
}
```

set print pretty off
关闭printf pretty这个选项，GDB显示结构体时会如下显示：

```
$1 = {next = 0x0, flags = {sweet = 1, sour = 1}, meat = 0x54 "Pork"}
```

show print pretty
查看GDB是如何显示结构体的。

set print sevenbit-strings
设置字符显示，是否按“\nnn”的格式显示，如果打开，则字符串或字符数据按\nnn显示，如“65”。
show print sevenbit-strings
查看字符显示开关是否打开。
set print union
设置显示结构体时，是否显式其内的联合体数据。例如有以下数据结构：

```
typedef enum {Tree, Bug} Species;
typedef enum {Big_tree, Acorn, Seedling} Tree_forms;
typedef enum {Caterpillar, Cocoon, Butterfly}
Bug_forms;
struct thing {
Species it;
union {
Tree_forms tree;
Bug_forms bug;
} form;
};
struct thing foo = {Tree, {Acorn}};
```

当打开这个开关时，执行 p foo 命令后，会如下显示：

```
$1 = {it = Tree, form = {tree = Acorn, bug = Cocoon}}
```

当关闭这个开关时，执行 p foo 命令后，会如下显示：

```
$1 = {it = Tree, form = {...}}
```

show print union
查看联合体数据的显示方式
set print object
在C++中，如果一个对象指针指向其派生类，如果打开这个选项，GDB会自动按照虚方法调用的规则显示输出，如果关闭这个选项的话，GDB就不管虚函数表了。这个选项默认是off。
show print object
查看对象选项的设置。
set print static-members
这个选项表示，当显示一个C++对象中的内容是，是否显示其中的静态数据

成员。默认是on。
show print static-members
查看静态数据成员选项设置。
set print vtbl
当此选项打开时，GDB将用比较规整的格式来显示虚函数表时。其默认是关闭的。
show print vtbl
查看虚函数显示格式的选项。

八、历史记录
当你用GDB的print查看程序运行时的数据时，你每一个print都会被GDB记录下来。GDB会以$1, $2, $3 .....这样的方式为你每一个print命令编上号。于是，你可以使用这个编号访问以前的表达式，如$1。这个功能所带来的好处是，如果你先前输入了一个比较长的表达式，如果你还想查看这个表达式的值，你可以使用历史记录来访问，省去了重复输入。

九、GDB环境变量
你可以在GDB的调试环境中定义自己的变量，用来保存一些调试程序中的运行数据。要定义一个GDB的变量很简单只需。使用GDB的set命令。GDB的环境变量和UNIX一样，也是以$起头。如：
set $foo = *object_ptr
使用环境变量时，GDB会在你第一次使用时创建这个变量，而在以后的使用中，则直接对其赋值。环境变量没有类型，你可以给环境变量定义任一的类型。包括结构体和数组。
show convenience
该命令查看当前所设置的所有的环境变量。
这是一个比较强大的功能，环境变量和程序变量的交互使用，将使得程序调试更为灵活便捷。例如：
set $i = 0
print bar[$i++]->contents
于是，当你就不必，print bar[0]->contents, print bar[1]->contents地输入命令了。输入这样的命令后，只用敲回车，重复执行上一条语句，环境变量会自动累加，从而完成逐个输出的功能。

十、查看寄存器
要查看寄存器的值，很简单，可以使用如下命令：
info registers
查看寄存器的情况。（除了浮点寄存器）
info all-registers
查看所有寄存器的情况。（包括浮点寄存器）
info registers
查看所指定的寄存器的情况。
寄存器中放置了程序运行时的数据，比如程序当前运行的指令地址（ip），程序的当前堆栈地址（sp）等等。你同样可以使用print命令来访问寄存器的

情况，只需要在寄存器名字前加一个$符号就可以了。如：p $eip。