# 用gdb调C++标准库

临峰不畏 发表于 3年前 阅读 1336 收藏 7 点赞 0 评论 3



*摘要: 研究了一下用gdb调试C++程序，研究如何让gdb显示标准库中有效的调试信息。*

我一直都是在Linux下做开发的，但是我对GDB的使用并不多。因为平都是用QtCreator调试程序的。因为工作的原因，以后可能不能再依赖QtCreator了。于是我好好研究一下～

之前为什么没有深入使用GDB，QtCreator带来一定的便利是一方面，另一方面是觉得GDB遇到了vector, map, set, list就没办法看数据了。

今天我研究了一下，其实也是Easy的。

示例代码：

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
using namespace std;

int main()
{
    int i1 = 32;
    int i2 = 45;
    double d = i1 + i2 / 3;

    vector<string> vstr;
    vstr.push_back("Hello");
    vstr.push_back("World");
    vstr.push_back("!");

    map<string, int> m_si;
    m_si["A"] = 12;
    m_si["D"] = 93;
    m_si["B"] = 77;

    return 0;
}
```

编译的时候：

```
$ g++ -o test-gdb test-gdb.cpp -g
```

在GDB调试中，可以用print指令查看变量或常量的值。

可能是我本机所安装的GDB版本较高的缘故，本机的GDB本谢就很支持STL中的容器。如下是我GDB的版本信息：

```
$ gdb --version
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-75.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
```

对STL容器的支持是极好的：

```
(gdb) p vstr
$1 = std::vector of length 3, capacity 4 = {"Hello", "World", "!"}

(gdb) p m_si
$2 = std::map with 3 elements = {
```

```
  ["A"] = 12,
  ["B"] = 77,
  ["D"] = 93
}
```

能看到这样的信息其实已经很不错了。

但是我公司系统的GDB可没这么智能。打印一下vector，就会蹦出好大堆信息。如果是map的话，那就更吓人了。

<此处展示可怕的提示信息>

其实，在这大堆信息里面只有小部分是我们关注的。GDB很灵活，我们可以在里面自定义函数。我可以在GDB里定义一个函数从vector里提取重要的信息进行显示。

从 这里 下载一个stl-views.gdb文件。在这个文里定义了很多绪如：pvector, pmap, pstring之类的GDB函数供我们使用。

将这个文件下载到HOME目录，然后：

```
$ cat stl-views-1.0.3.gdb >> .gdbinit
```

这样，每次启动 gdb的时候，都会自动加载 ~/.gdbinit 文件中的内容。

```
13        vector<string> vstr;
(gdb) n
14        vstr.push_back("Hello");
(gdb) n
15        vstr.push_back("World");
(gdb) n
16        vstr.push_back("!");
(gdb) p<TAB><TAB>
passcount     plist_member  print         pstring       python
path          pmap          print-object  ptype
pbitset       pmap_member   printf        pvector
pdequeue      ppqueue       pset          pwd
plist         pqueue        pstack        pwstring
```

输入了p之后，每次连续按两次TAB键都会列出以p开头的命令。这里我们看到了：pstring, pvector, pwstring, pstack, pmap, pmap_member等等。

我们用一下pvector来查看vstr中的内容：

```
(gdb) pvector vstr
elem[0]: $3 = "Hello"
elem[1]: $4 = "World"
elem[2]: $5 = "!"
Vector size = 3
Vector capacity = 4
Element type = std::basic_string<char, std::char_traits<char>, std::allocator<char> > *
```

这个命令果然打印出了很多有价值的信息。

博主有个习惯，我不仅要知其然，我还要知其所以然。于是我研究了一下 .gdbinit文件里的内容。这个函数都是在 ~/.gdbinit 里定义的。我们打开这个文件看一下。

```
define pvector
    if $argc == 0
        help pvector
    else
        set $size = $arg0._M_impl._M_finish - $arg0._M_impl._M_start
        set $capacity = $arg0._M_impl._M_end_of_storage - $arg0._M_impl._M_start
        set $size_max = $size - 1
    end
    if $argc == 1
        set $i = 0
        while $i < $size
            printf "elem[%u]: ", $i
            p *($arg0._M_impl._M_start + $i)
            set $i++
        end
    end
    if $argc == 2
        set $idx = $arg1
        if $idx < 0 || $idx > $size_max
```

```
            printf "idx1, idx2 are not in acceptable range: [0..%u].\n", $size_max
        else
            printf "elem[%u]: ", $idx
            p *($arg0._M_impl._M_start + $idx)
        end
    end
    if $argc == 3
      set $start_idx = $arg1
      set $stop_idx = $arg2
      if $start_idx > $stop_idx
        set $tmp_idx = $start_idx
        set $start_idx = $stop_idx
        set $stop_idx = $tmp_idx
      end
      if $start_idx < 0 || $stop_idx < 0 || $start_idx > $size_max || $stop_idx > $size_max
        printf "idx1, idx2 are not in acceptable range: [0..%u].\n", $size_max
      else
        set $i = $start_idx
        while $i <= $stop_idx
            printf "elem[%u]: ", $i
            p *($arg0._M_impl._M_start + $i)
            set $i++
        end
      end
    end
    if $argc > 0
        printf "Vector size = %u\n", $size
        printf "Vector capacity = %u\n", $capacity
        printf "Element "
        whatis $arg0._M_impl._M_start
    end
end

document pvector
    Prints std::vector<T> information.
    Syntax: pvector <vector> <idx1> <idx2>
    Note: idx, idx1 and idx2 must be in acceptable range [0..<vector>.size()-1].
    Examples:
    pvector v - Prints vector content, size, capacity and T typedef
    pvector v 0 - Prints element[idx] from vector
    pvector v 1 2 - Prints elements in range [idx1..idx2] from vector
end
```

看全部有点多，我们暂且不看多个参数的那部分，我们分析一下只有一个参数的一部分：

```
define pvector
    if $argc == 0           # 如果没有带参数，那么就打印帮助提示信息
        help pvector
    else                    # 如果有参数，那么接下来准备一下size, capacity, size_max 这三个重要的参数。
        set $size = $arg0._M_impl._M_finish - $arg0._M_impl._M_start        # arg0 就是第一个参数，也就是vstr数组对象。注重 size 是怎么计算的。
        set $capacity = $arg0._M_impl._M_end_of_storage - $arg0._M_impl._M_start
        set $size_max = $size - 1
    end
    if $argc == 1           # 如果只有一个参数，说明要求打印出vector中所有的元素
        set $i = 0
        while $i < $size    # 用一个 while 循环，用printf与p，打印出列表中的所有元素
            printf "elem[%u]: ", $i
            p *($arg0._M_impl._M_start + $i)     # 注意看哦！！！！
            set $i++
        end
    end
```

所有的说明都写在上面的注释中了，自己去悟吧！

如果print命令能像C++里的模板函数那可以对特定类型进行“偏特化”就好了。上面是有个问题的：

在“p *($arg0._M_impl._M_start + $i)”这行命令中，如果vector的成员还是个STL的容器该怎么办？这是个问题～