gdb简介

GDB是一个由GNU开源组织发布的、UNIX/LINUX操作系统下的、基于命令行的、功能强大的程序调试工具。对于一名Linux下工作的c++程序员，gdb是必不可少的工具。

启动gdb

对C/C++程序的调试，需要在编译前就加上-g选项:

```
$g++ -g hello.cpp -o hello
```

调试可执行文件:

```
$gdb <program>
```

program也就是你的执行文件，一般在当前目录下。
调试core文件(core是程序非法执行后core dump后产生的文件)。

```
$gdb <program> <core dump file>
$gdb program core.11127
```

调试服务程序。

```
$gdb <program> <PID>
$gdb hello 11127
```

如果你的程序是一个服务程序，那么你可以指定这个服务程序运行时的进程ID。gdb会自动attach上去，并调试他。program应该在PATH环境变量中搜索得到。

gbd交互式命令

启动gdb后，进入到交互模式，通过以下命令完成对程序的调试；注意高频使用的命令一般都会有缩写，熟练使用这些缩写命令能提高调试的效率。

运行

- run：简记为 r ，其作用是运行程序，当遇到断点后，程序会在断点处停止运行，等待用户输入下一步的命令。
- continue （简写c ）：继续执行，到下一个断点处（或运行结束）
- next：（简写 n），单步跟踪程序，当遇到函数调用时，也不进入此函数体；此命令同 step 的主要区别是，step 遇到用户自定义的函数，将步进到函数中去运行，而 next 则直接调用函数，不会进入到函数体内。
- step （简写s）：单步调试如果有函数调用，则进入函数；与命令n不同，n是不进入调用的函数的
- until：当你厌倦了在一个循环体内单步跟踪时，这个命令可以运行程序直到退出循环体。
- until+行号： 运行至某行，不仅仅用来跳出循环 finish： 运行程序，直到当前函数完成返回，并打印函数返回时的堆栈地址和返回值及参数值等信息。
- call 函数(参数)：调用程序中可见的函数，并传递“参数”，如：call gdb_test(55)
- quit：简记为 q ，退出gdb

设置断点

- break n （简写b n）:在第n行处设置断点 （可以带上代码路径和代码名称： b OAGUPDATE.cpp:578）
- b fn1 if a > b：条件断点设置

- break func（break缩写为b）：在函数func()的入口处设置断点，如：break cb_button
- delete 断点号n：删除第n个断点
- disable 断点号n：暂停第n个断点
- enable 断点号n：开启第n个断点
- clear 行号n：清除第n行的断点
- info b （info breakpoints）：显示当前程序的断点设置情况
- delete breakpoints：清除所有断点。

查看源代码

- list ：简记为 l ，其作用就是列出程序的源代码，默认每次显示10行。
- list 行号：将显示当前文件以“行号”为中心的前后10行代码，如：list 12
- list 函数名：将显示“函数名”所在函数的源代码，如：list main
- list ：不带参数，将接着上一次 list 命令的，输出下边的内容。

打印表达式

- print 表达式：简记为 p ，其中“表达式”可以是任何当前正在被测试程序的有效表达式，比如当前正在调试C语言的程序，那么“表达式”可以是任何C语言的有效表达式，包括数字，变量甚至是函数调用。
- print a：将显示整数 a 的值
- print ++a：将把 a 中的值加1,并显示出来
- print name：将显示字符串 name 的值
- print gdb_test(22)：将以整数22作为参数调用 gdb_test() 函数
- print gdb_test(a)：将以变量 a 作为参数调用 gdb_test() 函数
- display 表达式：在单步运行时将非常有用，使用display命令设置一个表达式后，它将在每次单步进行指令后，紧接着输出被设置的表达式及值。如： display a
- watch 表达式：设置一个监视点，一旦被监视的“表达式”的值改变，gdb将强行终止正在被调试的程序。如： watch a

- whatis ：查询变量或函数
- nfo function： 查询函数
- 扩展info locals： 显示当前堆栈页的所有变量

查询运行信息

- where/bt ：当前运行的堆栈列表；
- bt backtrace 显示当前调用堆栈
- up/down 改变堆栈显示的深度
- set args 参数:指定运行时的参数
- show args：查看设置好的参数
- info program： 来查看程序的是否在运行，进程号，被暂停的原因。

分割窗口

- layout：用于分割窗口，可以一边查看代码，一边测试
- layout src：显示源代码窗口
- layout asm：显示反汇编窗口
- layout regs：显示源代码/反汇编和CPU寄存器窗口
- layout split：显示源代码和反汇编窗口
- Ctrl + L：刷新窗口

更强大的工具cgdb

cgdb可以看作gdb的界面增强版,用来替代gdb的 gdb -tui。cgdb主要功能是在调试时进行代码的同步显示，这无疑增加了调试的方便性，提高了调试效率。界面类似vi，符合unix/linux下开发人员习惯;如果熟悉gdb和vi，几乎可以立即使用cgdb。

实践

测试程序

测试程序test.c如下。

```
#include<stdio.h>

   int main()
  {
      int i=0;
      for( i=0;i<5;i++)
     {
          printf("i=%d\n",i);
      }

      return 0;
  }
```

调试步骤

1. 编译可调试程序

```
gcc -g test.c -o test
```

必须添加-g的编译选项才可生成支持调试的可执行文件。

2. 启动GDB

```
gdb test
gdb 可执行文件名:这样子就启动GDB工具对执行文件的调试。相当于IDE中的Debug选项。
```

3. 设置断点

```
```

```
break(b) 函数名：在某函数入口处添加断点
break(b) 行号：在指定行添加断点
break(b) 文件名:行号,在指定文件的指定行添加断点
```

括号中的b为简写式命令，同样可用（下同）。

4. 程序运行

```
run      ：运行程序
next(n) ： 单步运行程序(不进入子函数)
step(s) ： 单步运行程序(进入子函数)
continue(c) ： 继续运行程序
```

5. 查看变量值：

```
print(p)  变量名    ：查看指定变量值
```

6. 退出GDB

```
quit(q)     ：退出GDB
```

实测过程