如何寫 gdb 命令腳本

作為 UNIX/Linux 下使用廣泛的調試器，gdb 不僅提供了豐富的命令，還引入了對腳本的支持：一種是對已存在的腳本語言支持，比如 python，用戶可以直接書寫 python 腳本，由 gdb 調用 python 解釋器執行；另一種是命令腳本（command file），用戶可以在腳本中書寫 gdb 已經提供的或者自定義的 gdb 命令，再由 gdb 執行。在這篇文章裡，我會介紹一下如何寫 gdb 的命令腳本。

(一) 自定義命令
gdb 支持用戶自定義命令，格式是：

```
define commandName
    statement
    ......
end
```

其中 statement 可以是任意 gdb 命令。此外自定義命令還支持最多 10 個輸入參數：**$arg0，$arg1 …… $arg9**，並且還用**$argc** 來標明一共傳入了多少參數。

下面結合一個簡單的 C 程序（test.c），來介紹如何寫自定義命令：

```
#include <stdio.h>

int global = 0;

int fun_1(void)
{
    return 1;
}

int fun_a(void)
{
    int a = 0;
    printf("%d\n", a);
}

int main(void)
{
    fun_a();
    return 0;
}
```

首先編譯成可執行文件：

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

接著用 gdb 進行調試：

```
[root@linux:~]$ gdb test
GNU gdb (GDB) 7.6
Copyright (C) 2013 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /data2/home/nanxiao/test...done.
(gdb) b fun_a
Breakpoint 1 at 0x4004d7: file test.c, line 12.
(gdb) r
Starting program: /data2/home/nanxiao/test

Breakpoint 1, fun_a () at test.c:12
12              int a = 0;
(gdb) bt
#0  fun_a () at test.c:12
#1  0x0000000000400500 in main () at test.c:18
```

可以看到使用 bt（backtrace 縮寫）命令可以打印當前線程的調用棧。我們的第一個自定義命令就是也實現一個 backtrace 功能：

```
define mybacktrace
    bt
end
```

怎麼樣？簡單吧，純粹復用 gdb 提供的命令。下面來驗證一下：

```
(gdb) define mybacktrace
Type commands for definition of "mybacktrace".
End with a line saying just "end".
>bt
```

```
>end
(gdb) mybacktrace
#0  fun_a () at test.c:12
#1  0x0000000000400500 in main () at test.c:18
```

功能完全正確！

接下來定義一個賦值命令，把第二個參數的值賦給第一個參數：

```
define myassign
    set var $arg0 = $arg1
end
```

執行一下：

```
(gdb) define myassign
Type commands for definition of "myassign".
End with a line saying just "end".
>set var $arg0 = $arg1
>end
(gdb) myassign global 3
(gdb) p global
$1 = 3
```

可以看到 global 變量的值變成了 3。

對於自定義命令來說，傳進來的參數隻是進行簡單的文本替換，所以你可以傳入賦值的表達式，甚至是函數調用：

```
(gdb) myassign global fun_1()
(gdb) p global
$2 = 1
```

可以看到 global 變量的值變成了 1。

除此以外，還可以為自定義命令寫幫助文檔，也就是執行 help 命令時打印出的信息：

```
document myassign
    assign the second parameter value to the first parameter
end
```

執行 help 命令：

```
(gdb) document myassign
Type documentation for "myassign".
End with a line saying just "end".
>assign the second parameter value to the first parameter
>end
(gdb) help myassign
assign the second parameter value to the first parameter
```

可以看到打印出了 myassign 的幫助信息。

（二） 命令腳本

首先對於命令腳本的命名，其實 gdb 沒有什麼特殊要求，只要文件名不是 gdb 支持的其它腳本語言的文件名就可以了（比如.py）。因為這樣做會使 gdb 按照相應的腳本語言去解析命令腳本，結果自然是不對的。

其次為了幫助用戶寫出功能強大的腳本，gdb 提供了如下的流程控制命令：
（1）條件命令：**if...else...end**。這個同其它語言中提供的 if 命令沒什麼區別，只是注意結尾的 end。
（2）循環命令：**while...end**。gdb 同樣提供了 **loop_break** 和 **loop_continue** 命令分別對應其它語言中的 break 和 continue，另外同樣注意結尾的 end。

另外 gdb 還提供了很多輸出命令。比方說 echo 命令，如果僅僅是輸出一段文本，echo 命令特別方便。此外還有和 C 語言很相似的支持格式化輸出的 printf 命令，等等。

腳本文件的**註釋也是以#開頭**的，這個同很多其它腳本語言都一樣。

最後指出的是**在 gdb 中執行腳本要使用 source 命令**，例如：「source xxx.gdb」。

（三） 一個完整的例子

最後以一個完整的 gdb 腳本（search_byte.gdb）做例子，來總結一下本文提到的內容:

```
define search_byte
    if $argc != 3
        help search_byte
    else
        set $begin_addr = $arg0
        set $end_addr = $arg1

        while $begin_addr <= $end_addr
            if *((unsigned char*)$begin_addr) == $arg2
                printf "Find it ! The address is 0x%x\n", $begin_addr
```

```
            loop_break
        else
            set $begin_addr = $begin_addr + 1
        end
    end

    if $begin_addr > $end_addr
        printf "Can't find it!\n"
    end
  end
end

document search_byte
    search a specified byte value(0 ~ 255) during a memory
    usage: search_byte begin_addr end_addr byte
end
```

這個腳本定義了 search_byte 命令，目的是在一段指定內存查找一個值（unsigned char 類型）：需要輸入內存的**起始地址**，**結束地址**和**要找的值**。
命令邏輯可以分成 3 個部分：
（a） 首先判斷輸入參數是不是 3 個，如果不是，就輸出幫助信息；
（b） 從起始地址開始查找指定的值，如果找到，打印地址值並退出循環，否則把地址加 1，繼續查找；
（c） 如果在指定內存區域沒有找到，打印提示信息。

另外這個腳本還定義了 search_byte 的幫助信息。

仍然以上面的 C 程序為例，來看一下如何使用這個 gdb 腳本：

```
[root@linux:~]$ gdb test
GNU gdb (GDB) 7.6
Copyright (C) 2013 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 "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /data2/home/nanxiao/test...done.
(gdb) p &global
```

```
$1 = (int *) 0x600900 <global>
(gdb) p global
$2 = 0
(gdb) source search_byte.gdb
(gdb) search_byte 0x600900 0x600903 0
Find it! The address is 0x600900
(gdb) search_byte 0x600900 0x600903 1
Can't find it!
```

可以看到 global 的值是 0，起始地址是 0x600900，結束地址是 0x600903。在 global 的內存區域查找 0 成功，查找 1 失敗。

受篇幅所限，本文只是對 gdb 命令腳本做了一個粗淺的介紹，旨在起到拋磚引玉的效果。如果大家想更深入地瞭解這部分知識，可以參考 gdb 手冊的相關章節：Extending GDB (https://sourceware.org/gdb/onlinedocs/gdb/Extending-GDB.html)。最後向大家推薦一個 github 上的.gdbinit 文件：https://github.com/gdbinit/Gdbinit/blob/master/gdbinit，把這個弄懂，相信 gdb 腳本文件就不在話下了。

參考文獻：

（1）Extending GDB (https://sourceware.org/gdb/onlinedocs/gdb/Extending-GDB.html);
（2）捉蟲日記（http://www.ituring.com.cn/book/909）。