关于作者

|| << 2010年10月 庚寅年（虎） >> ||

日	一	二	三	四	五	六
					1 国庆节	2 廿五
3 廿六	4 廿七	5 廿八	6 廿九	7 三十	8 寒露	9 初二
10 初三	11 初四	12 初五	13 初六	14 初七	15 初八	16 重阳节
17 初十	18 十一	19 十二	20 十三	21 十四	22 十五	23 霜降
24 十七	25 十八	26 十九	27 二十	28 廿一	29 廿二	30 廿三
31 廿四						

我的分类

- 我的文章分类
- 我的图片分类
  - me
- 我的链接分类
- 我的音乐分类

Linux下C程序命令行参数处理函数getopt

Linux下很多程序甚至那些具有图形用户界面（graphical user interface，GUI）的程序，都能接受和处理命令行选项。对于某些程序，这是与用户进行交互的主要手段。具有可靠的复杂命令行参数处理机制，会使得您的应用程序更好、更有用。getopt() 是一个专门设计来减轻命令行处理负担的库函数。

1、命令行参数

命令行程序设计的首要任务是解析命令行参数，GUI派的程序员很少关心这个。这里，对参数采用了一种比较通俗的定义：命令行上除命令名之外的字符串。参数由多项构成，项与项之间用空白符彼此隔开。

参数进一步分为选项 和操作数 。选项用于修改程序的默认行为或为程序提供信息，比较老的约定是以短划线开头。选项后可以跟随一些参数，称为选项参数。剩下的就是操作数了。

2、POSIX约定

POSIX表示可移植操作系统接口： Portable Operating System Interface，电气和电子工程师协会（Institute of Electrical and Electronics Engineers，IEEE）最初开发 POSIX 标准，是为了提高 UNIX 环境下应用程序的可移植性。然而，POSIX 并不局限于 UNIX。许多其它的操作系统，例如 DEC OpenVMS 和 Microsoft Windows NT，都支持 POSIX 标准。

下面是POSIX标准中关于程序名、参数的约定：

- 程序名不宜少于2个字符且不多于9个字符；
- 程序名应只包含小写字母和阿拉伯数字；
- 选项名应该是单字符活单数字，且以短横'-'为前缀；
- 多个不需要选项参数的选项，可以合并。（譬如：foo -a -b -c ---->foo -abc）
- 选项与其参数之间用空白符隔开；
- 选项参数不可选。
- 若选项参数有多值，要将其并为一个字串传进来。譬如：myprog -u "arnold,joe,jane"。这种情况下，需要自己解决这些参数的分离问题。
- 选项应该在操作数出现之前出现。
- 特殊参数'--'指明所有参数都结束了，其后任何参数都认为是操作数。
- 选项如何排列没有什么关系，但对互相排斥的选项，如果一个选项的操作结果覆盖其他选项的操作结果时，最后一个选项起作用；如果选项重复，则顺序处理。
- 允许操作数的顺序影响程序行为，但需要作文档说明。
- 读写指定文件的程序应该将单个参数'-'作为有意义的标准输入或输出来对待。

3、GNU长选项

GNU鼓励程序员使用--help、--verbose等形式的长选项。这些选项不仅不与POSIX约定冲突，而且容易记忆，另外也提供了在所有GNU工具之间保持一致性的机会。GNU长选项有自己的约定：

- 对于已经遵循POSIX约定的GNU程序，每个短选项都有一个对应的长选项。
- 额外针对GNU的长选项不需要对应的短选项，仅仅推荐要有。
- 长选项可以缩写成保持惟一性的最短的字串。
- 选项参数与长选项之间或通过空白字符活通过一个'='来分隔。
- 选项参数是可选的（只对短选项有效）。
- 长选项允许以一个短横线为前缀。

4、基本的命令行处理技术

C程序通过argc和argv参数访问它的命令行参数。argc是整型数，表示参数的个数（包括命令名）。main()函数的定义方式有两种，区别仅在于argv如何定义：

```
int main(int argc, char *argv[])
{
  ......
}
```

```
int main(int argc, char **argv)
{
  ......
}
```

当 C 运行时库的程序启动代码调用 main() 时，已经对命令行进行了处理。argc 参数包含参数的计数值，而 argv 包含指向这些参数的指针数组。argv[0]是程序名。

一个很简单的命令行处理技术的例子是echo程序，它可以将参数输出到标准设备上，用空格符隔开，最后换行。若命令行第一个参数为-n，那么就不会换行。

清单1：

```
#include <stdio.h>

int main(int argc, char **argv)
{
   int i, nflg;

   nflg = 0;
   if(argc > 1 && argv[1][0] == '-' && argv[1][1] == 'n'){
      nflg++;
      argc--;
      argv++;
   }
   for(i=1; i<argc; i++){
      fputs(argv[i], stdout);
      if(i < argc-1)
         putchar(' ');
   }
   if(nflg == 0)
      putchar('\n');

   return 0;
}
```

echo程序中，对于命令行参数的解析是手动实现的。很久以前，Unix支持小组为了简化对于命令行参数的解析，开发了**getopt()**函数，同时提供了几个外部变量，使得编写遵守POSIX的代码变得更加容易了。

5、命令行参数解析函数 —— getopt()

getopt()函数声明如下：

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;
```

该函数的argc和argv参数通常直接从main()的参数直接传递而来。optstring是选项字母组成的字串。如果该字串里的任一字符后面有冒号，那么这个选项就要求有选项参数。

当给定getopt()命令参数的数量 (argc)、指向这些参数的数组 (argv) 和选项字串 (optstring) 后，getopt() 将返回第一个选项，并设置一些全局变量。使用相同的参数再次调用该函数时，它将返回下一个选项，并设置相应的全局变量。如果不再有可识别的选项，将返回 -1 ，此任务就完成了。

getopt() 所设置的全局变量包括：

- char *optarg ——当前选项参数字串（如果有）。
- int optind ——argv的当前索引值。当getopt()在while循环中使用时，循环结束后，剩下的字串视为操作数，在argv[optind]至argv[argc-1]中可以找到。
- int opterr——这个变量非零时，getopt()函数为“无效选项”和“缺少参数选项，并输出其错误信息。
- int optopt ——当发现无效选项字符之时，getopt()函数或返回'?'字符，或返回':'字符，并且optopt包含了所发现的无效选项字符。

以下面的程序为例：
选项：

- -n —— 显示“我的名字”。
- -g —— 显示“我女朋友的名字”。
- -l —— 带参数的选项.

清单2：

```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char **argv)
{
   int oc;                /*选项字符 */
   char *b_opt_arg;       /*选项参数字串 */

   while((oc = getopt(argc, argv, "ngl:")) != -1)
   {
```

```
    switch(oc)
    {
       case 'n':
          printf("My name is Lyong.\n");
          break;
       case 'g':
          printf("Her name is Xxiong.\n");
          break;
       case 'l':
          b_opt_arg = optarg;
          printf("Our love is %s\n", optarg);
          break;
    }
  }
  return 0;
}
```

运行结果：

```
$ ./opt_parse_demo -n
My name is Lyong.
$ ./opt_parse_demo -g
Her name is Xxiong.
$ ./opt_parse_demo -l forever
Our love is forever
$ ./opt_parse_demo -ngl forever
My name is Lyong.
Her name is Xxiong.
Our love is forever
```

6、改变getopt()对错误命令行参数信息的输出行为

不正确的调用程序在所难免，这种错误要么是命令行选项无效，要么是缺少选项参数。正常情况下，getopt()会为这两种情况输出自己的出错信息，并且返回'?'。为了验证此事，可以修改一下上面的清单2中的代码。

清单3：

```
#include <stdio.h>
#include <unistd.h>

int main (int argc, char **argv)
{
   int oc;               /*选项字符 */
   char *b_opt_arg;         /*选项参数字串 */

   while((oc = getopt(argc, argv, "ngl:")) != -1)
   {
      switch(oc)
      {
         case 'n':
            printf("My name is Lyong.\n");
            break;
          case 'g':
            printf("Her name is Xxiong.\n");
            break;
         case 'l':
            b_opt_arg = optarg;
            printf("Our love is %s\n", optarg);
            break;
         case '?':
            printf("arguments error!\n");
            break;
      }
   }
   return 0;
}
```

输入一个错误的命令行，结果如下：

```
$ ./opt_parse_demo -l
./opt_parse_demo: option requires an argument -- l
arguments error!
```

如果不希望输出任何错误信息，或更希望输出自定义的错误信息。可以采用以下两种方法来更改getopt()函数的出错信息输出行为：

□□□ 在调用getopt()之前，将opterr设置为0，这样就可以在getopt()函数发现错误的时候强制它不输出任何消息。

□□□ 如果optstring参数的第一个字符是冒号，那么getopt()函数就会保持沉默，并根据错误情况返回不同字符，如下：

- □□□ “无效选项” —— getopt()返回'?'，并且optopt包含了无效选项字符（这是正常的行为）。
- □□□ “缺少选项参数” —— getopt()返回':'，如果optstring的第一个字符不是冒号，那么getopt()返回'?'，这会使得这种情况不能与无效选项的情况区分开。

清单4：

```
#include <stdio.h>
```

```
#include <unistd.h>

int main (int argc, char **argv)
{
   int oc;               /*选项字符 */
   char ec;                    /*无效的选项字符*/
   char *b_opt_arg;          /*选项参数字串 */

   while((oc = getopt(argc, argv, ":ngl:")) != -1)
   {
      switch(oc)
      {
         case 'n':
            printf("My name is Lyong.\n");
            break;
          case 'g':
            printf("Her name is Xxiong.\n");
            break;
         case 'l':
            b_opt_arg = optarg;
            printf("Our love is %s\n", optarg);
            break;
         case '?':
            ec = (char)optopt;
            printf("无效的选项字符 \' %c \'!\n", ec);
            break;
         case ':':
            printf("缺少选项参数! \n");
            break;
      }
   }
   return 0;
}
```

测试结果:

```
$ ./opt_parse_demo -a
无效的选项字符 ' a '!
$ ./opt_parse_demo -l
缺少选项参数!
```

7、GNU提供的getopt()函数的特点

上面所设计的getopt()函数是UNIX支持小组提供的，其执行时一碰到不以'-'开始的 命令行参数就停止寻找选项。而GNU提供的getopt()函数与之不同，它会扫描整个命令行来寻找选项。当调用GNU getopt()函数并处理命令行参数的时候，它重新排列argv中的元素，这样当重排结束时，所有选项都被移动到前面并且那些继续检查argv [optind]至argv[argc-1]中剩余参数的代码仍正常工作，但在任何情况下，碰到特殊参数'--'就结束对选项的扫描。

可以输入一个乱序的命令行，查看opt_parse_demo的输出:

```
$ ./opt_parse_demo -l forever a b c d -g -n
Our love is forever
Her name is Xxiong.
My name is Lyong.
```

GNU getopt()第二个特点是可以在optstring中使用特殊的首字符改变getopt()的默认行为:

- optstring[0] = '+'，这样就与UNIX支持小组提供的getopt()很相近了。
- optstring[0] = '-'，会在optarg中得到命令行中的每个参数。
- 以上两种情况下，':'可以作为第二个字符使用。

GNU getopt()第三个特点是optstring中的选项字符后面接两个冒号，就允许该选项有可选的选项参数。在选项参数不存在的情况下，GNU getopt()返回选项字符并将optarg设置为NULL。

8、GNU长选项命令行解析

20 世纪 90 年代，UNIX 应用程序开始支持长选项，即一对短横线、一个描述性选项名称，还可以包含一个使用等号连接到选项的参数。

GNU提供了getopt-long()和getopt-long-only()函数支持长选项的命令行解析，其中，后者的长选项字串是以一个短横线开始的，而非一对短横线。

getopt_long() 是同时支持长选项和短选项的 getopt() 版本。下面是它们的声明:

```
#include <getopt.h>

int getopt_long(int argc, char * const argv[], const char *optstring, const struct option *longopts, int *longindex);

int getopt_long_only(int argc, char * const argv[],const char *optstring,const struct option *longopts, int
*longindex);
```

getopt_long()的前三个参数与上 面的getopt()相同，第4个参数是指向option结构的数组，option结构被称为“长选项表”。longindex参数如果没有设置为 NULL，那么它就指向一个变量，这个变量会被赋值为寻找到的长选项在longopts中的索引值，

这可以用于错误诊断。

option结构在getopt.h中的声明如下：

```
struct option{
   const char *name;
   int has_arg;
   int *flag;
   int val;
};
```

对结构中的各元素解释如下：

const char *name

这是选项名，前面没有短横线。譬如"help"、"verbose"之类。

int has_arg

描述了选项是否有选项参数。如果有，是哪种类型的参数，此时，它的值一定是下表中的一个。

符号常量	数值	含义
no_argument	0	选项没有参数
required_argument	1	选项需要参数
optional_argument	2	选项参数可选

int *flag

如果这个指针为NULL，那么 getopt_long()返回该结构val字段中的数值。如果该指针不为NULL，getopt_long()会使得它所指向的变量中填入val字段中 的数值，并且getopt_long()返回0。如果flag不是NULL，但未发现长选项，那么它所指向的变量的数值不变。

int val

这个值是发现了长选项时的返回值，或者flag不 是NULL时载入*flag中的值。典型情况下，若flag不是NULL，那么val是个真／假值，譬如1或0；另一方面，如果flag是NULL，那么 val通常是字符常量，若长选项与短选项一致，那么该字符常量应该与optstring中出现的这个选项的参数相同。

每个长选项在长选项表中都有一个单独条目，该条目里需要填入正确的数值。数组中最后的元素的值应该全是0。数组不需要排序，getopt_long()会进行线性搜索。但是，根据长名字来排序会使程序员读起来更容易。

以上所说的flag和val的用法看上去有点混乱，但它们很有实用价值，因此有必要搞透彻了。

大部分时候，程序员会根据getopt_long()发现的选项，在选项处理过程中要设置一些标记变量，譬如在使用getopt()时，经常做出如下的程序格式：

```
int do_name, do_gf_name, do_love; /*标记变量*/
char *b_opt_arg;

while((c = getopt(argc, argv, ":ngl:")) != -1)
{
   switch (c){
   case 'n':
      do_name = 1;
   case 'g':
      do_gf_name = 1;
      break;
      break;
   case 'l':
      b_opt_arg = optarg;
   ......
   }
}
```

当flag不为NULL时，getopt_long*()会为你设置标记变量。也就是说上面的代码中，关于选项'n'、'l'的处理，只是设置一些 标记，如果flag不为NULL,时，getopt_long()可以自动为各选项所对应的标记变量设置标记，这样就能够将上面的switch语句中的两 种种情况减少到了一种。下面给出一个长选项表以及相应处理代码的例子。

清单**5**：

```
#include <stdio.h>
#include <getopt.h>

int do_name, do_gf_name;
char *l_opt_arg;

struct option longopts[] = {
   { "name",      no_argument,        &do_name,      1   },
   { "gf_name",   no_argument,        &do_gf_name,   1   },
   { "love",      required_argument,  NULL,          'l'   },
   {   0,   0,   0,   0},
};

int main(int argc, char *argv[])
```

```
{
   int c;

   while((c = getopt_long(argc, argv, ":l:", longopts, NULL)) != -1){
      switch (c){
      case 'l':
         l_opt_arg = optarg;
         printf("Our love is %s!\n", l_opt_arg);
         break;
      case 0:
         printf("getopt_long()设置变量：do_name = %d\n", do_name);
         printf("getopt_long()设置变量：do_gf_name = %d\n", do_gf_name);
         break;
      }
   }
   return 0;
}
```

在进行测试之前，再来回顾一下有关option结构中的指针flag的说明吧。

> 如果这个指针为NULL，那么 getopt_long()返回该结构val字段中的数值。如果该指针不为NULL，getopt_long()会使得它所指向的变量中填入val字段中 的数值，并且getopt_long()返回0。如果flag不是NULL，但未发现长选项，那么它所指向的变量的数值不变。

下面测试一下：

```
$ ./long_opt_demo --name
getopt_long()设置变量：do_name = 1
getopt_long()设置变量：do_gf_name = 0

$ ./long_opt_demo --gf_name
getopt_long()设置变量：do_name = 0
getopt_long()设置变量：do_gf_name = 1

$ ./long_opt_demo --love forever
Our love is forever!

$ ./long_opt_demo -l forever
Our love is forever!
```

测试过后，应该有所感触了。关于flag和val的讨论到此为止。下面总结一下get_long()的各种返回值的含义：

返回值	含 义
0	getopt_long()设置一个标志，它的值与option结构中的val字段的值一样
1	每碰到一个命令行参数，optarg都会记录它
'?'	无效选项
':'	缺少选项参数
'x'	选项字符'x'
-1	选项解析结束

从实用的角度来说，我们更期望每个长选项都对应一个短选项，这种情况下，在option结构中，只要将flag设置为NULL，并将val设置为长选项所对应的短选项字符即可。譬如上面清单5中的程序，修改如下。

清单**6**：

```
#include <stdio.h>
#include <getopt.h>

int do_name, do_gf_name;
char *l_opt_arg;

struct option longopts[] = {
   { "name",      no_argument,         NULL,           'n'    },
   { "gf_name",   no_argument,         NULL,           'g'    },
   { "love",      required_argument,   NULL,           'l'    },
   {    0,   0,   0,   0},
};

int main(int argc, char *argv[])
{
   int c;

   while((c = getopt_long(argc, argv, ":l:", longopts, NULL)) != -1){
      switch (c){
      case 'n':
         printf("My name is LYR.\n");
         break;
      case 'g':
         printf("Her name is BX.\n");
         break;
      case 'l':
         l_opt_arg = optarg;
         printf("Our love is %s!\n", l_opt_arg);
         break;
```

```
        }
    }
    return 0;
}
```

测试结果如下：

```
$ ./long_opt_demo --name --gf_name --love forever
My name is LYR.
Her name is BX.
Our love is forever!

$ ./long_opt_demo -ng -l forever
My name is LYR.
Her name is BX.
Our love is forever!
```

9、在LINUX之外的系统平台上使用GNU getopt()或getopt_long()

只 要从GNU程序或GNU C Library(GLIBC)的CVS档案文件中copy源文件即可（http://sourceware.org/glibc/）。所需源文件是 getopt.h、getopt.c和getoptl.c，将这些文件包含在你的项目中。另外，你的项目中最好也将COPYING.LIB文件包含进去， 因为GNU LGPL（GNU 程序库公共许可证）的内容全部包括在命名为COPYING.LIB 的文件中。

10、结论

程序需要能够快速处理各个选项和参数，且要求 不会浪费开发人员的太多时间。在这一点上，无论是GUI(图形用户交互）程序还是CUI（命令行交互）程序，都是其首要任务，其区别仅在于实现方式的不 同。GUI通过菜单、对话框之类的图形控件来完成交互，而CUI使用了纯文本的交互方式。在程序开发中，许多测试程序用CUI来完成是首选方案。

getopt() 函数是一个标准库调用，可允许您使用直接的 while/switch 语句方便地逐个处理命令行参数和检测选项（带或不带附加的参数）。与其类似的 getopt_long() 允许在几乎不进行额外工作的情况下处理更具描述性的长选项，这非常受开发人员的欢迎。

原文地址 http://www.1to2.us/Linux-getopt-a118887.htm

原文地址 http://www.1to2.us/Linux-getopt-a118887.htm

发表于： 2010-10-17，修改于： 2010-10-18 00:16，已浏览3次，有评论0条 推荐 投诉

网友评论

发表评论

用户名： 密码： 免费注册

验证码： 64G2 匿名

提交