一个人的天空

libpopt的使用（译）

更新日志：
2012.02.27 更新（校正并基本完成选项表部分内容的翻译）
2012.02.28 更新（完成popt基本使用的第1部分以及第2部分的内容翻译）
2012.03.01 更新（完成popt基本使用的全部翻译——剩下的3、4、5部分，增加示例部分）

软件安装：
ubuntu (deb)

```
$ apt-cache search popt|head
libpopt-dev - lib for parsing cmdline parameters - development files
libpopt0 - lib for parsing cmdline parameters
...
$ sudo apt-get install libpopt-dev
```

CentOS (rpm)

```
$ sudo yum install popt-devel
```

本文译自linux程序员指南（Linux Programmer's Manual）

man 3 popt

概要

```
#include <popt.h>
poptContext poptGetContext(const char * name, int argc,
                           const char ** argv,
                           const struct poptOption * options,
                           int flags);
void poptFreeContext(poptContext con);
void poptResetContext(poptContext con);
int poptGetNextOpt(poptContext con);
const char * poptGetOptArg(poptContext con);
```

随笔档案


```
const char * poptGetArg(poptContext con);
const char * poptPeekArg(poptContext con);
const char ** poptGetArgs(poptContext con);
const char *const poptStrerror(const int error);
const char * poptBadOption(poptContext con, int flags);
int poptReadDefaultConfig(poptContext con, int flags);
int poptReadConfigFile(poptContext con, char * fn);
int poptAddAlias(poptContext con, struct poptAlias alias,
                 int flags);
int poptParseArgvString(char * s, int *  argcPtr,
                        const char *** argvPtr);
int poptDupArgv(int argc, const char ** argv, int * argcPtr,
                        const char *** argvPtr);
int poptStuffArgs(poptContext con, const char ** argv);
```

描述

popt 库的存在，主要是为了解析命令行选项。相比手动的解析argv数组或者使用getopt函数getopt()或getopt_long()，使用popt库在很多方面都占有优势。popt的一些特点：它不使用全局变量，因此可以并行解析argv；它可以解析任意的argv风格的元素数组，可以解析来自任何源文件的命令行字符串（ps:估计是命令行字符串参数可保存在文件中，程序通过导入配置文件解析获得参数）；它提供了一个使用选项别名的标准方法（后面会再讨论）；它可以执行外部选项过滤器；最后，它能够自动生成程序的help和usage信息。

类似getopt_long()，popt库支持短风格和长风格选项。一个短风格选项由一个连字符 '-' 后面接一个字母或数字组成。在GNU实用程序（工具）中，通常的，一个长风格选项，由两个连字符 '-' 后面接一个由字母、数字或连字符组成的字符串。长选项可选的允许由一个连字符 '-' 开头， 主要是为了允许基于popt的程序与基于X toolkit（工具包）的程序命令行兼容。不管那种类型（短风格或长风格）选项，后面都跟随一个参数。短选项及其参数由一个空格隔开，长选项及其参数由一个空格或者一个 = 号隔开。

popt库是高度可移植的，而且将能够工作于任何POSIX兼容的平台。最新的发布版本可从 ftp://ftp.rpm.org/pub/rpm/dist 获得。

popt的基本使用

1. 选项表

应用程序通过"选项表"的方式为popt提供命令行选项信息，例如，一个 struct poptOption 的结构数组：

```
#include <popt.h>

struct poptOption {
    const char * longName; /* may be NULL */
    char shortName;        /* may be '\0' */
```

2013年12月 (7)
2013年11月 (5)
2013年10月 (7)
2013年9月 (12)
2013年8月 (19)
2013年7月 (6)
2013年6月 (4)
2013年5月 (29)
2013年4月 (21)
2013年3月 (15)
2013年2月 (16)
2013年1月 (52)
2012年12月 (25)
2012年11月 (10)
2012年10月 (10)
2012年8月 (23)
2012年7月 (21)
2012年6月 (23)
2012年5月 (23)
2012年4月 (8)
2012年3月 (22)
2012年2月 (18)
2012年1月 (3)
2011年12月 (38)
2011年11月 (19)
2011年10月 (2)
2011年9月 (9)
2011年8月 (9)
2011年7月 (17)
2011年6月 (25)
2011年5月 (59)
2011年4月 (2)
2011年3月 (4)
2011年2月 (6)
2011年1月 (6)
2010年12月 (3)
2010年11月 (13)
2010年10月 (11)
2010年9月 (6)
2010年8月 (22)
2010年7月 (1)
2010年6月 (3)
2010年5月 (15)
2010年4月 (2)
2010年3月 (6)
2010年2月 (2)
2010年1月 (5)
2009年12月 (5)
2009年11月 (1)
2009年9月 (3)

```
    int argInfo;
    void * arg;             /* depends on argInfo */
    int val;                /* 0 means don't return, just update flag */
    char * descrip;         /* description for autohelp -- may be
NULL */
    char * argDescrip;      /* argument description for autohelp */
};
```

（为表达方便，下文用选项域指代struct poptOption的每一个成员）

选项表（struct poptOption 结构数组）的每一个成员（struct poptOption 结构体）定义了一个将要传给应用程序的单一选项，长选项或短选项视为一个单一选项可能出现两种不同形式。**结构体**（struct poptOption）最前面的两个成员，longName 和 shortName，定义了选项的名字，前者是长名字，后者是一个单一字符。

选项域 argInfo 指定了 popt 选项后面期待的参数类型，如果选项后面没有参数， 宏 POPT_ARG_NONE 将被使用，其他的有效值如下表所示：

Value	Description	arg Type
POPT_ARG_NONE	No argument expected	int
POPT_ARG_STRING	No type checking to be performed	char *
POPT_ARG_ARGV	No type checking to be performed	char **
POPT_ARG_INT	An integer argument is expected	int
POPT_ARG_LONG	A long integer is expected	long
POPT_ARG_LONGLONG	A long long integer is expected	long long
POPT_ARG_VAL	Integer value taken from val	int
POPT_ARG_FLOAT	An float argument is expected	float
POPT_ARG_DOUBLE	A double argument is expected	double

对于数值类型的变量，如果argInfo选项域的值与 POPT_ARGFLAG_OR, POPT_ARGFLAG_AND 或 POPT_ARGFLAG_XOR 按位相或，保存的值将是进行 OR（或）, AND（与）, XOR（异或）操作之后的值。如果 argInfo选项域的值与 POPT_ARGFLAG_NOT 按位相或，保存的值将是取反后的值。对于通常的置位（setting bits）和复位（clearing bits）操作，可以设置 POPT_BIT_SET 或 POPT_BIT_CLR 标志来执行位操作。

如果argInfo选项域的值与 POPT_ARGFLAG_ONEDASH 按位相或，长参数将可以替换两个'-'为单个'-'（即只使用单个连字符'-'），例如，如果 --longopt 是一个设置了 POPT_ARGFLAG_ONEDASH 标志的选项，那么 -longopt 也同样被接受。

下一个选项域 arg ，如果使用（即arg不为NULL），则允许

2009年8月 (5)
2009年7月 (2)
2009年6月 (9)
2009年3月 (4)
2008年10月 (27)
2008年6月 (3)
2008年5月 (1)
2008年4月 (4)
2008年3月 (2)

编程

AstralWind
非常好的python博文
C 语言常见问题集
c/c++博客(chio)
ChinaUnix
CoderZh的技术博客
非常好的python文章，Python天天美味系列
high scalability
HoNooD的Blog
有很多好的C++ on windows文章
ian的个人博客
python、django、爬虫、android
iTech--Python
很全的python参考博客
ITPUB
Just Steps
linux c/c++、shell等编程
linux服务器
linux基础
Linux内核API手册（Kernel API）
Linux内核源码在线阅读（LXR）
Linux设备驱动开发指南
NoSqlFan
redis、mongdb
PHP手册
PHP资源分享门户
python学习blog
Qt - 窦宁波 的专栏
qt博客
Steven.Leong的专栏
wuzhekai的专栏
点点星光（测试）
读书笔记
逛奔的蜗牛(c++/qt)

popt 自动更新程序参数。如果 arg 为 NULL，那么它将被忽略，popt不执行任何额外的操作，否则，它将指向一个上述表格最右列所指示类型的变量。POPT_ARG_ARGV 类型参数将（再）分配一个 char * 字符串指针数组添加字符串参数，并以NULL作为字符数组的结束。POPT_ARG_ARGV 参数的地址 char ** 应初始化为 NULL。

如果选项不带参数（argInfo 选项域的值为 POPT_ARG_NONE，值得注意的是，POPT_ARG_NONE 的值为 0），则选项被启用时，由 arg 指向的变量被设置为 1 。如果选项带有参数（argInfo 不为 POPT_ARG_NONE），选项被启用时，arg 指向的变量将被更新为参数的值。
POPT_ARG_STRING 和 POPT_ARG_ARGV 类型的参数可接受任意的字符串，但 POPT_ARG_INT, POPT_ARG_LONG, POPT_ARG_LONGLONG, POPT_ARG_FLOAT 和 POPT_ARG_DOUBLE 类型参数将会做适当的转换，如果转换失败，则返回错误。

选项信息（argInfo）中如果指定了 POPT_ARG_VAL， 选项域 arg 将被设置为域 val 的值。这通常在允许多个互斥参数同时出现，而你希望最后指定的参数有效的情况下有用。例如， “rm -i -f”。 POPT_ARG_VAL 导致语法分析函数不返回值，因为已经使用了选项域 val 的值。

如果选项域 argInfo 的值与 POPT_ARGFLAG_OPTIONAL 按位或，则该选项的长选项的参数可以省略，如果该长选项不带参数，则保存一个默认值0或NULL（选项域 arg 不为空），否则，等同于带参数的长选项。

下一个选项域 val，是popt语法分析函数在遇到该选项时将要返回的值。如果该选项域为0，语法分析函数将不返回值，而继续分析下一个命令行参数。

最后两个选项域 descrip 和 argDescrip，仅仅在想得到自动帮助信息(automatic help messages)时才需要(生成自动用法信息(automatic usage messages)可以不需要它们)。descrip 是关于参数的文本描述，argdescrip 是关于选项参数的类型的简短概要，如果选项不需要参数，则可设置为NULL。(descrip is a text description of the argument and argdescrip is a short summary of the type of arguments the option expects, or NULL if the option doesn't require any arguments.)

如果 popt 需要自动提供 --usage 和 --help (-?) 选项，选项表中应该有一行是宏POPT_AUTOHELP，这个宏在主选项表中包含进另外一个选项表（通过 POPT_ARG_INCLUDE_TABLE， 见下文）。当 --usage 或 --help 传递给使用了popt自动帮助的应用程序，popt发现该选项时将打印适当的信息(help message)到标准错误，然后退出程序执行并返回 0。如果你希望使用不同的方式生成popt的自动帮助，你需要显式的添加选项条目到你的程序的选项表中，以取代使用 POPT_AUTOHELP。

如果选项域 argInfo 的值与 POPT_ARGFLAG_DOC_HIDDEN 按位或，该参数将不会在帮助信息输出(help output)中显示。

如果选项域 argInfo 的值与 POPT_ARGFLAG_SHOW_DEFAULT 按位或，该参数的初始值将显示在帮助信息输出(help output)中。

选项表中的最后的一个结构体（struct poptOption），所有的指针设置为NULL，所有的算术值设置为0，以标记选项表的结束。宏 POPT_TABLEEND 提供了这个实现。

有两种类型的选项表项（entry）不指定命令行选项，不管是其中的哪一种类型的项，选项域 longName 必须为NULL，且 shortName 必须为 '\0'。

这两种表项类型中第一种类型允许程序在当前的选项表中嵌套另外一个选项表，这种嵌套可以扩展的很深（实际的深度为程序的栈大小所限制）。包含进其他选项表，则允许一个库（library）为任何使用它的程序提供一个标准的命令行选项集合（例如，这通常在图形程序设计工具包中实现）。为实现这个功能，需要设置选项域 argInfo 为 POPT_ARG_INCLUDE_TABLE，而选项域 arg 则指向了将要被包含进来的选项表。如果使用了自动生成帮助信息的功能，则选项域 descrip 需要包含将要被包含进来的选项表的所有描述信息。

另外的一种特别的选项表项类型则告诉popt在遇到该选项表中的选项时调用一个函数（一个回调），这在包含一个正在使用的选项表时尤其有用，因为提供顶级(top-level)选项表的程序不需要知道包含进来的选项表所提供的选项。如果为选项表设置了回调函数，语法分析函数则不返回选项表中选项的信息，而是通过回调函数保存选项信息或让popt通过选项域 arg 设置一个变量。选项回调函数应匹配下面的原型：

```
void poptCallbackType(poptContext con,
                      const struct poptOption * opt,
                      const char * arg, void * data);
```

第一个参数是将要进行分析的上下文(context)（请查看下一节关于上下文的信息），opt 指向触发该回调函数的选项， arg 是选项的参数，如果该选项不带参数，则 arg 为NULL，最后的一个参数 data 从定义该回调函数的选项的选项域 decript 获取，这允许给回调函数传入一个任意的数据集合（尽管要使用类型转换）

定义回调函数的选项表项，选项域 argInfo 值为 POPT_ARG_CALLBACK, 选项域 arg 则指向回调函数，而选项域 descrip 则指定一个将要传给回调函数的任意指针。

2. 创建一个上下文

popt 可以交错分析多个命令行选项集合，这通过为每一个特定的命令行参数集合保存其所有的状态信息到一个 poptContex 数据结构来实现的，poptContex 数据结构是一个不透明类型（opaque type），不能在popt库之外修改。（即实现了信息隐藏，接口与数据分离）
可以通过 poptGetContext() 函数创建一个 popt 上下文：

```
poptContext poptGetContext(const char * name, int argc,
```

```
const char ** argv,
const struct poptOption * options,
int flags);
```

第一个参数 name 仅仅用作别名处理，应该设置为程序（其选项正在被分析）的名称或者为NULL（如果不需要选项别名）。往后的两个参数指定了将要分析的命令行参数，传给poptGetContext()的它们通常等同于传给程序的main()函数的参数。options 参数指向命令行选项表（上一节描述）。最后的一个参数 flag, 可以从以下的三个值中选择：

Value	Description
POPT_CONTEXT_NO_EXEC	Ignore exec expansions
POPT_CONTEXT_KEEP_FIRST	Do not ignore argv[0]
POPT_CONTEXT_POSIXMEHARDER	Options cannot follow arguments

一个 poptContex 跟踪记录了哪些选项已经被分析和哪些选项仍未分析。如果程序希望重新开始一个参数集合的处理过程，可以通过把该上下文(contex)作为唯一的参数传给poptResetContext().

如果参数处理完成了，进程应该释放 poptContex, 因为它包含了动态分配的组成部分。poptFreeContext() 函数使用poptContext 作为唯一的参数释放该上下文所使用的资源。

下面是poptResetContext()和poptFreeContext()的原型：

```
#include <popt.h>
void poptFreeContext(poptContext con);
void poptResetContext(poptContext con);
```

3. 分析命令行

应用程序创建一个 poptContext 之后，就将开始分析参数，由poptGetNextOpt()函数完成实际的参数分析。

```
#include <popt.h>
int poptGetNextOpt(poptContext con);
```

使用contex作为唯一的参数，这个函数分析命令行上的参数。当找到选项表中的下一个参数，如果改选项的选项域arg指针不为空，则填写arg指针所指向的对象。如果该选项的选项域val的值不为0，函数将返回该值，否则，poptGetNextOpt()继续分析下一个参数。

当最后一个参数分析完成，poptGetNextOpt()返回-1，当出现错误时，返回其他的负值。因此，设置选项的选项域val的值大于0会是一个好的做法。

如果所有的命令行选项都是通过选项域arg指针(不为NULL)处理，命令行分析将简化到如下的一行代码：

```
rc = poptGetNextOpt(poptcon);
```

然而，许多应用程序需要比这个更复杂的命令行分析，则可以使

用如下的语句结构：

```
while ((rc = poptGetNextOpt(poptcon)) > 0) {
    switch (rc) {
        /* specific arguments are handled here */
    }
}
```

当选项处理返回，如果应用程序需要知道指定在选项之后的任意参数的值（即获取选项的参数值），有两种方法可以发现它们。一种方法是请求popt填写该参数值到选项域arg指向的变量，另一种方法是使用poptGetOptArg()函数：

```
#include <popt.h>
char * poptGetOptArg(poptContext con);
```

这个函数返回提供给poptGetNextOpt()返回的最后一个选项的参数，或者返回NULL，如果没有指定任何参数。调用函数负责释放该字符串。（这段的翻译有点纠结，个人理解即是返回选项的参数值，不论是通过arg指针还是上述的poptGetOptArg()函数，附原文）

When returned options are handled, the application needs to know the value of any arguments that were specified after the option. There are two ways to discover them. One is to ask popt to fill in a variable with the value of the option through the option table's arg elements. The other is to use poptGetOptArg():

```
#include <popt.h>char * poptGetOptArg(poptContext con);
```

This function returns the argument given for the final option returned by poptGetNextOpt(), or it returns NULL if no argument was specified. The calling function is responsible for deallocating this string.

4. 剩余参数(leftover arguments)

许多应用程序接受任意数量的命令行参数，例如一个文件名列表。当popt遇到一个不是以一个 - 开始的参数，它(popt)将认为它(参数)是个剩余参数(leftover argument)，并把它(参数)添加到一个剩余参数列表。有三个函数允许应用程序访问这些参数：

```
const char * poptGetArg(poptContext con);
```

这个函数返回下一个剩余参数，并标记它已处理。

```
const char * poptPeekArg(poptContext con);
```

返回下一个剩余参数，但不标记它已处理，这允许应用程序继续读取参数列表而不修改它。

```
const char ** poptGetArgs(poptContext con);
```

所有的剩余参数以等同于argv的方式返回，返回数组的最后一个元素指向NULL，表明参数的结尾。

5. 自动帮助信息

popt库能够自动生成描述程序所接受的选项的信息。可以生成有两种类型的帮助信息，用法(usage)信息是一个简短的有效选项列表（没有描述），帮助(help)信息是描述每一个选项的一行（或多行）文本，需要为每一个选项填写的选项域descrip和argDescrip。
使用宏 POPT_AUTOHELP 可以很方便的添加 --usage 和 --help 信息到你的应用程序，关于POPT_AUTOHELP，本帮助文档的第一部分已讲述。如果需要更好的控制你的程序的帮助信息，可以使用下面的两个函数：

```
#include <popt.h>
void poptPrintHelp(poptContext con, FILE * f, int flags);
void poptPrintUsage(poptContext con, FILE * f, int flags);
```

poptPrintHelp() 打印标准的帮助(help)信息到标准输入输出文件描述符 f，而 poptPrintUsage() 则打印简短的用法信息。两个函数当前都忽略 falgs 参数，该参数是为了以后扩展使用。

错误处理

。。。
（待续～ 任务还很艰巨）

EXAMPLE

下面的例子是程序"robin"的一个简单版本。这个程序(Robin)已经裁剪掉除了参数分析逻辑部分外的所有东西，经过稍微的修改，并重命名为"parse"。
这在举例说明功能强大的popt库的一些特征将是十分有用的。

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

void usage(poptContext
optCon, int exitcode, char *error, char *addl) {
    poptPrintUsage(optCon, stderr, 0);
    if (error) fprintf(stderr, "%s: %s\n", error, addl);
    exit(exitcode);
}

int main(int argc, char *argv[]) {
    int     c;          /* used for argument parsing */
    int     i = 0;       /* used for tracking options */
    int     speed = 0;    /* used in argument parsing to set speed
*/
    int     raw = 0;      /* raw mode? */
    int     j;
    char    buf[BUFSIZ+1];
```

```
    const char *portname;
    poptContext optCon;   /* context for parsing command-line
options */

    struct poptOption optionsTable[] = {

    { "bps", 'b', POPT_ARG_INT, &speed, 0,
            "signaling rate in bits-per-second", "BPS" },
    { "crnl", 'c', 0, 0, 'c',
        "expand cr characters to cr/lf sequences", NULL },
    { "hwflow", 'h', 0, 0, 'h',
        "use hardware (RTS/CTS) flow control", NULL },
    { "noflow", 'n', 0, 0, 'n',
        "use no flow control", NULL },
    { "raw", 'r', 0, &raw, 0,
            "don't perform any character conversions", NULL },
    { "swflow", 's', 0, 0, 's',
        "use software (XON/XOF) flow control", NULL } ,
    POPT_AUTOHELP
    { NULL, 0, 0, NULL, 0 }
};


optCon = poptGetContext(NULL, argc, (const char **)argv, opti
onsTable, 0);
    poptSetOtherOptionHelp(optCon, "[OPTIONS]* <port>");

    if (argc < 2) {
        poptPrintUsage(optCon, stderr, 0);
        exit(1);
    }

    /* Now do options processing, get portname */
    while ((c = poptGetNextOpt(optCon)) >= 0) {
        switch (c) {

        case 'c':
            buf[i++] = 'c';
            break;
        case 'h':
            buf[i++] = 'h';
            break;
        case 's':
            buf[i++] = 's';
            break;
        case 'n':
            buf[i++] = 'n';
            break;
        }
    }
```

```
    portname = poptGetArg(optCon);
    if((portname == NULL) || !(poptPeekArg(optCon) == NULL))
        usage(optCon, 1, "Specify a single port", ".e.g.,
/dev/cua0");

    if (c < -1) {
        /* an error occurred during option processing */
        fprintf(stderr, "%s: %s\n",

poptBadOption(optCon, POPT_BADOPTION_NOALIAS),
                poptStrerror(c));
        return 1;
    }

    /* Print out options, portname chosen */

    printf("Options  chosen: ");
    for(j = 0; j < i ; j++)
        printf("-%c ", buf[j]);
    if(raw) printf("-r ");
    if(speed) printf("-b %d ", speed);
    printf("\nPortname chosen: %s\n", portname);

    poptFreeContext(optCon);
    exit(0);
}
```

保存代码到文件 parse.c 并编译：

```
jarson@ubuntu:~$ gcc -Wall -o parse parse.c -lpopt

jarson@ubuntu:~$ ./parse
Usage: parse [-chnrs?] [-b|--bps=BPS] [-c|--crnl] [-h|--
hwflow] [-n|--noflow] [-r|--raw]
        [-s|--swflow] [-?|--help] [--usage] [OPTIONS]* <port>

jarson@ubuntu:~$ ./parse -?
Usage: parse [OPTIONS]* <port>
  -b, --bps=BPS     signaling rate in bits-per-second
  -c, --crnl        expand cr characters to cr/lf sequences
  -h, --hwflow      use hardware (RTS/CTS) flow control
  -n, --noflow      use no flow control
  -r, --raw         do not perform any character conversions
  -s, --swflow      use software (XON/XOF) flow control

Help options:
  -?, --help        Show this help message
      --usage       Display brief usage message
```

RPM, a popular Linux package management program, makes heavy use of popt's features.
Many of its command-line arguments are implemented

through popt aliases, which makes
RPM an excellent example of how to take advantage of the popt library. For more
information on RPM, see http://www.rpm.org. The popt source code distribution
includes test program(s) which use all of the features of the popt libraries in vari-
ous ways. If a feature isn't working for you, the popt test code is the first place
to look.

分类: Linux C

一个人的天空@
关注 - 2
粉丝 - 113
+加关注

« 上一篇：VC中字符串由于版本不同而导致的错误问题
» 下一篇：变长参数列表函数

posted on 2012-06-05 11:57 一个人的天空@ 阅读(1496) 评论(0) 编辑 收藏

» 更多新闻...

最新知识库文章:

· 用核心-路径法设计页面
· 在短平快项目中如何减少测试返工
· 想要提高网页转换率？试试这16个UI秘诀
· 天猫浏览型应用的CDN静态化架构演变
· 单元测试本质：面向逻辑块

» 更多知识库文章...