

当前位置: 首页 > 程序阶梯 > C/C++ > 用Google的gflags轻松的编码解析命令行参数

用Google的gflags轻松的编码解析命令行参数

2014年10月26日 | 标签: C/C++ Google gflags Linux | 浏览: 3,528 | 评论关闭

有了上文《用Google的gflags优雅的解析命令行参数》到位的前戏,已经知道gflags是何方“尤物”了。接下来就该深入一下了。

支持的参数类型

gflags支持的类型有bool, int32, int64, uint64, double和string。可以说这些基本类型大体上满足了我们的需求。

1. DEFINE_bool: boolean
2. DEFINE_int32: 32-bit integer
3. DEFINE_int64: 64-bit integer
4. DEFINE_uint64: unsigned 64-bit integer
5. DEFINE_double: double
6. DEFINE_string: C++ string

比如上文中,我就定义了confPath, port, daemon三个命令行参数,回顾一下:

```
1 DEFINE_string(confPath, "../conf/setup.ini", "program configure file.");
2 DEFINE_int32(port, 9090, "program listen port");
3 DEFINE_bool(daemon, true, "run daemon mode");
```

稍微讲解一下:

- 第一个字段 confPath就是命令行里要输入的参数名,比如 -confPath=../love.ini
- 第二个字段“../conf/setup.ini”,就是如果命令行里没指定这个参数,那默认值就是 ../conf/setup.ini
- 第三个字段“program configure file.”,就是这个参数的帮助说明信息,当用户输入 -hlep 的时候,会显示出来。

代码中使用这个变量

以前我们使用getopt_long函数来自己解析命令行参数的时候,都得内部定义一个变量来保存从命令行得到的值。后续就可以使用这个变量来完成相应的代码逻辑。那其实,DEFINE_string等“指令”就相当于定义了变量,只不过变量名多了个前缀“FLAGS_”。即,我们可以在代码里面直接操作FLAGS_confPath, FLAGS_port, FLAGS_port, FLAGS_daemon这三个变量。

解析命令行参数

gflags是使用ParseCommandLineFlags这个方法来完成命令行参数的解析的。具体如下:

```
1 gflags::ParseCommandLineFlags(&argc, &argv, true);
```

一目了然,唯一值得注意的就是第三个参数了。如果设置为true, gflags就会移除解析过的参数。即argc, argv就会变了。否则gflags还会保持这些参数继续留在argc, argv中。但是参数的顺序有可能会发生变化。

欢迎订阅本站



热门文章

随机推荐

- c/c++(hiredis)批量请求redis
- 阿里云(CentOS 6.3)安装MySQL + PHP + ...
- 小试MySQL的Hash分区
- 阿里云(CentOS 6.3)安装Nginx
- 笔试题: 旋转数组中查找某一个元素
- cmake自动生成protobuf代码
- C++非递归方式实现二叉树
- 巧用redis的有序集合实现一个高效订单...
- 两个房间、三个开关和三盏灯
- 巧用Linux下date命令互转时间戳和指定...

读者墙

标签云

C/C++ CentOS 6.3 cMake

Google gflags hello world hiredis hostname
Linux Linux登陆欢迎语 man man-pages min()
MySQL MySQL开机自动启动 Nginx Nginx
开机自动启动 No manual entry pcre PHP php-
fpm开机自动启动 pop() protobuf push()
redis SecureCRT socket ssh WordPress
wordpress主题 yum ZeroMQ 二叉排序树
二叉树 微信面试 技术宅 描述符 数据结构
旋转数组二分查找 时间复杂度 栈 程序员
面试 算法 腾讯面试 递归 阿里云

如果不好理解的话，没关系，来一段代码就明白什么意思了。

```
1 #include <iostream>
2 #include <gflags/gflags.h>
3
4 using namespace std;
5
6 DEFINE_string(confPath, "../conf/setup.ini", "program configure file");
7 DEFINE_int32(port, 9090, "program listen port");
8 DEFINE_bool(daemon, true, "run daemon mode");
9
10 int main(int argc, char** argv)
11 {
12     for (int i = 0; i < argc; i++) {
13         printf("argv[%d] = %s\n", i, argv[i]);
14     }
15     printf("-----here-----\n");
16
17     gflags::SetVersionString("1.0.0.0");
18     gflags::SetUsageMessage("Usage : ./demo ");
19     gflags::ParseCommandLineFlags(&argc, &argv, true);
20
21     for (int i = 0; i < argc; i++) {
22         printf("argv[%d] = %s\n", i, argv[i]);
23     }
24     printf("-----there-----\n");
25
26     cout << "confPath = " << FLAGS_confPath << endl;
27     cout << "port = " << FLAGS_port << endl;
28
29     if (FLAGS_daemon) {
30         cout << "run background ..." << endl;
31     }
32     else {
33         cout << "run foreground ..." << endl;
34     }
35
36     cout << "good luck and good bye!" << endl;
37
38     gflags::ShutDownCommandLineFlags();
39     return 0;
40 }
```

运行后，看一下true的情况：

```
1 [amcool@leoox build]$ ./demo --port=8888 --confPath=./happy.ini --da
2 argv[0] = ./demo
3 argv[1] = --port=8888
4 argv[2] = --confPath=./happy.ini
5 argv[3] = --daemon
6 -----here-----
7 argv[0] = ./demo
8 -----there-----
9 confPath = ./happy.ini
10 port = 8888
11 run background ...
12 good luck and good bye!
```

修改为false，在运行一下的情况：

```
1 [amcool@leoox build]$ ./demo --port=8888 --confPath=./happy.ini --da
2 argv[0] = ./demo
3 argv[1] = --port=8888
4 argv[2] = --confPath=./happy.ini
5 argv[3] = --daemon
6 -----here-----
7 argv[0] = ./demo
8 argv[1] = --port=8888
9 argv[2] = --confPath=./happy.ini
10 argv[3] = --daemon
11 -----there-----
12 confPath = ./happy.ini
13 port = 8888
14 run background ...
```

```
15 | good luck and good bye!
```

参数检查

按照以前的习惯，我们可以获取到所有参数的值后，再在代码里面进行判断这个参数是否是我们想要的。比如，我们需要端口是36800 到 36888之间的，那我们可以这样检查。

```
1 | if (FLAGS_port < 36800 || FLAGS_port > 36888) {
2 |     printf("port must [36800, 36888]\n");
3 |     return -1;
4 | }
```

当然gflags里面建议使用 `RegisterFlagValidator` 这个方法来做参数检查。参数不通过的时候，程序是启动失败的。

```
1 | static bool ValidatePort(const char* flagname, gflags::int32 value)
2 |     if (value >= 36800 && value <= 36888) {
3 |         printf("param(%s) = (%d) is valid!\n", flagname, value);
4 |         return true;
5 |     }
6 |
7 |     printf("param(%s) = (%d) is invalid!\n", flagname, value);
8 |     return false;
9 | }
10 | DEFINE_int32(port, 36810, "program listen port");
11 | static const bool validPort = gflags::RegisterFlagValidator(&FLAGS_p
```

运行一下，看看效果：

```
1 | [amcool@leoox build]$ ./demo --port=36889 --confPath=./happy.ini --da
2 | param(port) = (36889) is invalid!
3 | param(port) = (36810) is valid!
4 | ERROR: failed validation of new value '36889' for flag 'port'
```

【疑问】：我们手动指定端口36889不合法，默认的36810合法。怎么让gflags当发现参数不合法的时候，使用合法的默认参数呢？！

其他代码文件使用参数变量

正常来说，我们的代码不可能只有1个cpp，还会有很多模块。而每个模块可能会使用到不同的参数值。所以我们之前在demo.cpp定义的参数变量（比如FLAGS_port），在其他模块怎么引用和使用呢？so easy，与DEFINE相对应的有DECLARE。声明一下，就可以使用了。

1. DECLARE_bool: boolean
2. DECLARE_int32: 32-bit integer
3. DECLARE_int64: 64-bit integer
4. DECLARE_uint64: unsigned 64-bit integer
5. DECLARE_double: double
6. DECLARE_string: C++ string

来一段简单的代码，就一目了然啦。

示例代码目录结构

```
1 | [amcool@leoox demo]$ tree
2 | .
3 | |-- CMakeLists.txt
4 | |-- build
5 | |-- demo.cpp
6 | |-- logic.cpp
7 | `-- logic.h
8 |
```

```
9 1 directory, 4 files
10 [amcool@leoox demo]$
```

logic.h

```
1 #ifndef _LEOOX_LOGIC_H_
2 #define _LEOOX_LOGIC_H_
3
4 #include <iostream>
5 #include <gflags/gflags.h>
6
7 using namespace std;
8
9 DECLARE_string(confPath);
10 DECLARE_int32(port);
11 DECLARE_bool(daemon);
12
13 int process();
14
15 #endif
```

logic.cpp

```
1 #include "logic.h"
2
3 int process()
4 {
5     printf("----- process start -----\\n");
6     cout << "confPath = " << FLAGS_confPath << endl;
7     cout << "port = " << FLAGS_port << endl;
8     if (FLAGS_daemon) {
9         cout << "run background ..." << endl;
10    }
11    else {
12        cout << "run foreground ..." << endl;
13    }
14    printf("----- process end -----\\n");
15
16    return 0;
17 }
```

demo.cpp

```
1 #include "logic.h"
2
3 DEFINE_string(confPath, "../conf/setup.ini", "program configure file")
4
5 DEFINE_bool(daemon, true, "run daemon mode");
6
7 static bool ValidatePort(const char* flagname, gflags::int32 value)
8 {
9     if (value >= 36800 && value <= 36888) {
10         printf("param(%s) = (%d) is valid!\\n", flagname, value);
11         return true;
12     }
13     printf("param(%s) = (%d) is invalid!\\n", flagname, value);
14     return false;
15 }
16
17 DEFINE_int32(port, 36810, "program listen port");
18 static const bool validPort = gflags::RegisterFlagValidator(&FLAGS_p
19
20 int main(int argc, char** argv)
21 {
22     gflags::SetVersionString("1.0.0.0");
23     gflags::SetUsageMessage("Usage : ./demo ");
24     gflags::ParseCommandLineFlags(&argc, &argv, false);
25
26     process();
27
28     cout << "good luck and good bye!" << endl;
29
30     gflags::ShutDownCommandLineFlags();
31     return 0;
32 }
```

```
31 }
```

CMakeLists.txt

```
1 project(demo)
2 cmake_minimum_required(VERSION 2.8)
3 set(CMAKE_VERBOSE_MAKEFILE on)
4
5 include_directories(".")
6 include_directories("/home/leoox/local/gflags-2.1.1/include")
7 link_directories("/home/leoox/local/gflags-2.1.1/lib")
8
9 add_executable(demo demo.cpp logic.cpp)
10 target_link_libraries(demo gflags pthread)
```

运行结果

```
1 [amcool@leoox build]$ ./demo --port=36850 --confPath=./love.ini
2 param(port) = (36850) is valid!
3 param(port) = (36850) is valid!
4 ----- process start -----
5 confPath = ./love.ini
6 port = 36850
7 run background ...
8 ----- process end -----
9 good luck and good bye!
10 [amcool@leoox build]$
```

至此，Google的强大的开源组件之一的“gflags”，就算完成了深入浅出的学习了。自己以后可以把getopt_long深藏功与名了。哈哈。



« 用Google的gflags优雅的解析命令行参数

cmake自动生成protobuf代码 »

作者: leoox

除非注明，本文原创: 狮子牛

如需转载，请联系本站。转载请以链接形式注明本文地址，否则进行追究！

原文链接: <http://www.leoox.com/?p=275>

相关文章

近期热评

最新日志

- Linux gcc编译参数 ---- Werror的威力
- Linux下非ROOT权限安装MySQL
- 巧用Linux下date命令互转时间戳和指定日期
- 用Google的gflags优雅的解析命令行参数
- 小记Linux下安装搭建RSYNC服务器
- expect + scp 免输入密码完成机器间的文件拷贝

留言 联系我

Copyright © 2014-2016 狮子牛 All rights reserved. 登录
Powered by WordPress & Theme by 技术宅. 68 Q in 0.523. 粤ICP备14039403号.

