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。

如果不好理解的话，没关系，来一段代码就明白什么意思了。

```
#include <iostream>
#include <gflags/gflags.h>

using namespace std;

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

int main(int argc, char** argv)
{
  for (int i = 0; i < argc; i++) {
    printf("argv[%d] = %s\n", i, argv[i]);
  }
  printf("---------------here--------------\n");

  gflags::SetVersionString("1.0.0.0");
  gflags::SetUsageMessage("Usage : ./demo ");
  gflags::ParseCommandLineFlags(&argc, &argv, true);

  for (int i = 0; i < argc; i++) {
    printf("argv[%d] = %s\n", i, argv[i]);
  }
  printf("---------------there--------------\n");

  cout << "confPath = " << FLAGS_confPath << endl;
  cout << "port = " << FLAGS_port << endl;

  if (FLAGS_daemon) {
    cout << "run background ..." << endl;
  }
  else {
    cout << "run foreground ..." << endl;
  }

  cout << "good luck and good bye!" << endl;

  gflags::ShutDownCommandLineFlags();
  return 0;
}
```

运行后，看一下true的情况：

```
[amcool@leoox build]$ ./demo --port=8888 --confPath=./happy.ini --da
argv[0] = ./demo
argv[1] = --port=8888
argv[2] = --confPath=./happy.ini
argv[3] = --daemon
---------------here--------------
argv[0] = ./demo
---------------there--------------
confPath = ./happy.ini
port = 8888
run background ...
good luck and good bye!
```

修改为false，在运行一下的情况：

```
[amcool@leoox build]$ ./demo --port=8888 --confPath=./happy.ini --da
argv[0] = ./demo
argv[1] = --port=8888
argv[2] = --confPath=./happy.ini
argv[3] = --daemon
---------------here--------------
argv[0] = ./demo
argv[1] = --port=8888
argv[2] = --confPath=./happy.ini
argv[3] = --daemon
---------------there--------------
confPath = ./happy.ini
port = 8888
run background ...
```

```
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。

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

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。

```
static bool ValidatePort(const char* flagname, gflags::int32 value)
  if (value >= 36800 && value <= 36888) {
    printf("param(%s) = (%d) is valid!\n", flagname, value);
    return true;
  }

  printf("param(%s) = (%d) is invalid!\n", flagname, value);
  return false;
}
DEFINE_int32(port, 36810, "program listen port");
static const bool validPort = gflags::RegisterFlagValidator(&FLAGS_p
```

运行一下，看看效果：

```
[amcool@leoox build]$ ./demo --port=36889 --confPath=./happy.ini --da
param(port) = (36889) is invalid!
param(port) = (36810) is valid!
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

```
[amcool@leoox demo]$ tree
.
|-- CMakeLists.txt
|-- build
|-- demo.cpp
|-- logic.cpp
`-- logic.h

```

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

logic.h

```
#ifndef _LEOOX_LOGIC_H_
#define _LEOOX_LOGIC_H_

#include <iostream>
#include <gflags/gflags.h>

using namespace std;

DECLARE_string(confPath);
DECLARE_int32(port);
DECLARE_bool(daemon);

int process();

#endif
```

logic.cpp

```
#include "logic.h"

int process()
{
  printf("----------------- process start ---------------\n");
  cout << "confPath = " << FLAGS_confPath << endl;
  cout << "port = " << FLAGS_port << endl;
  if (FLAGS_daemon) {
    cout << "run background ..." << endl;
  }
  else {
    cout << "run foreground ..." << endl;
  }
  printf("----------------- process end ---------------\n");

  return 0;
}
```

demo.cpp

```
#include "logic.h"

DEFINE_string(confPath, "../conf/setup.ini", "program configure file

DEFINE_bool(daemon, true, "run daemon mode");

static bool ValidatePort(const char* flagname, gflags::int32 value)
  if (value >= 36800 && value <= 36888) {
    printf("param(%s) = (%d) is valid!\n", flagname, value);
    return true;
  }

  printf("param(%s) = (%d) is invalid!\n", flagname, value);
  return false;
}
DEFINE_int32(port, 36810, "program listen port");
static const bool validPort = gflags::RegisterFlagValidator(&FLAGS_p

int main(int argc, char** argv)
{
  gflags::SetVersionString("1.0.0.0");
  gflags::SetUsageMessage("Usage : ./demo ");
  gflags::ParseCommandLineFlags(&argc, &argv, false);

  process();

  cout << "good luck and good bye!" << endl;

  gflags::ShutDownCommandLineFlags();
  return 0;
```

```
}
```

CMakeLists.txt

```
project(demo)
cmake_minimum_required(VERSION 2.8)
set(CMAKE_VERBOSE_MAKEFILE on)

include_directories(".")
include_directories("/home/leoox/local/gflags-2.1.1/include")
link_directories("/home/leoox/local/gflags-2.1.1/lib")

add_executable(demo demo.cpp logic.cpp)
target_link_libraries(demo gflags pthread)
```

运行结果

```
[amcool@leoox build]$ ./demo --port=36850 --confPath=./love.ini
param(port) = (36850) is valid!
param(port) = (36850) is valid!
----------------- process start ---------------
confPath = ./love.ini
port = 36850
run background ...
----------------- process end ---------------
good luck and good bye!
[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 »

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