用Google的gflags优雅的解析命令行参数 | 狮子牛

leoox.com/

用Google的gflags优雅的解析命令行参数

写了这么多年的Linux下C/C++代码，一直使用getopt_long来解析命令行参数，同时定义一个全局的struct来保存各个命令行参数的值。虽然用得比较“繁琐”，但也安于现状。最近突然发现了Google早在多年前就开源了一个解析命令行参数的“神器”gflags。赶紧来爽一把。

安装

1、去官网下载一个最新的版本(gflags-2.1.1.tar.gz)。

https://github.com/schuhschuh/gflags/archive/v2.1.1.tar.gz

2、现在流行cmake的构建方式，gflags的最新版本也改为使用cmake了。还好我最近也刚刚学习了cmake，算是跟上了潮流。有兴趣的话，可以看 《让cmake显示gcc/g++的编译信息》

C++

```
[amcool@leoox soft]$ tar xzvf gflags-2.1.1.tar.gz
[amcool@leoox soft]$ cd gflags-2.1.1
[amcool@leoox gflags-2.1.1]$ mkdir build
[amcool@leoox gflags-2.1.1]$ cd build/
[amcool@leoox build]$ cmake .. -DCMAKE_INSTALL_PREFIX=/home/amcool/local/gflags-2.1.1
[amcool@leoox build]$ make
[amcool@leoox build]$ make install
```

就是这么简单，安装成功了。值得注意的是，我这里新建了一个build文件夹，即采用“外部构建”的方式。这样编译过程中产生的中间文件（比如.o文件）就都放在build里，不会“污染”gflags源码，做到干干净净。

爽一把

1、既然安装好了，那赶紧来写个简单的代码来爽一把。话不多说，代码才是王道！

demo.cpp

C++

```
// demo.cpp
#include
#include

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)
{
  gflags::ParseCommandLineFlags(&argc, &argv, true);
  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;
}

```

2、很明显，接下来就是要编译了。这里直接用g++写一行命令就可以编译了。但是既然学了cmake，那就“大材小用”一次吧。

CMakeLists.txt

C++

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

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

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

3、那当然就是编译了

C++

```
[amcool@leoox demo]$ ls
CMakeLists.txt  demo.cpp
[amcool@leoox demo]$ mkdir build
[amcool@leoox demo]$ cd build
[amcool@leoox build]$ cmake ..
[amcool@leoox build]$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  Makefile
[amcool@leoox build]$ make
[amcool@leoox build]$ ls
CMakeCache.txt  CMakeFiles  cmake_install.cmake  demo  Makefile
[amcool@leoox build]$
```

设定命令行参数

1、直接运行，得到的就是我们设定的默认参数。（聪明的你，结合代码一看，就知道参数的默认值是什么了）

C++

```
[amcool@leoox build]$ ./demo
confPath = ../conf/setup.ini
port = 9090
run background ...
good luck and good bye!
```

2、设定参数值

i）可以用 –参数名=参数值 或者 -参数名=参数值 的方式来设定参数值。

ii）对于bool类型的参数，除了上述方式外，还可以用 –参数名 的方式设定为true（即不带值），使用 –no参数名 的方式设定为false。为了统一，我建议都使用 上面的 第 i）种方法来设定参数。

C++

```
[amcool@leoox build]$ ./demo --port=8888 --confPath=./setup.ini --daemon=true
confPath = ./setup.ini
port = 8888
run background ...
good luck and good bye!
[amcool@leoox build]$ ./demo -port=8888 -confPath=./setup.ini -daemon=false
confPath = ./setup.ini
port = 8888
run foreground ...
good luck and good bye!
[amcool@leoox build]$ ./demo -port=8888 -confPath=./setup.ini -daemon
confPath = ./setup.ini
port = 8888
run background ...
good luck and good bye!
[amcool@leoox build]$ ./demo -port=8888 -confPath=./setup.ini -nodaemon
confPath = ./setup.ini
port = 8888
run foreground ...
good luck and good bye!
[amcool@leoox build]$
```

3、从文件读入“命令行”参数

如果我们的程序比较牛逼，配置项非常多，也就是说命令行参数很多，那你每次启动都要一个一个的输入，那岂不是很麻烦？gflags已经帮我们解决了，用 –flagfile=命令行文件 的方式就可以了。你接着往下看，就明白了。param.cmd就是上面说的命令行文件。

C++

```
[amcool@leoox build]$ vi param.cmd
--port=8888
--confPath=./setup.ini
--daemon=true
[amcool@leoox build]$ ./demo --flagfile=param.cmd
confPath = ./setup.ini
port = 8888
run background ...
good luck and good bye!
[amcool@leoox build]$
```

怎么样，这样就不怕参数配置错误了吧。保存到文件，每次启动，就很轻松了。

4、从环境变量读入参数值

gflags另外还给我们提供了 –fromenv 和 –tryfromenv 参数，通过这两个参数，我们的程序可以从环境变量中获取到具体的值。两者有什么不一样呢。你看到他们的区别仅仅是有无“try”，聪明的你一定猜到了。

- –fromenv 从环境变量读取参数值 –fromenv=port,confPath 表明要从环境变量读取port,confPath两个参数的值。但是当无法从环境变量中获取到的时候，会报错，同时程序退出。【注意：gflags的变量名是 FLAGS_我们定义的参数名，开篇的代码里，估计细心的你已经发现了】
- –tryfromenv 与–fromenv类似，当参数的没有在环境变量定义时，不退出。

也来一个例子，一看便明了。

C++

```
[amcool@leoox build]$ ./demo --fromenv=port,confPath
ERROR: FLAGS_confPath not found in environment
ERROR: FLAGS_port not found in environment
[amcool@leoox build]$ ./demo --tryfromenv=port,confPath
confPath = ../conf/setup.ini
port = 9090
run background ...
good luck and good bye!
[amcool@leoox build]$ export FLAGS_confPath=./loveyou.ini
[amcool@leoox build]$ export FLAGS_port=36888
[amcool@leoox build]$ env | grep FLAGS
FLAGS_port=36888
FLAGS_confPath=./loveyou.ini
[amcool@leoox build]$
[amcool@leoox build]$ ./demo --fromenv=port,confPath
confPath = ./loveyou.ini
port = 36888
run background ...
good luck and good bye!
[amcool@leoox build]$
```

版本号和帮助信息

我们一般使用程序的时候，都离不开两个参数 –version 和 –help。来看看上面实现的demo能否支持呢？

C++

```
[amcool@leoox build]$ ./demo --version
demo
[amcool@leoox build]$ ./demo --help
demo: Warning: SetUsageMessage() never called

Flags from /home/thrift/program/gflags/demo/demo.cpp:
-confPath (program configure file.) type: string
default: "../conf/setup.ini"
-daemon (run daemon mode) type: bool default: true
-port (program listen port) type: int32 default: 9090
```

哈，help支持了，但是version没支持，而且help信息里面还有waring。没关系，我们可以用 SetVersionString() 和 SetUsageMessage() 方法来满足需求。修改后的代码如下：

【注意：SetVersionString() 和 SetUsageMessage() 一定要在 ParseCommandLineFlags() 之前设定。】

C++

```
#include
#include

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

```

可以来炫一把了：

C++

```
[amcool@leoox build]$ ./demo --version
demo version 1.0.0.0
[amcool@leoox build]$ ./demo --help
demo: Usage : ./demo

  Flags from /home/amcool/program/gflags/demo/demo.cpp:
    -confPath (program configure file.) type: string
      default: "../conf/setup.ini"
    -daemon (run daemon mode) type: bool default: true
    -port (program listen port) type: int32 default: 9090



  Flags from /home/amcool/soft/gflags-2.1.1/src/gflags.cc:
    -flagfile (load flags from file) type: string default: ""
    -fromenv (set flags from the environment [use 'export FLAGS_flag1=value'])
      type: string default: ""
    -tryfromenv (set flags from the environment if present) type: string
      default: ""
    -undefok (comma-separated list of flag names that it is okay to specify on
      the command line even if the program does not define a flag with that
      name.  IMPORTANT: flags in this list that have arguments MUST use the
      flag=value format) type: string default: ""

  Flags from /home/amcool/soft/gflags-2.1.1/src/gflags_completions.cc:
    -tab_completion_columns (Number of columns to use in output for tab
      completion) type: int32 default: 80
    -tab_completion_word (If non-empty, HandleCommandLineCompletions() will
      hijack the process and attempt to do bash-style command line flag
      completion on this value.) type: string default: ""

  Flags from /home/amcool/soft/gflags-2.1.1/src/gflags_reporting.cc:
    -help (show help on all flags [tip: all flags can have two dashes])
      type: bool default: false currently: true
    -helpfull (show help on all flags -- same as -help) type: bool
      default: false
    -helpmatch (show help on modules whose name contains the specified substr)
      type: string default: ""
```

```
    -helpon (show help on the modules named by this flag value) type: string
      default: ""
    -helppackage (show help on all modules in the main package) type: bool
      default: false
    -helpshort (show help on only the main module for this program) type: bool
      default: false
    -helpxml (produce an xml version of help) type: bool default: false
    -version (show version and build info and exit) type: bool default: false
[amcool@leoox build]$
```

简单讲解如何使用gflags进行编码

有了上面的演示和代码展示，想必大家对gflags有了比较直观的认识。做了这么久的前戏，接下来，终于可以深入了解啦。请看下文《用Google的gflags轻松的编码解析命令行参数》。