## 命令行参数解析——gflags

google开源的gflags是一套命令行参数解析工具，比getopt功能更强大，使用起来更加方便，gflags还支持从环境变量、配置文件读取参数（可用gflags代替配置文件）。本文简单介绍gflags的使用，内容主要译自 <http://gflags.googlecode.com/svn/trunk/doc/gflags.html> 。

定义参数   
使用flags需要包含头文件  #include   
gflags主要支持的参数类型包括bool，int32, int64, uint64, double, string等，定义参数通过DEFINE\_type宏实现，如下所示，分别定义了一个bool和一个string类型的参数，该宏的三个参数含义分别为命令行参数名，参数默认值，以及参数的帮助信息。

DEFINE\_bool(big\_menu, true, "Include 'advanced' options in the menu listing");   
DEFINE\_string(languages, "english,french,german",   
                 "comma-separated list of languages to offer in the 'lang' menu");   
gflag不支持列表，用户通过灵活借助string参数实现，比如上述的languages参数，可以类型为string，但可看作是以逗号分割的参数列表。

访问参数   
当参数被定义后，通过FLAGS\_name就可访问到对应的参数，比如上述定义的big\_menu、languages两个参数就可通过FLAGS\_big\_menu、FLAGS\_languages访问。

if (FLAGS\_languages.find("english") != string::npos)   
     HandleEnglish();

以上的访问方式，仅在参数定义和访问在同一个文件（或是通过头文件包含）时，FLAGS\_name才能访问到参数，如果要访问其他文件里定义的参数，则需要使用DECLARE\_type。比如在foo.cc中DEFINE\_string(color, "red", "the color you want to use"); 这是如果你需要在foo\_test.cc中使用color这个参数，你需要加入DECLARE\_string(color, "red", "the color you want to use");

参数检查   
定义参数后，可以给参数注册一个检查函数（validator），当从命令行指定参数或通过SetCommandLineOption()指定参数时，检查函数就会被调用，两个参数分别为命令行参数名，以及设置的参数值。   
static bool ValidatePort(const char\* flagname, int32 value) {   
   if (value > 0 && value < 32768)   // value is ok   
     return true;   
   printf("Invalid value for --%s: %d\n", flagname, (int)value);   
   return false;   
}   
DEFINE\_int32(port, 0, "What port to listen on");   
static const bool port\_dummy = RegisterFlagValidator(&FLAGS\_port, &ValidatePort);   
建议在定义参数后，立即注册检查函数。RegisterFlagValidator()在检查函数注册成功时返回true；如果参数已经注册了检查函数，或者检查函数类型不匹配，返回false。

初始化参数   
在引用程序的main()里通过 google::ParseCommandLineFlags(&argc, &argv, true); 即完成对gflags参数的初始，其中第三个参数为remove\_flag，如果为true，gflags会移除parse过的参数，否则gflags就会保留这些参数，但可能会对参数顺序进行调整。 比如 "/bin/foo" "arg1" "-q" "arg2"  会被调整为 "/bin/foo", "-q", "arg1", "arg2"，这样更好理解。

在命令行指定参数   
比如要在命令行指定languages参数的值，可通过如下4种方式，int32, int64等类型与string类似。

app\_containing\_foo --languages="chinese,japanese,korean"

app\_containing\_foo -languages="chinese,japanese,korean"

app\_containing\_foo --languages "chinese,japanese,korean"

app\_containing\_foo -languages "chinese,japanese,korean"

对于bool类型，则可通过如下几种方式指定参数

app\_containing\_foo --big\_menu

app\_containing\_foo --nobig\_menu

app\_containing\_foo --big\_menu=true

app\_containing\_foo --big\_menu=false

特殊参数

--help 打印定义过的所有参数的帮助信息

--version 打印版本信息 通过google::SetVersionString()指定

--nodefok  但命令行中出现没有定义的参数时，并不退出（error-exit）

--fromenv 从环境变量读取参数值 --fromenv=foo,bar表明要从环境变量读取foo，bar两个参数的值。通过export FLAGS\_foo=xxx; export FLAGS\_bar=yyy 程序就可读到foo，bar的值分别为xxx，yyy。

--tryfromenv 与--fromenv类似，当参数的没有在环境变量定义时，不退出（fatal-exit）

--flagfile 从文件读取参数值，--flagfile=my.conf表明要从my.conf文件读取参数的值。在配置文件中指定参数值与在命令行方式类似，另外在flagfile里可进一步通过--flagfile来包含其他的文件。