gflags 安装及使用

版权说明

代码 & 板书链接

https://gitee.com/bitedu-tech/cpp-chatsystem

1. gflags 介绍

gflags 是 Google 开发的一个开源库，用于 C++ 应用程序中命令行参数的声明、定义和解析。gflags 库提供了一种简单的方式来添加、解析和文档化命令行标志（flags），使得程序可以根据不同的运行时配置进行调整。

它具有如下几个特点：

- **易于使用**：gflags 提供了一套简单直观的 API 来定义和解析命令行标志，使得开发者可以轻松地为应用程序添加新的参数。
- **自动帮助和文档**：gflags 可以自动生成每个标志的帮助信息和文档，这有助于用户理解如何使用程序及其参数。
- **类型安全**：gflags 支持多种数据类型的标志，包括布尔值、整数、字符串等，并且提供了类型检查和转换。
- **多平台支持**：gflags 可以在多种操作系统上使用，包括 Windows、Linux 和 macOS。
- **可扩展性**：gflags 允许开发者自定义标志的注册和解析逻辑，提供了强大的扩展性。

官方文档： https://gflags.github.io/gflags/

代码仓库：https://github.com/gflags/gflags.git

2. gflags 安装

直接命令安装：

```C++
dev@dev-host:~/workspace/gflags$ sudo apt-get install libgflags-dev
```

原码安装：

```Shell
# 下载源码
git clone https://github.com/gflags/gflags.git
# 切换目录
cd gflags/
```

```
mkdir build
cd build/
# 生成 Makefile
cmake ..
# 编译代码
make
# 安装
make install
```

```
Install the project...
-- Install configuration: "Release"
-- Installing: /usr/local/lib/libgflags.a
-- Installing: /usr/local/lib/libgflags_nothreads.a
-- Installing: /usr/local/include/gflags/gflags.h
-- Installing: /usr/local/include/gflags/gflags_declare.h
-- Installing: /usr/local/include/gflags/gflags_completions.h
-- Installing: /usr/local/include/gflags/gflags_gflags.h
-- Installing: /usr/local/lib/cmake/gflags/gflags-config.cmake
-- Installing: /usr/local/lib/cmake/gflags/gflags-config-version.cmake
-- Installing: /usr/local/lib/cmake/gflags/gflags-targets.cmake
-- Installing: /usr/local/lib/cmake/gflags/gflags-targets-release.cmake
-- Installing: /usr/local/lib/cmake/gflags/gflags-nonamespace-targets.cmake
-- Installing: /usr/local/lib/cmake/gflags/gflags-nonamespace-targets-release.cmake
-- Installing: /usr/local/bin/gflags_completions.sh
-- Installing: /usr/local/lib/pkgconfig/gflags.pc
-- Installing: /root/.cmake/packages/gflags/e5f7ce61772240490d3164df06f58ce9
```

至此，gflags 安装完毕。

3. gflags 使用

3.1 包含头文件

使用 gflags 库来定义/解析命令行参数必须包含如下头文件

```
C++
#include <gflags/gflags.h>
```

3.2 定义参数

利用 gflag 提供的宏定义来定义参数。该宏的 3 个参数分别为命令行参数名，参数默认值，参数的帮助信息。

```
C++
DEFINE_bool(reuse_addr, true, "是否开始网络地址重用选项");
DEFINE_int32(log_level, 1, "日志等级：1-DEBUG, 2-WARN, 3-ERROR");
DEFINE_string(log_file, "stdout", "日志输出位置设置，默认为标准输出
");
```

gflags 支持定义多种类型的宏函数：

```
C++
DEFINE_bool
DEFINE_int32
DEFINE_int64
DEFINE_uint64
DEFINE_double
DEFINE_string
```

3.3 访问参数

我们可以在程序中通过 `FLAGS_name` 像正常变量一样访问标志参数。比如在上面的例子中，我们可以通过 `FLAGS_big_menu` 和 `FLAGS_languages` 变量来访问命令行参数。

3.4 不同文件访问参数

如果想再另外一个文件访问当前文件的参数，以参数 `FLAGS_big_menu` 为例，我们可以使用用宏 `DECLARE_bool(big_menu)`来声明引入这个参数。

其实这个宏就相当于做了 `extern FLAGS_big_menu`， 定义外部链接属性。

3.5 初始化所有参数

当我们定义好参数后，需要告诉可执行程序去处理解析命令行传入的参数，使得 `FLAGS_*`变量能得到正确赋值。我们需要在 main 函数中，调用下面的函数来解决命令行传入的所有参数。

```
C++
google::ParseCommandLineFlags(&argc, &argv, true);
```

- `argc` 和 `argv` 就是 main 的入口参数
- 第三个参数被称为 `remove_flags`。如果它为 `true`, 表示 `ParseCommandLineFlags` 会从 `argv` 中移除标识和它们的参数，相应减少 `argc` 的值。如果它为 `false`,`ParseCommandLineFlags` 会保留 `argc` 不变，但将会重新调整它们的顺序，使得标识再前面。

3.6 运行参数设置

gflags 为我们提供了多种命令行设置参数的方式。

string 和 int 设置参数

```Shell
exec --log_file="./main.log"
exec -log_file="./main.log"
exec --log_file "./main.log"
exec -log_file "./main.log"
```

bool 设置参数

```Shell
exec --reuse_addr
exec --noreuse_addr
exec --reuse_addr=true
exec --reuse_addr=false
```

`--`将会终止标识的处理。比如在 `exec -f1 1 -- -f2 2` 中， `f1` 被认为是一个标识，但 `f2` 不会

3.7 配置文件的使用

配置文件的使用，其实就是为了让程序的运行参数配置更加标准化，不需要每次运行的时候都手动收入每个参数的数值，而是通过配置文件，一次编写，永久使用。

需要注意的是，配置文件中选项名称必须与代码中定义的选项名称一致。

样例：

```C++
-reuse_addr=true,
-log_level=3
-log_file=./log/main.log
```

3.8 特殊参数标识

gflags 也默认为我们提供了几个特殊的标识。

```Shell
--help  # 显示文件中所有标识的帮助信息
--helpfull  # 和-help 一样，帮助信息更全面一些
--helpshort  # 只显示当前执行文件里的标志
--helpxml  # 以 xml 方式打印，方便处理
--version  # 打印版本信息，由 google::SetVersionString()设定
```

```
--flagfile  -flagfile=f #从文件 f 中读取命令行参数
```

4. 入门案例

样例编写：

编写样例代码： main.cc

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

DEFINE_bool(reuse_addr, true, "是否开始网络地址重用选项");
DEFINE_int32(log_level, 1, "日志等级：1-DEBUG, 2-WARN, 3-ERROR");
DEFINE_string(log_file, "stdout", "日志输出位置设置，默认为标准输出
");

int main(int argc, char* argv[])
{
    google::ParseCommandLineFlags(&argc, &argv, true);
    std::cout << "reuse: " << FLAGS_reuse_addr << std::endl;
    std::cout << "reuse: " << FLAGS_log_level << std::endl;
    std::cout << "reuse: " << FLAGS_log_file << std::endl;
    return 0;
}
```

配置文件编写：main.conf

```
C++
-reuse_addr=true
-log_level=3
-log_file=./log/main.log
```

Makefile 编写：

```
Shell
main : main.cc
    g++ -std=c++17 $^ -o $@ -lgflags
```

样例运行：

运行代码:

```C++
dev@dev-host:~/workspace/gflags$ ./main --help
Flags from main.cc:
    -log_file (日志输出位置设置，默认为标准输出) type: string
      default: "stdout"
    -log_level (日志等级：1-DEBUG, 2-WARN, 3-ERROR) type: int32
default: 1
    -reuse_addr (是否开始网络地址重用选项) type: bool default: true
```

运行代码 2:

```Shell
dev@dev-host:~/workspace/gflags$ ./main
reuse: 1
reuse: 1
reuse: stdout
```

运行代码 3:

```Shell
dev@dev-host:~/workspace/gflags$ ./main --log_level=2 -- --
log_file=./log
reuse: 1
reuse: 2
reuse: stdout
dev@dev-host:~/workspace/gflags$
```

运行代码 3:

```Shell
dev@dev-host:~/workspace/gflags$ ./main -flagfile=./main.conf
reuse: 1
reuse: 3
reuse: ./log/main.log
```