


[返回博客列表](#)

原 linux下c++编程环境搭建，运行过程以及调试

 memristor

发布时间: 2014/04/27 18:37 阅读: 8479 收藏: 12 点赞: 2 评论: 4



摘要

linux下c++编程环境搭建，运行过程以及调试

安装g++环境

安装两个RPM包即可搞定

```
[root@localhost Desktop]# rpm -ivh /home/weiwei/Desktop/libstdc++-devel-4.4.5-6.el6.i686.rpm
[root@localhost Desktop]# rpm -ivh /home/weiwei/Desktop/gcc-c++-4.4.5-6.el6.i686.rpm
```

查看g++是否安装成功

```
[root@localhost Desktop]# g++ -v
Using built-in specs.
Target: i686-redhat-linux
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --wi
Thread model: posix
gcc version 4.4.5 20110214 (Red Hat 4.4.5-6) (GCC)
```

gcc与g++的区别

gcc可以用来编译C或者C++,但他只能编译c++源文件，不能自动和C++程序使用的库连接，g++可以实现C++程序的编译和链接，其实他也是调用gcc来编译的,要编译c++代码生成可执行文件要用 g++

编写一个简单的c++程序

```
// myfirst.cpp--displays a message

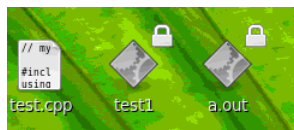
#include <iostream>          // make definitions visible
using namespace std;
int main()                  // function header
{                            // start of function body
    cout << "Come up and C++ me some time."; // message
    cout << endl;           // start a new line
    cout << "You won't regret it!" << endl; // more output
    return 10;              // terminate main() 返回值为0代表成功，非0返回值的含义由系
                             // end of function body
}
```

打开命令窗口进行编译

```
[root@localhost Desktop]# g++ -o test1 test.cpp
```

-o test1 test.cpp 从test.cpp编译生成test1文件，test1为可执行文件，没有后缀

如果不写-o test1 会默认生成一个a.out可执行文件



执行可执行文件

```
[root@localhost Desktop]# ./a.out
Come up and C++ me some time.
You won't regret it!
```

获取main函数返回值

```
[root@localhost Desktop]# echo $?
10
```

c/c++运行流程分解

预处理阶段

对c源文件预处理生成中间文件e.i

```
[root@localhost c]# g++ -E funcuse.c -o e.i
```

编译阶段

对预处理文件进行处理生成汇编语言文件e.s

```
[root@localhost c]# g++ -S e.i -o e.s
```

上述两部可以直接合并为

```
[root@localhost c]# g++ -s e.i -o e.s
```

汇编阶段

生成目标文件，目标文件是机器代码，但不能执行，必须将目标文件与其他目标文件或库文件连接生成可执行的二进制文件才能执行

```
[root@localhost c]# g++ -c e.s -o e.o
```

生成执行文件

```
[root@localhost c]# g++ e.o -o result
```

运行result

```
[root@localhost c]# ./result
```

在linux操作系统中运行程序必须指定程序所在的目录，除非程序的目录已经列在PATH环境变量中，所以程序前必须加./

注：echo \$? 显示main函数的返回值（int型）

如果想让编译和运行同时进行可以采用如下命令：

```
gcc funcuse.c -o result && ./result
```

&&表示如果成功。。。如果编译成功，会直接运行程序

可以将上述所有步骤合并写为

```
g++ funcuse.c -o result
```

或

```
g++ -o result funcuse.c
```

直接生成可执行文件

头文件与源文件

程序如果复杂的话，程序的各个部分会分别存储在不同的文件中，按照逻辑进行划分。

来自：<http://www.cnblogs.com/lidabo/archive/2012/04/17/2454568.html>

头文件的作用就是被其他的.cpp包含,本身并不参与编译,但实际上它们的内容却在多个.cpp文件中得到了编译.

头文件中应该只放变量和函数的声明,而不能放它们的定义

这个规则是有三个例外的

1. 头文件中可以写const对象的定义
2. 头文件中可以写内联函数 (inline) 的定义
3. 头文件中可以写类 (class) 的定义

分离式编译

如果将程序分成若干子程序,怎样在linux下进行编译呢?

下面以求圆的面积为例来说明

Circle.h

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
    private:
        double r;
    public:
        Circle();
        Circle(double R);
        double Area();
};

#endif
```

Circle.cpp

```
#include "Circle.h"
#include <iostream>
using namespace std;

Circle::Circle()
{
    this->r=5.0;
}

Circle::Circle(double R)
{
    this->r=R;
}

double Circle::Area()
{
    return 3.14*r*r;
}
```

main.cpp

```
#include "Circle.h"
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
    Circle c(3);
    cout<<"Area="<<c.Area()<<endl;

    return 0;
}
```



编译

```
[root@localhost cpp]# g++ -c Circle.cpp -o Circle.o
[root@localhost cpp]# g++ -c main.cpp -o main.o
[root@localhost cpp]# g++ main.o Circle.o -o main
[root@localhost cpp]# ./main
Area=28.26
```

-c命令表示编译，头文件不显示式编译，但实际已经编译。如果只修改了一个源文件，只需要编译改动的文件

makefile

但如果我们的程序有几百个源程序的时候，怎么办？难道也要编译器重新一个一个的编译？

makefile关系到了整个工程的编译规则。一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为makefile就像一个Shell脚本一样，其中也可以执行操作系统的命令。

makefile带来的好处就是——“自动化编译”，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率

```
#此行为注释
main: main.o Circle.o
    g++ main.o Circle.o -o main
Circle.o: Circle.cpp
    g++ -c Circle.cpp -o Circle.o
main.o: main.cpp
    g++ -c main.cpp -o main.o
```

注意：g++命令开头的行前面必须有tab空格，不然会报错：*** missing separator. Stop

如果将名字命名为Makefile或makefile，只需要在命令行下敲入make就可以进行自动化编译

```
[root@localhost cpp]# make
g++ -c main.cpp -o main.o
g++ -c Circle.cpp -o Circle.o
g++ main.o Circle.o -o main
[root@localhost cpp]# ./main
Area=28.26
```

参考：

http://blog.163.com/dong_box/blog/static/2625977820103310933870/

[精华] 跟我一起写 Makefile - ChinaUnix.net

编写Makefile - 学习，思考，记录，分享。 - 博客频道 - CSDN.NET

GDB调试

启动gdb

```
[root@localhost c]# gdb
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-48.el6)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
```

调试前要先进行编译连接

```
[root@localhost c]# g++ -g funcuse.c -o debug
```

进行调试

```
[root@localhost c]# gdb debug
Reading symbols from /home/weiwei/Desktop/c/debug...done.
```

列出代码

```
(gdb) list
1      #include<stdio.h>
2
3      int main(){
4          for(int i=0 ; i<5; i++){
5              printf("this is %d\n",i);
6          }
7          return 0;
8      }
```

```
(gdb) list 3,5
3      int main(){
4          for(int i=0 ; i<5; i++){
5              printf("this is %d\n",i);
```

执行程序

```
(gdb) run
Starting program: /home/weiwei/Desktop/c/dbug
this is 0
this is 1
this is 2
this is 3
this is 4
```

设置断点

```
(gdb) break 5
Breakpoint 1 at 0x8048487: file funcuse.c, line 5.
```

运行

```
(gdb) r
Starting program: /home/weiwei/Desktop/c/dbug

Breakpoint 1, main () at funcuse.c:5
5      printf("this is %d\n",i);
Missing separate debuginfos, use: debuginfo-install glibc-2.12-1.25.el6.i686 libgcc-4.4.5-6.el6.i686
```

到断点处程序停止，继续运行输入continue

```
(gdb) c
Continuing.
this is 0

Breakpoint 1, main () at funcuse.c:5
5      printf("this is %d\n",i);
(gdb) c
Continuing.
this is 1

Breakpoint 1, main () at funcuse.c:5
5      printf("this is %d\n",i);
(gdb) c
Continuing.
this is 2
```

监测某一个值

```
(gdb) watch i
Hardware watchpoint 2: i
```

```
(gdb) c
Continuing.
this is 3
Hardware watchpoint 2: i

Old value = 3
New value = 4
0x080484a0 in main () at funcuse.c:4
4      for(int i=0 ; i<5; i++){
```

查看某一个特定的变量

```
(gdb) print i
```

自动显示变量的值，每次运行到断点处都会自动显示

```
(gdb) display i
```

查看当前自动显示的所有变量

```
(gdb) info display
Auto-display expressions now in effect:
Num Enb Expression
1:   y i
```

查看变量类型

```
(gdb) whatis i
type = int
```

单步执行，step进入函数内部(使用return命令跳出来，跳出前可以使用finish执行完函数体)，next把函数当成一条语句不进入函数内部

```
(gdb) step
(gdb) next
```

删除编号为1的断点

```
(gdb) delete 1
```

© 著作权归作者所有

分类: c/c++ 字数: 1967 标签: linux c++ 环境配置 运行过程 调试 gdb g++



点赞 (2)



收藏 (12)



分享



memristor

[关注此人](#)

粉丝: 28

博客数: 170

共码了 158504 字



评论 (4)



智慧IT生活

1楼 2015/10/14 09:57

👍 不错，特别适合像我这样的初学者。



林凤g

2楼 2016/01/11 17:20

发



林凤g

3楼 2016/01/11 17:21

讲的好详细，条例清楚，感谢！



Kkidou

4楼 2016/05/03 17:38

谢谢，讲的很清楚，适合自己学习。

[登录后评论](#)