

首页 > 精品文库 > Linux 程序开发打印 Debug 信息的使用技巧--C语言中几种输出调试信息的方法

**Linux 程序开发打印 Debug 信息的使用技巧--C语言中几种输出调试信息的方法**

[摘要：Linux 顺序开辟挨印 Debug 疑息的应用技能 若何清楚明晰天挨印出顺序疑息，能够快速判别顺序运转环境，定位顺序出题目的中央。先去一段代码真战下再道： #include stdio . h #include stdli]

Linux 程序开发打印 Debug 信息的使用技巧

如何清晰明了地打印出程序信息，可以快速判断程序运行情况，定位程序出问题的地方。先来一段代码实战下再说：

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <errno.h>
4.
5. #ifndef __USE_DEBUG
6. #define __USE_DEBUG
7.
8. #define USE_DEBUG
9. #ifdef USE_DEBUG
10. #define DEBUG_LINE() printf("[%s:%s] line=%drn",__FILE__, __func__, __LINE__)
11. #define DEBUG_ERR(fmt, args...) printf("33[46;31m[%s:%d]33[0m "#fmt" errno=%d, %mrn",
    __func__, __LINE__, ##args, errno, errno)
12. #define DEBUG_INFO(fmt, args...) printf("33[33m[%s:%d]33[0m "#fmt"rn", __func__, __LINE__,
    ##args)
13. #else
14. #define DEBUG_LINE()
15. #define DEBUG_ERR(fmt,...)
16. #define DEBUG_INFO(fmt,...)
17. #endif
18.
19. #endif
20.
21. void func()
22. {
23.     DEBUG_LINE();
24.     DEBUG_INFO("Garfield test DEBUG_INFO() d: %d ; s: %s", 1, __FUNCTION__);
25.     DEBUG_ERR("Garfield test DEBUG_ERR() d: %d ; s: %s", 2, __FUNCTION__);
26. }
27.
28. int main(int argc,char**argv)
29. {
30.     func();
31.
```

最新专题

21 / 07月	<a href="#">android游戏开发教程</a>
21 / 07月	<a href="#">webos手机</a>
21 / 07月	<a href="#">重庆ios培训</a>
21 / 07月	<a href="#">ios开发教程</a>
21 / 07月	<a href="#">android应用开发实战</a>

IThao123周刊

14 / 09月	<a href="#">IThao123互联网周刊，互联网资讯不错...</a>
08 / 09月	<a href="#">IThao123互联网周刊，互联网资讯不错...</a>
02 / 09月	<a href="#">IThao123互联网周刊，互联网资讯不错过(...</a>
26 / 08月	<a href="#">IThao123互联网周刊，互联网资讯不错...</a>
17 / 08月	<a href="#">IThao123互联网周刊，互联网资讯不错过(...</a>

猜您关注

<a href="#">在ABP中通过EF直接执行原生Sq...</a>
<a href="#">T-SQL常用数据库操作语句</a>
<a href="#">go利用x/net/html包实现的蜘蛛</a>
<a href="#">P H P 根据二维数组的某键值...</a>
<a href="#">Xamarin.Forms项目无法添加服务引用</a>
<a href="#">springboot简单介绍</a>
<a href="#">[置顶] Prio...</a>
<a href="#">[置顶] Glob...</a>
<a href="#">简单实用的遮幕原点扩散交互特效</a>
<a href="#">《统计学习方法》学习笔记---...</a>
<a href="#">《统计学习方法》学习笔记--k...</a>
<a href="#">团体程序设计天梯赛“练习集”...</a>

1， 使用颜色打印调试信息：

```
1. printf("33[46;31m[%s:%d]33[0m "#fmt" errno=%d, %mrn", __func__, __LINE__, ##args, errno,
    errno);
```

上面printf时在Linux命令行下打印出带颜色的字体，方便一眼区分不同种类的调试信息，只需要加上一些颜色代码，例如：这里的46代表底色, 31代表字体的颜色。

使用ascii code 是对颜色调用的始末格式如下：

```
1. 033[; m ..... 033[0m
```

后面哪个”33[0m” 是对前面哪个颜色载入的结束，恢复到终端原来的背景色和字体色，可以把后面哪个修改成如下试试：

```
1. #define DEBUG_ERR(fmt, args...) printf("33[46;31m[%s:%d]33[40;37m "#fmt" errno=%d,
    %mrn", __func__, __LINE__, ##args, errno, errno);
```

下面列出 ascii code 的颜色值：

字背景颜色范围:40----49	字颜色:30-----39
40:黑	30:黑
41:深红	31:红
42:绿	32:绿
43:黄色	33:黄
44:蓝色	34:蓝色
45:紫色	35:紫色
46:深绿	36:深绿
47:白色	37:白色

2， 打印调试信息的跟踪位置：

```
1. printf("[%s:%s] line=%drn", __FILE__, __func__, __LINE__);  
2. printf("33[33m[%s:%d]33[0m "#fmt"rn", __func__, __LINE__, ##args);
```

如上代码：

- 1) `__FILE__` 打印出调试信息所在的文件名；
  - 2) `__func__` 将会打印出调试信息所在的函数名；
  - 3) `__LINE__` 将会打印出调试信息所在文件的行号；
- 3， 使用不定参数向打印信息里面加入自己想看到的调试信息：

```
1. #define DEBUG_INFO(fmt, args...) printf("\33[33m[%s:%d]\33[0m \"%fmt\"rn", __func__, __LINE__,  
    ##args);
```

调用方式如下：

```
1. int i = 110;  
2. char * s = "hello world!" ;  
3. DEBUG_INFO("Garfield test DEBUG_INFO() d: %d ; s: %s", i, s);
```

至于不定数量参数宏与不定参数函数的使用就没神马好说的啦，自己去google吧！

下面引用一位大侠的blog，列出一些常用的debug语句：

出自：[http://blog.mcuol.com/User/luoming/Article/16499\\_1.htm](http://blog.mcuol.com/User/luoming/Article/16499_1.htm)

```
1. #ifdef DEBUG  
2. #define F_OUT printf("%s:", __FUNCTION__);fflush(stdout);  
3. #define L_OUT printf("%s:%d:", __FILE__, __LINE__);fflush(stdout);  
4. #define A_OUT printf("%s:%d:%s:", __FILE__, __LINE__, __FUNCTION__);fflush(stdout);  
5. #define D_OUT printf("DEBUG:");fflush(stdout);  
6.  
7. #define F_PRINTF(fmt, arg...) F_OUT printf(fmt, ##arg)  
8. #define L_PRINTF(fmt, arg...) L_OUT printf(fmt, ##arg)  
9. #define A_PRINTF(fmt, arg...) A_OUT printf(fmt, ##arg)  
10. #define PRINTF(fmt, arg...) D_OUT printf(fmt, ##arg)  
11. #define DBUG(a){a;}  
12. #else  
13. #define F_OUT  
14. #define L_OUT  
15. #define A_OUT  
16. #define D_OUT  
17.  
18. #define F_PRINTF(fmt, arg...)  
19. #define L_PRINTF(fmt, arg...)  
20. #define A_PRINTF(fmt, arg...)  
21. #define PRINTF(fmt, arg...)
```

```
22. #define DBUG(a)
23. #endif
24.
25. #define F_PERROR(fmt) F_OUT perror(fmt)
26. #define L_PERROR(fmt) L_OUT perror(fmt)
27. #define A_PERROR(fmt) A_OUT perror(fmt)
28. #define PERROR(fmt) D_OUT perror(fmt)
```

## C语言中几种输出调试信息的方法

<http://blog.csdn.net/thinkerabc/article/details/615378>

在调试程序时，输出调试信息是一种普遍、有效的方法。输出调试信息一般有以下五种方法：

方法一：直接使用屏幕打印函数printf。

该方法直接在需要输出调试信息的位置使用函数printf输出相应的调试信息，以及某些关键变量的值。我们通过以下求阶层的函数fact来看看该方法的调试程序过程。

```
#include <stdio.h>
int fact(int n)
{
    int i,f=1;
    for( i=1; i<=n; i++)
    {
        f += i;
    }
    return f;
}
int main()
{
    printf( "4!=%d/n", fact(4) );
    return 0;
}
```

程序1: 有bug的求阶层函数

程序1编译运行的结果如下：

4!=11

结果错误。为了找到结果错误的原因，我们在语句" f += i;"之后插入函数printf输出调试信息，如程序2。

```
#include <stdio.h>
int fact(int n)
{
    int i,f=1;
    for( i=1; i<=n; i++)
    {
        f += i;
        printf("i=%d ; f=%d/n", i, f);
    }
}
```

```

    }
    return f;
}

int main()
{
    printf( "4!=%d/n", fact(4) );
    return 0;
}

```

程序2: 加入函数printf输出调试信息的求阶层函数

再编译运行该程序，屏幕输出如下：

```
i=1 ; f=2 i=2 ; f=4 i=3 ; f=7 i=4 ; f=11 4!=11
```

原来语句"f += i"错了，应该为"f \*=i"。修改过来（见程序3），再编译运行，结果如下：

```

i=1 ; f=1 i=2 ; f=2 i=3 ; f=6 i=4 ; f=24 4!=24 #include <stdio.h>

int fact(int n)
{
    int i,f=1;
    for( i=1; i<=n; i++)
    {
        f *= i;
        printf("i=%d ; f=%d/n", i, f);
    }
    return f;
}

int main()
{
    printf( "4!=%d/n", fact(4) );
    return 0;
}

```

程序3：修改正确的求阶层函数

调试完成，bug找到，并修改正确。然后将加入的调试的函数printf 删除或注释掉。

该方法的缺点是（1）在正式发布的程序中需要去除或注释掉这些调试语句；（2）若程序又出现bug，则又需要重新插入函数printf输出调试信息，造成工作的重复。

方法二：自定义调试函数debug。

为了避免方法一的缺点，可以利用条件编译技术，如程序4自定义调试函数debug。当程序正式发布的编译时取消宏定义\_\_DEBUG\_\_，在正式发布的程序中就不会输出调试信息。若又出现bug，只要重新在编译程序时定义宏\_\_DEBUG\_\_即可恢复原来的调试信息输出。可以在编写程序时就有目的事先插入些调试语句，这将有益于调试程序。另外，可以根据需要编写函数debug，将调试信息输出到除屏幕以外的其它地方，如文件或syslog服务器等。

```

#include <stdio.h>

#ifdef __DEBUG__
#include <stdarg.h>

void debug(const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);

```

```

    vprintf(fmt, ap);
    va_end(ap);
}
#else
void debug(const char *fmt, ...)
{
}
#endif

int fact(int n)
{
    int i, f = 1;
    for( i=1; i<=n; i++)
    {
        f *= i;
        debug("i=%d ; f=%d/n", i, f);
    }
    return f;
}

int main()
{
    printf( "4!=%d/n", fact(4) );
    return 0;
}

```

#### 程序4: 自定义调试函数debug

该方法的缺点是（1）调试信息要么全部输出，要么全不输出；（2）要重新输出调试信息时需要重新编译程序。

方法三：含调试等级的自定义调试函数debug。

可以继续改进方法，避免方法二中的缺点。我们可以根据调试信息的细节程度，将调试信息分成不同的等级。调试信息的等级必须大于0，若调试信息细节程度越高，则等级越高。在输出调试信息时，若调试等级高于调试信息等级才输出调试信息，否则忽略该调试信息，如程序5。当调试等级为0时，则不输出任何调试信息。

```

#include <stdio.h>
#include <stdlib.h> /* atoi() */

#include <stdarg.h>

int debug_level;
void debug(int level, const char *fmt, ...)
{
    if( level <= debug_level )
    {
        va_list ap;
        va_start(ap, fmt);
        vprintf(fmt, ap);
        va_end(ap);
    }
}

```

```
int fact(int n)
{
    int i, f = 1;
    for( i=1; i<=n; i++)
    {
        f *= i;
        debug(250, "i=%d ; f=%d/n", i, f);
    }
    return f;
}

int main(int argc, char *argv[])
{
    if ( argc < 2 )
    {
        debug_level = 0;
    }
    else
    {
        debug_level = atoi(argv[1]);
    }
    printf( "4!=%d/n", fact(4) );
    return 0;
}
```

程序5: 含调试等级的自定义调试函数debug

用命令"gcc -Wall -o fact fact.c"编译程序5, 得到可执行文件 fact。若需要输出调试信息, 只需要指定调试等级不低于250即可, 如运行命令"./fact 250", 否则将不会输出调试信息。

这样, 在正式发布版中包含调试信息也无伤大雅了, 因为只需将调试等级配置为0, 将不会出现任何调试信息。

该方法的缺点是效率不太高, 因为不管调试信息是否需要输出, 都会进行一次函数调用。若不需要输出调试信息, 这次函数调用就多余了。

方法四: 调试等级的判断放在自定义调试函数debug之外。

为了减少不必要的函数调用, 可以用宏定义将调试等级的判断放在函数debug之外, 如程序6。

```
#include <stdio.h>
#include <stdlib.h> /* atoi() */

#include <stdarg.h>

int debug_level;

#define debug(level, fmt, arg...) /
    if( level <= debug_level ) __debug(fmt, ##arg)

void __debug(const char *fmt, ...)
{
    va_list ap;
```

```

    va_start(ap, fmt);
    vprintf(fmt, ap);
    va_end(ap);
}

int fact(int n)
{
    int i, f = 1;
    for( i=1; i<=n; i++)
    {
        f *= i;
        debug(250, "i=%d ; f=%d/n", i, f);
    }
    return f;
}

int main(int argc, char *argv[])
{
    if ( argc < 2 )
    {
        debug_level = 0;
    }
    else
    {
        debug_level = atoi(argv[1]);
    }
    printf( "4!=%d/n", fact(4) );
    return 0;
}

```

程序6: 调试等级的判断放在自定义调试函数debug之外

这种方法对于不需要输出的高等级的调试信息操作来说，仅仅多了个两个整数之间的大小判断。在正式的程序运行时，效率是有所提高的。

但这种调试信息输出的方法依然不够完美。对于一个大项目，一般分为若干个模块，bug将会定位到某个或某几个模块。若整个项目的调试信息都输出，信息量将会非常大，也容易干扰调试人员的思维。这时，我们需要的是只输出我们关心的那些模块的调试信息，但该方法并不能达到我们的要求。它只能根据调试等级输出信息，对于同一调试等级的信息要么全输出，要么全不输出。

方法五：根据不同的功能模块分别定义不同的调试等级。

在squid[1]中，定义了以下的功能模块调试等级变量和调试函数：

```

int debugLevels[MAX_DEBUG_SECTIONS];
#define debug(SECTION, LEVEL) /
    ((_db_level = (LEVEL)) > debugLevels[SECTION]) ? (void) 0 : _db_print
然后在程序中如下使用它：

```

```

debug(17, 3) ("fwdStateFree: %p/n", fwdState);

```

上述调试函数很灵活，可以在不同的模块中定义有不同的调试等级，当需要调试某功能时，



只需将该模块的调试等级定义为相应的等级，就可输出需要的调试信息。

根据方法五的思想，本人编写了my\_debug.h（见程序7）和my\_debug.c文件（见程序8）。该文件可以应用于C语言程序中，支持根据不同的功能模块分别定义不同的调试等级。

```
#ifndef MY_DEBUG_H
#define MY_DEBUG_H

#include <stdio.h>

// 模块功能号
enum {
    MY_SECTION_FACT = 0,
    MY_SECTION_nnn1,
    MY_SECTION_nnn2,
    MY_SECTION_nnnn,
    MY_SECTION_END,
};

// 非my_debug.c文件的外部变量声明
#ifndef MY_DEBUG_C
extern int __my_allow_debug_levels[MY_SECTION_END];
#endif

// (内部使用) 判断"SECTION"模块功能号是否允许"DEBUG_LEVEL"等级的调试信息输出
#define __my_unallow_debug(SECTION, DEBUG_LEVEL) /
    ( DEBUG_LEVEL > __my_allow_debug_levels[SECTION] )

// (内部使用) 调试信息输出函数
#define __my_debug(FORMAT, ARG...) /
    printf("%s:%d %s: " FORMAT, __FILE__, __LINE__, __FUNCTION__, ##ARG)

// 初始化"SECTION"模块功能号的调试等级
#define my_init_debug_levels(SECTION, ALLOW_DEBUG_LEVEL) /
    ( __my_allow_debug_levels[SECTION] = ALLOW_DEBUG_LEVEL )

// 调试信息输出函数，该信息为"SECTION"模块功能号"DEBUG_LEVEL"等级的调试信息
#define my_debug(SECTION, DEBUG_LEVEL) /
    ( __my_unallow_debug(SECTION, DEBUG_LEVEL) ) ? (void) 0 : __my_debug

#endif //MY_DEBUG_H

程序7: my_debug.h

#define MY_DEBUG_C
#include "my_debug.h"

int __my_allow_debug_levels[MY_SECTION_END];
```

## 程序8: my\_debug.c

要使用上述文件，先得根据功能模块的数目扩展my\_debug.h中的“模块功能号”枚举类型，然后在程序相应位置中调用宏定义my\_init\_debug\_levels 初始化相应模块的调试等级，在所有需要输出调试信息的位置如下编写即可。

```
my_debug(MY_SECTION_FACT, 250)("i=%d ; f=%d/n", i, f);
```

下面我们来看看如何在fact.c中使用它们（见程序9）。

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include "my_debug.h"
```

```
int fact(int n)
```

```
{
```

```
    int i, f = 1;
```

```
    for( i=1; i<=n; i++)
```

```
    {
```

```
        f *= i;
```

```
        my_debug(MY_SECTION_FACT, 250)("i=%d ; f=%d/n", i, f);
```

```
    }
```

```
    return f;
```

```
}
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    if ( argc < 2 )
```

```
    {
```

```
        my_init_debug_levels(MY_SECTION_FACT, 0);
```

```
    }
```

```
    else
```

```
    {
```

```
        my_init_debug_levels(MY_SECTION_FACT, atoi(argv[1]));
```

```
    }
```

```
    printf( "4!=%d/n", fact(4) );
```

```
    return 0;
```

```
}
```


程序9: fact.c

相关推荐

猜你喜欢



Python 安装  
matplotlib,six,dateutil,  
The MySQL server is



[Error Code: 1290.  
The MySQL server is



安装ionic 以及  
cordova 环境配置详细

IOS开发----CMDeviceMotion...

技术人员教你淘宝屏蔽找同款...

App Transport Security has...

cannot run program...

xcodes7、iOS9 设置启动图片...

Android网络请求框架...

IOS项目集成ShareSDK实现...

解决Android studio启动时启...

百度推荐

感谢关注 Ithao123精品文库频道，ithao123.cn是专门为互联网人打造的学习交流平台，全面满足互联网人工作与学习需求，更多互联网资讯尽在 IThao123!

关键词：

linux

c

调试方法

文章点评：

提交

精选专题



Laravel入门教程

Laravel是一套简洁、优雅的  
PHP Web开发框架(PHP Web



hadoop学习及应用

Hadoop是一个由Apache基金  
会所开发的分布式系统基础架



互联网产品设计

产品设计是互联网产品经理的核  
心能力，一个好的产品经理一定



产品经理职责

随着国内互联网的发展，产品经  
理岗位需求大幅增加，在国内，

