



个人资料

sunboy\_2050

+ 加关注

发私信

专家

CSDN

访问：2465000次

积分：22397分

排名：第57名

原创：346篇 转载：210篇

译文：10篇 评论：1283条

CSDN 博客之星评选

2012-csdn

博客之星

评选

震撼来袭 周末好礼 等你来参与

我是阳光岛

投我一票

工作经历

中科院、百度、六所、创新工场

博客专栏

设计模式  
文章：3篇  
阅读：35811

Android开发的

💡 2013年4月微软MVP申请开始啦！

CSDN论坛2012年度职场优秀帖评选活动正式开始！

CSDN 博客频道年终送好礼！

2012CSDN博客之星评选正式上线

2000元大奖征异构开发博文

Q14年互联网产品进化史

## 原 \_\_attribute\_\_ 机制介绍

分类：C/C++/C#

2011-06-24 23:10

4404人阅读

评论(5)

收藏

分享

### 1. \_\_attribute\_\_

GNU C的一大特色（却不被初学者所知）就是\_\_attribute\_\_ 机制。

\_\_attribute\_\_ 可以设置函数属性(Function Attribute)、变量属性(Variable Attribute)和类型属性(Type Attribute)

\_\_attribute\_\_ 前后都有两个下划线，并且后面会紧跟一对原括弧，括弧里面是相应的\_\_attribute\_\_ 参数

\_\_attribute\_\_ 语法格式为：

\_\_attribute\_\_ ( ( attribute-list ) )

函数属性（Function Attribute），函数属性可以帮助开发者把一些特性添加到函数声明中，从而可以使编译器在错误查方面的功能更强大。

\_\_attribute\_\_ 机制也很容易同非GNU应用程序做到兼容。

GNU CC需要使用 -Wall，这是控制警告信息的一个很好的方式。下面介绍几个常见的属性参数。

### 2. format

该属性可以使编译器检查函数声明和函数实际调用参数之间的格式化字符串是否匹配。它可以给被声明的函数加上类似printf或者scanf的特征，该功能十分有用，尤其是处理一些很难发现的bug。

format的语法格式为：

format ( archetype, string-index, first-to-check )

format属性告诉编译器，按照printf，scanf，strftime或strfmon的参数表格式规则对该函数的参数进行检查。archetype：指定是哪种风格；

string-index：指定传入函数的第几个参数是格式化字符串；

first-to-check：指定从函数的第几个参数开始按上述规则进行检查。

具体使用格式如下：

\_\_attribute\_\_( ( format( printf, m, n ) ) )

\_\_attribute\_\_( ( format( scanf, m, n ) ) )

其中参数m与n的含义为：

m：第几个参数为格式化字符串（format string）；

n：参数集合中的第一个，即参数“...”里的第一个参数在函数参数总数排在第几

注意，有时函数参数里还有“隐身”的呢，后面会提到；

在使用上，\_\_attribute\_\_((format(printf,m,n)))是常用的，而另一种却很少见到。

下面举例说明，其中myprint为自己定义的一个带有可变参数的函数，其功能类似于printf：

//m=1；n=2



点点滴滴

文章：32篇


阅读：750528

博客公告

本博客内容，由本人精心整理

欢迎交流，欢迎转载，大家转载  
请注明出处，禁止用于商业目的

文章搜索



文章分类

C/C++/C# (112)

Algorithm (31)

Linux (126)

QT (13)

Script (55)

NetWork (21)

SQL Index (47)

SoftWare (23)

Java/JSP (31)

Learn (17)

IT Trend (32)

Android (90)

iOS (4)

文章存档

2012年12月 (10)

2012年11月 (15)

2012年10月 (12)

2012年09月 (15)

2012年08月 (16)

 展开

阅读排行

Android APK反编译详解

(256038)

SVN常用命令

(106538)

Ubuntu搭建Eclipse+JDK

(95616)

Android 获取屏幕尺寸与

(92541)

Linux 抓取网页实例 (sh

(87408)

各种基本算法实现小结 (

(83512)

Android 创建与解析XML

(82868)

Windows搭建Eclipse+JC

(82185)

去360还是留百度?

(49623)

5大科技巨头的战争

(32004)

```
extern void myprint( const char *format, ... ) __attribute__( ( format( printf, 1, 2 ) ) );

//m=2; n=3

extern void myprint( int l, const char *format, ... ) __attribute__( ( format( printf, 2, 3 ) ) );

需要特别注意的是，如果myprint是一个函数的成员函数，那么m和n的值可有点“悬乎”了，例如：

//m=3; n=4

extern void myprint( int l, const char *format, ... ) __attribute__( ( format( printf, 3, 4 ) ) );

其原因是，类成员函数的第一个参数实际上一个“隐身”的“this”指针。（有点C++基础的都知道点this指针，不知道位
这里还知道吗？）

这里给出测试用例：attribute.c，代码如下：
```

```
extern void myprint(const char *format,...) __attribute__((format(printf,1,2)));

void test()

{

    myprint("i=%d/n",6);

    myprint("i=%s/n",6);

    myprint("i=%s/n","abc");

    myprint("%s,%d,%d/n",1,2);

}
```

运行\$gcc -Wall -c attribute.c attribute后，输出结果为：

attribute.c: In function `test':

attribute.c:7: warning: format argument is not a pointer (arg 2)

attribute.c:9: warning: format argument is not a pointer (arg 2)

attribute.c:9: warning: too few arguments for format

如果在attribute.c中的函数声明去掉\_\_attribute\_\_((format(printf,1,2)))，再重新编译，

既运行\$gcc -Wall -c attribute.c attribute后，则并不会输出任何警告信息。

注意，默认情况下，编译器是能识别类似printf的“标准”库函数。

3. noreturn

该属性通知编译器函数从不返回值。

当遇到函数需要返回值却还没运行到返回值处就已退出来的情况，该属性可以避免出现错误信息。C库函数中的abort（）和exit（）的声明格式就采用了这种格式：

```
extern void exit(int) __attribute__( ( noreturn ) );

extern void abort(void) __attribute__( ( noreturn ) );
```

为了方便理解，大家可以参考如下的例子：

```
//name: noreturn.c    ；测试__attribute__((noreturn))

extern void myexit();

int test( int n )

{

    if ( n > 0 )

    {

        myexit();

        /* 程序不可能到达这里 */

    }

    else

    {

        return 0;

    }

}
```

```

    }
}

```

编译\$gcc -Wall -c noreturn.c 显示的输出信息为：

noreturn.c: In function `test':

noreturn.c:12: warning: control reaches end of non-void function

警告信息也很好理解，因为你定义了一个有返回值的函数test却有可能没有返回值，程序当然不知道怎么办了！加上\_\_attribute\_\_((noreturn))则可以很好的处理类似这种问题。把extern void myexit();修改为：

```
extern void myexit() __attribute__((noreturn));
```

之后，编译不会再出现警告信息。

#### 4. const

该属性只能用于带有数值类型参数的函数上，当重复调用带有数值参数的函数时，由于返回值是相同的。所以此时编译器可以进行优化处理，除第一次需要运算外，其它只需要返回第一次的结果。

该属性主要适用于没有静态状态（static state）和副作用的一些函数，并且返回值仅仅依赖输入的参数。为了说明问题，下面举个非常“糟糕”的例子，该例子将重复调用一个带有相同参数值的函数，具体如下：

```
extern int square( int n ) __attribute__( (const) );
```

```
for (i = 0; i < 100; i++)
```

```

{
    total += square (5) + i;
}

```

添加\_\_attribute\_\_((const))声明，编译器只调用了函数一次，以后只是直接得到了相同的一个返回值。

事实上，const参数不能用在带有指针类型参数的函数中，因为该属性不但影响函数的参数值，同样也影响到了参数指向的数据，它可能会对代码本身产生严重甚至是不可恢复的严重后果。并且，带有该属性的函数不能有任何副作用或是静态的状态，类似getchar（）或time（）的函数是不适合使用该属性。

#### 5. finstrument-functions

该参数可以使程序在编译时，在函数的入口和出口处生成instrumentation调用。恰好在函数入口之后并恰好在函数出口之前，将使用当前函数的地址和调用地址来调用下面的profiling函数。（在一些平台上，\_\_builtin\_return\_address不在超过当前函数范围之外正常工作，所以调用地址信息可能对profiling函数是无效的）

```
void __cyg_profile_func_enter( void *this_fn, void *call_site );
```

```
void __cyg_profile_func_exit( void *this_fn, void *call_site );
```

其中，第一个参数this\_fn是当前函数的起始地址，可在符号表中找到；第二个参数call\_site是调用处地址。

#### 6. instrumentation

也可用于在其它函数中展开的内联函数。从概念上来说，profiling调用将指出在哪里进入和退出内联函数。这就意味着这种函数必须具有可寻址形式。如果函数包含内联，而所有使用到该函数的程序都要把该内联展开，这会额外地增代码长度。如果要在C 代码中使用extern inline声明，必须提供这种函数的可寻址形式。

可对函数指定no\_instrument\_function属性，在这种情况下不会进行instrumentation操作。例如，可以在以下情况下用no\_instrument\_function属性：上面列出的profiling函数、高优先级的中断例程以及任何不能保证profiling正常调用函数。

```
no_instrument_function
```

如果使用了-finstrument-functions，将在绝大多数用户编译的函数的入口和出口点调用profiling函数。使用该属性，不进行instrument操作。

#### 7. constructor/destructor

若函数被设定为constructor属性，则该函数会在main（）函数执行之前被自动的执行。类似的，若函数被设定

为**destructor**属性，则该函数会在main（）函数执行之后或者exit（）被调用后被自动的执行。拥有此类属性的函数经常隐式的用在程序的初始化数据方面，这两个属性还没有在面向对象C中实现。

### 8. 同时使用多个属性

可以在同一个函数声明里使用多个\_\_attribute\_\_，并且实际应用中这种情况是十分常见的。使用方式上，你可以选择一个单独的\_\_attribute\_\_，或者把它们写在一起，可以参考下面的例子：

```
extern void die(const char *format, ...) __attribute__((noreturn)) __attribute__((format(printf, 1, 2)));
```

或者写成

```
extern void die(const char *format, ...) __attribute__((noreturn, format(printf, 1, 2)));
```

如果带有该属性的自定义函数追加到库的头文件里，那么所以调用该函数的程序都要做相应的检查。

### 9. 和非GNU编译器的兼容性

\_\_attribute\_\_设计的非常巧妙，很容易作到和其它编译器保持兼容。也就是说，如果工作在其它的非GNU编译器上，以很容易的忽略该属性。即使\_\_attribute\_\_使用了多个参数，也可以很容易的使用一对圆括弧进行处理，例如：

```
/* 如果使用的是非GNU C, 那么就忽略__attribute__ */
```

```
#ifndef __GNUC__
```

```
    #define __attribute__(x) /* NOTHING */
```

```
#endif
```

需要说明的是，\_\_attribute\_\_适用于函数的声明而不是函数的定义。所以，当需要使用该属性的函数时，必须在同一文件里进行声明，例如：

```
/* 函数声明 */
```

```
void die( const char *format, ... ) __attribute__((noreturn)) __attribute__((format(printf, 1, 2)));
```

```
void die( const char *format, ... )
```

```
{ /* 函数定义 */ }
```

更多属性参考：<http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Function-Attributes.html>

### 10. 变量属性 (Variable Attributes)

关键字\_\_attribute\_\_也可以对变量 (variable) 或结构体成员 (structure field) 进行属性设置。

在使用\_\_attribute\_\_参数时，你也可以在参数的前后都加上“\_\_”（两个下划线），例如，使用\_\_aligned\_\_而不是aligned，这样，你就可以在相应的头文件里使用它而不用关心头文件里是否有重名的宏定义。

### 11. 类型属性 (Type Attribute)

关键字\_\_attribute\_\_也可以对结构体 (struct) 或共用体 (union) 进行属性设置。

大致有六个参数值可以被设

定：aligned, packed, transparent\_union, unused, deprecated, may\_alias

### 12. aligned (alignment)

该属性设定一个指定大小的对齐格式（以字节为单位），例如：

```
struct S { short f[3]; } __attribute__((aligned(8)));
```

```
typedef int more_aligned_int __attribute__((aligned(8)));
```

这里，如果sizeof (short) 的大小为2 (byte)，那么，S的大小就为6。取一个2的次方值，使得该值大于等于6，则值为8，所以编译器将设置S类型的对齐方式为8字节。该声明将强制编译器确保（尽它所能）变量类型为struct S或者more-aligned-int的变量在分配空间时采用8字节对齐方式。

如上所述，你可以手动指定对齐的格式，同样，你也可以使用默认的对齐方式。例如：

```
struct S { short f[3]; } __attribute__((aligned));
```

上面，aligned后面不紧跟一个指定的数字值，编译器将依据你的目标机器情况使用最大最有益的对齐方式。

```
int x __attribute__ ( ( aligned (16) ) ) = 0;
```

编译器将以16字节（注意是字节byte不是位bit）对齐的方式分配一个变量。也可以对结构体成员变量设置该属性，例如，创建一个双字对齐的int对，可以这么写：

```
Struct foo { int x[2] __attribute__ ( ( aligned (8) ) );};
```

选择针对目标机器最大的对齐方式，可以提高拷贝操作的效率。

aligned属性使被设置的对象占用更多的空间，相反的，使用packed可以减小对象占用的空间。

需要注意的是，attribute属性的效力与你的连接器也有关，如果你的连接器最大只支持16字节对齐，那么你此时定义32字节对齐也是无济于事的。

### 13. packed

使用该属性可以使得变量或者结构体成员使用最小的对齐方式，即对变量是一字节对齐，对域（field）是位对齐。使用该属性对struct或者union类型进行定义，设定其类型的每一个变量的内存约束。当用在enum类型定义时，暗示了应使用最小完整的类型（it indicates that the smallest integral type should be used）。

下面的例子中，x成员变量使用了该属性，则其值将紧放置在a的后面：

```
struct test
{
    char a;
    int x[2] __attribute__ ((packed));
};
```

下面的例子中，my-packed-struct类型的变量数组中的值将会紧紧的靠在一起，但内部的成员变量s不会被“pack”，如希望内部的成员变量也被packed，my-unpacked-struct也需要使用packed进行相应的约束。

```
struct my_packed_struct
{
    char c;
    int i;
    struct my_unpacked_struct s;
}__attribute__ ( ( __packed__ ) );
```

其它可选的属性值还可以是：cleanup，common，nocommon，deprecated，mode，section，shared，tls\_model，transparent\_union，unused，vector\_size，weak，dllimport，dlexport等。

更多详细参考：<http://gcc.gnu.org/onlinedocs/gcc-4.0.0/gcc/Variable-Attributes.html#Variable-Attributes>

### 14. 变量属性与类型属性举例

下面的例子中使用\_\_attribute\_\_属性定义了一些结构体及其变量，并给出了输出结果和对结果的分析。

程序代码为：

```
struct p
{
    int a;
    char b;
    char c;
}__attribute__( ( aligned(4) ) ) pp;

struct q
{
    int a;
    char b;
    struct n qn;
    char c;
```

```
__attribute__((aligned(8))) qq;

int main()
{
    printf("sizeof(int)=%d, sizeof(short)=%d, sizeof(char)=%d/n", sizeof(int), sizeof(short), sizeof(char));
    printf("pp=%d, qq=%d /n", sizeof(pp), sizeof(qq));
    return 0;
}
```

输出结果:

sizeof(int)=4, sizeof(short)=2, sizeof(char)=1

pp=8, qq=24

分析:

sizeof(pp):

sizeof(a)+ sizeof(b)+ sizeof(c)=4+1+1=6<2^3=8= sizeof(pp)

sizeof(qq):

sizeof(a)+ sizeof(b)=4+1=5

sizeof(qn)=8;

即qn是采用8字节对齐的,所以要在a, b后面添3个空余字节,然后才能存储qn,

4+1+ (3) +8+1=17

因为qq采用的对齐是8字节对齐,所以qq的大小必定是8的整数倍,即qq的大小是一个比17大又是8的倍数的一个最小值,由此得到

17<2^4+8=24= sizeof(qq)

更详细的介绍见: <http://gcc.gnu.org/>

下面是一些便捷的连接:

[GCC 4.0 Function Attributes](#)

[GCC 4.0 Variable Attributes](#)

[GCC 4.0 Type Attributes](#)

## 15. Ref

简单\_\_attribute\_\_介绍: <http://www.unixwiz.net/techtips/gnu-c-attributes.html>

详细\_\_attribute\_\_介绍: <http://gcc.gnu.org/>

分享到:  

上一篇: [微软面试、经典算法、编程艺术、红黑树4大系列总结](#)

下一篇: [Qt坐标系](#)

顶

4

踩

0

查看评论

5楼 [fork\\_thread](#) 2012-12-12 10:44 发表 



写的很好!  
五月的风。

4楼 [endlessbest](#) 2012-12-08 09:41 发表 

非常实用,感谢



3楼 [therunninglion](#) 2012-12-05 16:30 发表



写的很好，收藏了。THX

2楼 [creating2008](#) 2012-11-12 22:37 发表



总结的很好

1楼 [jiangwlee](#) 2012-11-09 13:19 发表



很全，很详细！

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#)

京 ICP 证 070598 号

北京创新乐知信息技术有限公司 版权所有

世纪乐知(北京)网络技术有限公司 提供技术支持

江苏乐知网络技术有限公司 提供商务支持

联系邮箱: [webmaster\(at\)csdn.net](mailto:webmaster(at)csdn.net)

Copyright © 1999-2012, CSDN.NET, All Rights Reserved