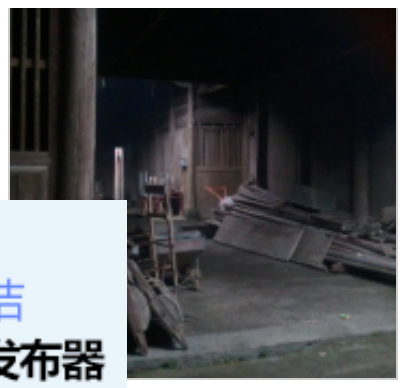


wuqiseu

ng　微博

友　发纸条

言　加关注

15

博客积分：1193
博客访问：71,217
关注人气：24
获赠金笔：0
赠出金笔：0
荣誉徽章：

相关博文

女星穿西装大秀性感好身材
八卦月月

福利大放送，玻尿酸免费注射
花漾整形问答

评论：伊拉克虽败犹荣　尤尼斯
郑道锦

金融股砸盘是主力洗盘伎俩
玉名

【解析】《何以笙箫默》未沉没
影视独舌

我离婚改嫁情人，他却外遇不断
妮夏OL

展锋：高开低走，大幅下挫要小
展锋

午后可以加仓热点题材股
拾金客v

陈毅对张灵甫自杀说法震怒的真
中天飞鸿

风格转换春节前最
神仙看盘

C/C++ 编译器的命名粉碎规则(name mangling)和C/C++混合(ZZ)

(2012-03-06 14:49:49)

转 载 ▼

标签： 杂谈　分类： 编程语言C/C /PERL/Tcl

http://hi.baidu.com

在面向对象编程语言出现之前，如果你想要打印不同类型的数据,需要写多个方法 ,象是 PrintInteger(int i)，PrintString(string s) 和 PrintFloat(float f) 。也就是说，你必须通过命名来区别行为和数据类型，因为 OOP语言出现前任一语言象是C，不允许你用相同的名字写方法， 即使他们的参数类型不同。

C++的来到实现了方法重载。因此，你可以写多个方法 ， 象是 PrintInteger(int i)、PrintString(string s) 和 PrintFloat(float f)，编译器自会准确调用特定的Print方法。方法重载被一种称为名称重整（name mangling）的技术所支持，在这种技术中，编译器通过把原方法名称与其参数相结合产生一个独特的内部名字来取代原方法名称。 如此，当你调用Print(1)的时候，编译器可能在内部用源于参数类型的前缀重命名Print方法，这样一来Print(1)可能就变成 i_Print (1) 。

下面是更详细的例子：

C++编译器实际上将下面这些重载函数：

```
1 void print(int i);
2 void print(char c);
3 void print(float f);
4 void print(char* s);
```

编译为：

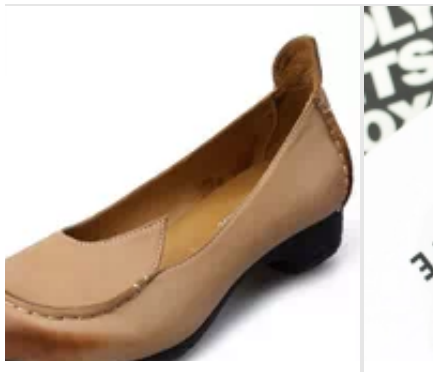
```
1 _print_int
2 _print_char
3 _print_float
4 _pirnt_string
```

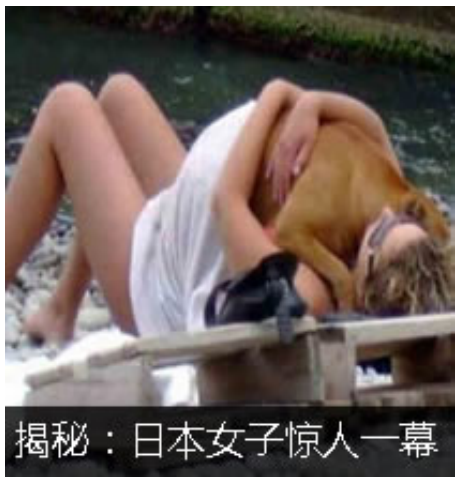
这样的函数名，来唯一标识每个函数。注：不同的编译器实现可能不一样，但是都是利用这种机制。所以当连接是调用print(3)时，它会去查找_print_int(3)这样的函数。下面说个题外话，正是因为这点，重载被认为不是多态，多态是运行时动态绑定（“一种接口多种实现”），如果硬要认为重载是多态，它顶多是编译时“多态”。

C++中的变量，编译也类似，如全局变量可能编译g_xx，类变量编译为c_xx等。连接是也是按照这种机制去查找相应的变量

方法重载仅是多态性的一种情形。 名称重整是一种支持方法重载的机制。更普遍的情况下，多态性是与继承相联系。 什么是继承呢？继承就是一个新类（称为子类）从被继承类（称为父类或超类）取得自身的部分定义同时增加一些自己的新的信息。 如果你在相同的类中重载方法，数据类型必须是不同的。如果你在继承关系下重载方法，子类与父类的方法可能完全相同，而且名称重整器生成同样的重整名称。

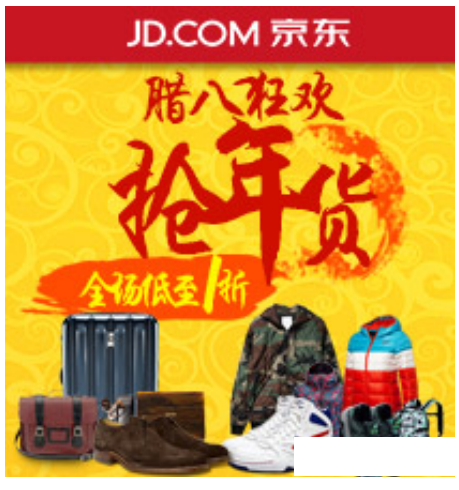
举例来说，假设一个超类定义一个Print(int i)方法而一个从它继承的子类也定义了一个Print(int i)方法。当你有一个子类的实例时，运用多态性调用Child.Print(int)；而当你产生一个父类的实例时运用多态性调用Parent.Print(int)。这就是继承多态性：相同的名字和签字但是类却不同。





盘点被成龙熊抱过的女明星





继承多态性是通过使用一种与名称重整相关的另外一种机制实现的。编译器把方法放置在一个被称为虚拟方法表（其实是一个方法数组）的地方。每一个方法在VMT中都有一个索引，如此当Print(int)被调用的时候，编译器将被路由到VMT处找寻Print方法和类的内在索引。这样一来，编译器就可以调用正确的方法实现。由编译器负责管理所有的VMT索引和类偏移量。

简言之，多态性使你能够用非常相似的名字定义许多方法,这里的名字往往都是直观易记的。OOP编译器自会根据调用者类理解到底该调用哪个方法。

http://it.china-b.com/cxsj/cc/20090612/91952_1.html

Name-mangling是指为了在目标文件符号表中和连接过程中使用的名字通常和编译目标文件的源程序中的名字不一样，编译器将目标源文件中的名字进行调整。Name-mangling不是一个非常新的技术，例如在C语言中也有，我们在汇编C语言时经常看到的以下划线“_”开头的函数名，其实就是C编译器将函数名进行了Name-mangling。但是在C++中Name-mangling要复杂的多。因为C++中支持overload和override，这就导致了C++编译器必须要有完成的Name-mangling把函数名或者变量名进行调整。 一种Name-mangling的方法（选自Linkerandloader）：在C++类外的数据变量的名字完全不进行调整。一个叫做foo的数组的名字调整后还是foo。没有与类相关的函数名字的调整是根据参数类型用__F后缀和一串代表参数类型的字母进行的。 参数类型一般会被进行简写（如void→v，int→i，float→f，char→c，double→d，varages→e，long→l，unsigned→U，const→C，volatile→V，pointer→P，reference→R，function→f，pointertonthmembers→MnS等等）。举例，一个函数是func(float，int，unsignedchar)可能变成func__FfiUc。

类的名字被认为是有类型的，被编码成类的名字的长度后面跟着名字，如4Pair。类还可以含有多个层次的内部类的名字；这些”合法的”名字用Q来编码，后面是一个数字标明层次的数目，然后是类的名字，这样First::Second::Third变成了Q35First6Second5Third。也就是说一个有两个类参数的函数f(Pair，First::Second::Third)变成了f__F4PairQ35First6Second5Third。 类成员函数编码成函数名字，然后是两个下划线，然后是编码后的类的名字，然后是F和参数，这样，cl::fn(void)变成fn__2clFv。所有的操作符也都编码成4或5个字符的名字，如__ml代表*，而__aor代表

http://blog.sina.com.cn/s/blog_4bb59dc40100ea1d.html

以vc为例，
1、c和c++之间：
void foo(int x, int y);
该函数被C编译器编译后在库中的名字为_foo,而C++编译器则会产生像_foo_int_int之类的名字用来支持函数重载和类型安全连接.由于编译后的名字不同,C++程序不能直接调用C函数.C++提供了一个C连接交换指定符号extern"C"来解决这个问题.
2、不同编译器之间：
即使是按照c链接，但是不同的调用约定，比如__stdcall 和 __cdecl调用也会产生不同的名字改编。

关于调用约定

调用约定	堆栈清除	参数传递
__cdecl	调用者	从右到左,通过堆栈传递
__stdcall	函数体	从右到左,通过堆栈传递
__fastcall	函数体	从右到左,优先使用寄存器(ECX,EDX),然后使用堆栈
thiscall	函数体	this指针默认通过ECX传递,其它参数从右到左入栈

note：
（1）__cdecl是C\C++的默认调用约定;
VC的调用约定中并没有thiscall这个关键字,它是类成员函数默认调用约定;
C\C++中的main(或wmain)函数的调用约定必须是__cdecl,不允许更改;
默认调用约定一般能够通过编译器设置进行更改,如果你的代码依赖于调用约定,请明确指出需要使用的调用约定;
（2）常见的函数调用约定中,只有cdecl约定需要调用者来清除堆栈;C\C++中的函数支持参数数目不定的参数列表,比如printf函数;由于函数体不知道调用者在堆栈中压入了多少参数,所以函数体不能方便的知道应该怎样清除堆栈,那么最好的办法就是把清除堆栈的责任交给调用者;这应该就是cdecl调用约定存在的原因吧;

C编译在进行编译的时候也会进行名字的改编，当函数使用_stdcall(WINAPI)调用规则时，MS编译器就会改编函数的名称。
比如：__declspec(DLLexport) LONG __stdcall Proc(int a ,int b);

编译器会改编为__Proc@8

因此 当非C++或非C编译器调用该DLL中的函数Proc时，就会出现找不到函数的情况。

这样我们就可以定义DEF文件来解决，并且在DEF文件加上下面的EXPORTS：

```
EXPORTS
  Proc
```

Def模块执行原理：当连接程序分析这个DEF文件时，它发现Proc和__Proc@8都被输出，由于这两个函数是相互匹配的，因此连接程序使用Proc来输出该函数，根本不使用__Proc@8来输出函数名

(3)下面是

调用习惯	VC++命名	C++Builder命名
__stdcall	_MyFunction@4	MyFunction
__cdecl	MyFunction	_MyFunction

可以从网上搜索“在C++Builder里创建可以被Visual C++使用的DLL”以及“Using Visual C++ DLLs in a C++Builder Project”这两篇文章，看看不同编译器生成的dll之间是如何互相调用的。

第二部分 C/C++混合编程：

http://bbs.ednchina.com/BLOG_ARTICLE_479334.HTM

由于C++包含了C语言的特征，因此一个代码是C语言的还是C++的取决的是你要使用哪一个编译器去编译它，是C编译器(gcc)还是C++编译器(g++)．而不取决于使用的语法和文件后缀。

由于上面的name mangling机制，在C编译器和C++编译器中的不同，因此使用C的规则去找C++的函数是找不到的，反之亦然；因此需要明确的在代码中告诉编译器，下面的代码需要使用哪一种名字粉碎规则去匹配(在object文件中找函数名)．

1. 在C++中调用C语言的函数

extern "C"表示编译生成的内部符号名使用C约定。C++支持函数重载，而C不支持，两者的编译规则也不一样。函数被C++编译后在符号库中的名字与C语言的不同。例如，假设某个函数的原型为：void foo(int x, int y); 该函数被C编译器编译后在符号库中的名字可能为_foo，而C++编译器则会产生像_foo_int_int之类的名字（不同的编译器可能生成的名字不同，但是都采用了相同的机制，生成的新名字称为“mangled name”）。_foo_int_int这样的名字包含了函数名、函数参数数量及类型信息，C++就是靠这种机制来实现函数重载的。下面以例子说明，如何在C++中使用C的函数，或者在C中使用C++的函数。

C++引用C函数的例子(C++调用C，extern "C" 的作用是：让C++连接器找调用函数的符号时采用C的方式 如)

```
//test.c

#include <stdio.h>

void mytest()

{

 printf("mytest in .c file ok\n");

}

//main.cpp

extern "C"

{

void mytest(); //表示按照C语言的名字粉碎规则去调用这个函数 _mytest()

}

int main()

{

 mytest();
```

```
 return 0;

}
```

上述也可以加个头文件

```
//test.h

void mytest()
```

在后在main.cpp中extern "C"

```
{

#include “test.h”

}
```

2. C 程序中调用C++函数和变量

在C中引用C++函数(C调用C++，使用extern "C"则是告诉编译器把cpp文件中extern "C"定义的函数依照C的方式来编译封装接口，当然接口函数里面的C++语法还是按C++方式编译)

在C中引用C++语言中的函数和变量时，C++的函数或变量要声明在extern "C"{}里，但是在C语言中不能使用extern "C"，否则编译出错。(出现错误：error C2059: syntax error : 'string',这个错误在网上找了很久，国内网站没有搜到直接说明原因的，原因是extern "C"是C++中的关键词，不是C的，所有会出错。那怎么用？看本文，或者搜extern "C"！)

```
//test.cpp

#include <stdio.h>

extern "C"

{

void mytest() //这个函数是在C++中定义的，因为要给C调用，因此需要告诉C++编译器使用C的名字粉碎规
              则来生成这个函数名。可见，extern "C"的含义在两个地方是不同的。

{

 printf("mytest in .cpp file ok\n");

}

}

//main.c

void mytest();

int main()

{

 mytest();

 return 0;

}
```

3.综合调用（事先不知道谁调用谁)

一般我们都将函数声明放在头文件，当我们的函数有可能被C或C++使用时，我们无法确定被谁调用，使得不能确定是否要将函数声明在extern "C"里，所以，我们可以添加

```
#ifdef __cplusplus  //用预编译选项来确定使用哪种编译器
```

```
 extern "C"

 {

 #endif

//函数声明

#ifdef __cplusplus

 }

 #endif
```

这样的话这个文件无论是被C或C++调用都可以，不会出现上面的那个错误：error C2059: syntax error : 'string'。

如果我们注意到，很多头文件都有这样的用法，比如string.h，等等。

```
//test.h

#ifdef __cplusplus

#include <iostream>

using namespace std;

 extern "C"

 {

 #endif

void mytest();

#ifdef __cplusplus

 }

 #endif
```

这样，可以将mytest()的实现放在.c或者.cpp文件中，可以在.c或者.cpp文件中include "test.h"后使用头文件里面的函数，而不会出现编译错误。

```
//test.c

#include "test.h"

void mytest()

{

#ifdef __cplusplus

 cout << "cout mytest extern ok " << endl;

#else

 printf("printf mytest extern ok n");

#endif

}

//main.cpp

#include "test.h"

int main()
```

```
{

 mytest();

 return 0;

}
```

综合使用的另外一种解决方案(使用extern关键字)：

关于C++引用C函数和变量的例子还有一个

两个文件：

c文件：C.c

```
**********************************************
int external="5"; //全局变量，缺省为extern。
int func() //全局函数，缺省为extern。
{
return external;
}
**********************************************
```

cpp文件：CPP.cpp

```
**********************************************
#include "iostream"
using namespace std;
#ifdef __cplusplus
extern "C"
{
#endif
extern int external; //告诉编译器extern是在别的文件中定义的int，这里并不会为其分配存储空间。
extern int func(); //虽然这两个都是在extern "C"的{}里，但是仍然要显式指定extern，否则报错。
#ifdef __cplusplus //不仅仅是函数，变量也要放在extern "C"中。
}
#endif


void main(void)
{
cout<<"the value of external in c file is: "<<EXTERNAL<<ENDL;
external=10;
cout<<"after modified in cpp is : "<<FUNC()<<ENDL;
}
**********************************************
```

extern是C/C++语言中表明函数和全局变量作用范围（可见性）的关键字.，它告诉编译器，其声明的函数和变量可以在本模块或其它模块中使用。

1.对于extern变量来说，仅仅是一个变量的声明，其并不是在定义分配内存空间。如果该变量定义多次，会有连接错误

2.通常，在模块的头文件中对本模块提供给其它模块引用的函数和全局变量以关键字extern声明。也就是说c文件里面定义，如果该函数或者变量与开放给外面，则在h文件中用extern加以声明。所以外部文件只用include该h文件就可以了。而且编译阶段，外面是找不到该函数的，但是不报错。link阶段会从定义模块生成的目标代码中找到此函数。

3.与extern对应的关键字是static，被它修饰的全局变量和函数只能在本模块中使用。

总结：由于C++编译器和C编译器在name mangling命名粉碎规则上的不同，因此C和C++互相调用其的函数和变量时，需要进行处理，否则会找不到。一种方法就是在C++文件中使用extern ”C“，这个做法对C不生效，只能在C++文件中使用。

2 0

喜欢 赠金笔

分享：

前一篇：流媒体传输协议和TS流
后一篇：C语言中变量的作用域和生命周期

〈 前一篇
流媒体传输协议和TS流

后一篇 〉
C语言中变量的作用域和生命周期