extern "c"用法解析

字数1875 阅读14521 评论2 喜欢13

引言

C++保留了一部分过程式语言的特点，因而它可以定义不属于任何类的全局变量和函数。但是，C++毕竟是一种面向对象的程序设计语言，为了支持函数的重载，C++对全局函数的处理方式与C有明显的不同。
extern "C"的主要作用就是为了能够正确实现C++代码调用其他C语言代码。加上extern "C"后，会指示编译器这部分代码按C语言的进行编译，而不是C++的。由于C++支持函数重载，因此编译器编译函数的过程中会将函数的参数类型也加到编译后的代码中，而不仅仅是函数名；而C语言并不支持函数重载，因此编译C语言代码的函数时不会带上函数的参数类型，一般之包括函数名。
比如说你用C 开发了一个DLL 库，为了能够让C ++语言也能够调用你的DLL输出(Export)的函数，你需要用extern "C"来强制编译器不要修改你的函数名。

揭秘extern "C"

从标准头文件说起

```
#ifndef __INCvxWorksh  /*防止该头文件被重复引用*/
#define __INCvxWorksh

#ifdef __cplusplus    //__cplusplus是cpp中自定义的一个宏
extern "C" {          //告诉编译器，这部分代码按C语言的格式进行编译，而不是C++的
#endif

    /**** some declaration or so *****/

#ifdef __cplusplus
}
#endif

#endif /* __INCvxWorksh */
```

extern "C"的含义

extern "C" 包含双重含义，从字面上即可得到：首先，被它修饰的目标是“extern”的；其次，被它修饰的目标是“C”的。
被extern "C"限定的函数或变量是extern类型的；
1、extern关键字
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，该关键字告诉编译器，其声明的函数和变量可以在本模块或其它模块中使用。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。例如，如果模块B欲引用该模块A中定义的全局变量和函数时只需包含模块A的头文件即可。这样，模块B中调用模块A中的函数时，在编译阶段，模块B虽然找不到该函数，但是并不会报错；它会在链接阶段中从模块A编译生成的目标代码中找到此函数。
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。因此，一个函数或变量只可能被本模块使用时，其不可能被extern“C”修饰。

2、被extern "C"修饰的变量和函数是按照C语言方式编译和链接的
首先看看C++中对类似C的函数是怎样编译的。
作为一种面向对象的语言，C++支持函数重载，而过程式语言C则不支持。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。 例如，在C++中，函数void foo(int x, int y)与void foo(int x, float y)编译生成的符号是不相同的，后者为_foo_int_float。
同样地，C++中的变量除支持局部变量外，还支持类成员变量和全局变量。用户所编写程序的类成员变量可能与全局变量同名，我们以"."来区分。而本质上，编译器在进行编译时，与函数的处理相似，也为类中的变量取了一个独一无二的名字，这个名字与用户程序中同名的全局变量名字不同。

3、举例说明
（1）未加extern "C"声明时的连接方式
假设在C++中，模块A的头文件如下：

```
// 模块A头文件  moduleA.h
```

```
#ifndef MODULE_A_H
#define MODULE_A_H
int foo( int x, int y );
#endif
```

```
//在模块B中引用该函数：
// 模块B实现文件  moduleB.cpp
#include "moduleA.h"
foo(2,3);
```

实际上，在连接阶段，链接器会从模块A生成的目标文件moduleA.obj中寻找_foo_int_int这样的符号！

（2）加extern "C"声明后的编译和链接方式
加extern "C"声明后，模块A的头文件变为：

```
// 模块A头文件  moduleA.h
#ifndef MODULE_A_H
#define MODULE_A_H
extern "C" int foo( int x, int y );
#endif
```

在模块B的实现文件中仍然调用foo(2,3)，其结果是：

<1>A编译生成foo的目标代码时，没有对其名字进行特殊处理，采用了C语言的方式；

<2>链接器在为模块B的目标代码寻找foo(2,3)调用时，寻找的是未经修改的符号名_foo。

如果在模块A中函数声明了foo为extern "C"类型，而模块B中包含的是extern int foo(int x, int y)，则模块B找不到模块A中的函数；反之亦然。

extern “C”这个声明的真实目的是为了实现C++与C及其它语言的混合编程。

应用场合

- C++代码调用C语言代码、在C++的头文件中使用
  在C++中引用C语言中的函数和变量，在包含C语言头文件（假设为cExample.h）时，需进行下列处理：

  ```
  extern "C"
  {
  #include "cExample.h"
  }
  ```

  而在C语言的头文件中，对其外部函数只能指定为extern类型，C语言中不支持extern "C"声明，在.c文件中包含了extern "C"时会出现编译语法错误。

  ```
  /* c语言头文件：cExample.h */
  #ifndef C_EXAMPLE_H
  #define C_EXAMPLE_H
  extern int add(int x,int y);     //注:写成extern "C" int add(int , int ); 也可以
  #endif

  /* c语言实现文件：cExample.c */
  #include "cExample.h"
  int add( int x, int y )
  {
    return x + y;
  }

  // c++实现文件，调用add：cppFile.cpp
  extern "C"
  {
    #include "cExample.h"        //注：此处不妥，如果这样编译通不过，换成 extern "C" int add(int , int ); 可以通过
  }

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

```
  add(2,3);
  return 0;
}
```

如果C++调用一个C语言编写的.DLL时，当包括.DLL的头文件或声明接口函数时，应加extern "C"{}。

- 在C中引用C++语言中的函数和变量时，C++的头文件需添加extern "C"，但是在C语言中不能直接引用声明了extern "C"的该头文件，应该仅将C文件中将C++中定义的extern "C"函数声明为extern类型

```
//C++头文件 cppExample.h
#ifndef CPP_EXAMPLE_H
#define CPP_EXAMPLE_H
extern "C" int add( int x, int y );
#endif

//C++实现文件 cppExample.cpp
#include "cppExample.h"
int add( int x, int y )
{
  return x + y;
}

/* C实现文件 cFile.c
/* 这样会编译出错：#include "cExample.h" */
extern int add( int x, int y );
int main( int argc, char* argv[] )
{
  add( 2, 3 );
  return 0;
}
```