C/C++中的 extern 和extern“C“关键字的理解和使用（对比两者的异同）

原创 呋喃吖 于 2021-08-29 23:18:38 发布 520 收藏 20 版权

分类专栏： c++ C语言 文章标签： c++ c语言 extern extern”C“

前言

文章目录

不知道有人是否在意过C中的 `extern` 这个关键字，又或者说是否使用过该关键字，当学C++时候，我发现了在C++中有关键字 `extern"C"` 的用法，和C语言中的 `extern` 还是有区别的，所以今天来总结一些对他们的理解，和使用的方式。

`extern` 关键字的用法很简单，就是简简单单的声明，它可以明确的指出一个语句是声明；
比如 `extern int i` ;那么就说明 这是声明变量 i,而不是定义i,**声明是不开辟内存的，定义是开辟内存的。**
假如 `int i` ； 没有 `extern` 修饰，那就说明为定义，会分配内存空间的。

extern 的使用场景1

extern 可以声明一个变量，使得该变量是来自其他文件的变量在本文件可以被访问。

比如：创建两个文件 test.c 和 main.c文件；
在test.c文件中定义一个全局变量：

```
//test.c
int i = 20; //定义一个全局变量
```

在main.c文件，声明变量 i;

```
main.c文件
# include<stdio.h>
extern int i; //声明变量i，当编译链接时候，main.c文件就可以访问到test.c文件的i的值了;
int main()
{
    printf("%d",i);
    return 0;
}
```

这样我们就可以跨文件（test.c），在本文件（main.c）访问这个变量了；

---

extern 的使用场景2

但是上诉的使用方式并不好，假如我一个大工程，这个工程由超级多的文件，这些文件假如都要访问 `test.c` 文件的 变量 i,那么，只能在这些文件中，每个文件的开头都 声明变量 i,并且，假如我的 `test.c` ，不止一个定义一个变量i,有好多其他变量呢？在其他文件访问时候，都要声明好多变量，**这会使得书写难度很繁琐，并且维护成本也大**；

所以，一般，我们都把**声明语句放到头文件使用**，即我定义一个 test.h 的头文件；

然后，假如你在其他文件要使用改变量i,直接包含该头文件即可，
比如：test.h 头文件

```
//test.h 头文件
extern int i;
extern int j;
extern int k;
//...
//声明很多很多变量
```

在其他.c文件，只要包含该头文件，就可以啦，比如在 `main.c` 文件：

```
# include<stdio.h>
# include"test.h"
//extern int i; 不用写了
//extern int j;不用写了
//extern int k;不用写了
//...
//声明很多很多变量,都不用写了，因为包含了头文件，声明都在头文件中
int main()
{
    printf("%d %d %d",i,j,k);
    return 0;
}
```

---

extern的用法总结

extern一般用于声明，在.h文件中，声明变量或者函数（函数可以不加extern，但是最好加上，这样比较统一）；在其他文件要访问该变量函数时候，包含头文件就行哦。

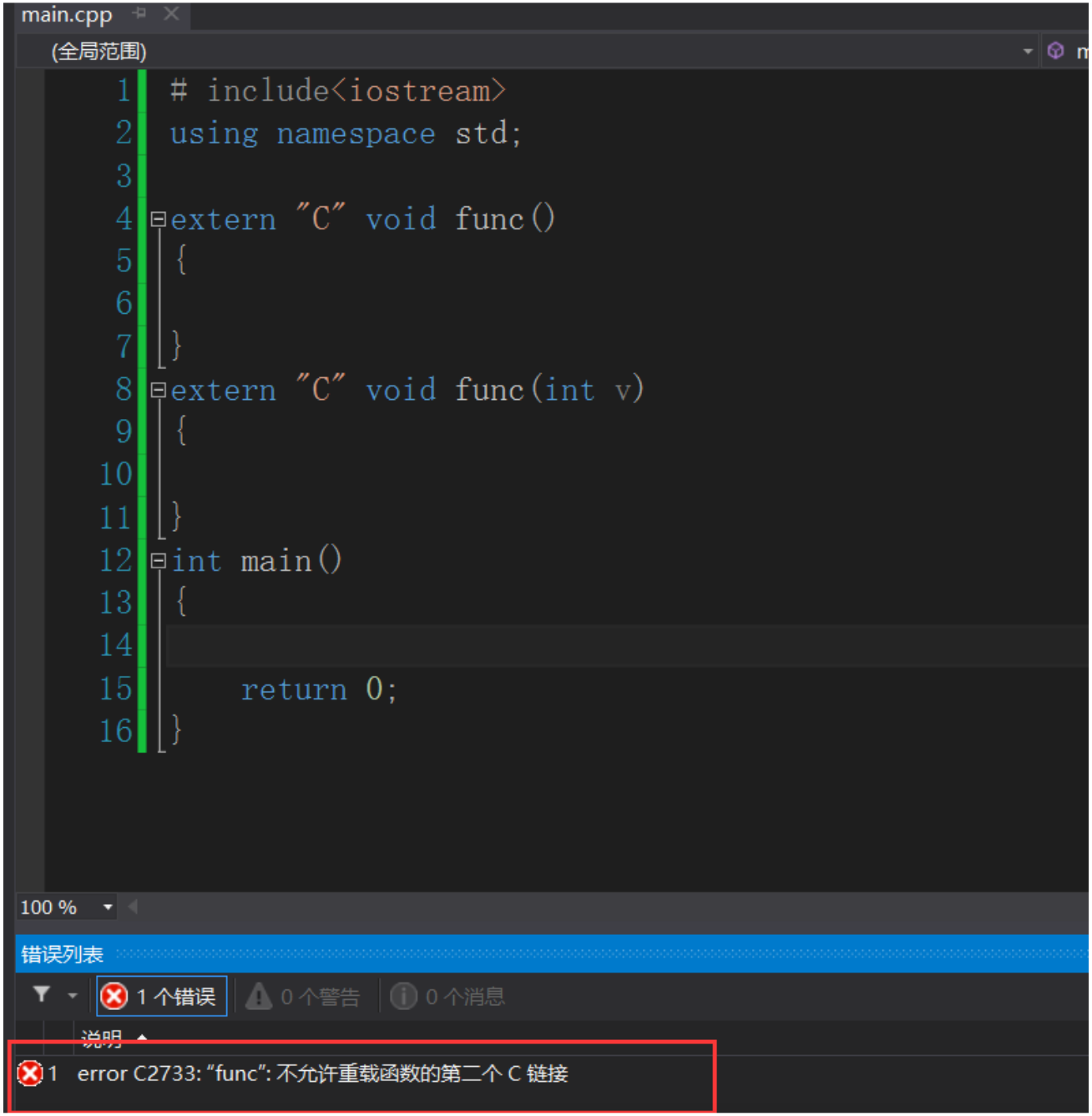
二. extern “C” 的理解和用法

在C++中，extern"C"修饰的语句是按照C语言的方式进行编译的

我来举几个例子：（理解它extern"C"的含义）
我们知道在C++中函数是可以发生重载的，即编译的时候并不会报错,在C语言中，是没有重载的说法的;那么假如我用extern"C"去修饰重载的函数的话，即在编译时候，就会按照C语言的方式去编译了：
这个时候，就会发生错误；
看例子：创建一个main.cpp文件

```
# include<iostream>
using namespace std;

extern "C" void func() //用 extern"C"修饰
{

}
extern "C" void func(int v)//用 extern"C"修饰
{
{

}
int main()
{

    return 0;
}
```

看下面的报错信息，不允许重载啊，本来我C++文件就是可以重载的，但是**用extern“C”修饰过后，就不可重载函数了，这就是按照C的编译方式编译**，假如按C++编译方式编译那就是能通过的。


main.cpp
(全局范围)
1 # include<iostream>
2 using namespace std;
3
4 extern "C" void func()
5 {
6
7 }
8 extern "C" void func(int v)
9 {
10
11 }
12 int main()
13 {
14
15 return 0;
16 }
100 %
错误列表
1 个错误
0 个警告
0 个消息
说明
1 error C2733: "func": 不允许重载函数的第二个 C 链接

extern"C" 也是可以修饰声明函数的，也说明该函数时按照C语言的方式编译；
extern"C" 也是可以用大括号{ } 的方式声明函数的；
如下例子：

```
extern "C" void func(); //用 extern"C"修饰声明函数
extern "C" void func(int v);//用 extern"C"修饰声明函数
//上诉例子会报错，C语言编译没有重载啊。

///
///
///

extern "C" { //修饰函数声明
    void func();
    void func(int v);
} //用大括号的方式一起写进来，也是可以。
//上诉例子会报错，C语言编译没有重载啊。

///
///
///

extern "C" { //修饰函数定义
    void func()
    {

    }
    void func(int v)
    {

    }
} //用大括号的方式一起写进来，也是可以。
//上诉例子会报错，C语言编译没有重载啊。

但是可以使用大括号的方式使用extern“C”，这是正确的
```

看看下面的例子：

```
# include<iostream>
using namespace std;

extern "C" void func() //用 extern"C"修饰
{

}
 void func(int v) //这个不用extern“C”声明
 {

 }

int main()
{

    return 0;
}
```

上面的情况**不会报错**，原因很简单： `extern "C" void func() { }` 按C语言方式编译，这个不会错； `void func(int v){ }` 按C++方式编译，所以也不会报错，这两者虽然函数名字相同，但是他们时按照不同的编译方式编译的，所以不报错；

---

extern"C "使用 在C与C++混合开发中的使用方式

通常在C++ 中，假如需要使用C语言中的库文件的话，可以使用extern "C"去包含；

那如何使用呢？接下来我一步一步带你理解如何使用，先举几个例子。

比如：创建math.c（C文件）中，有一些函数是数学的加法和减法功能函数。
我想在main.cpp文件中使用math.c文件中的函数，如何使用呢？
第一种办法：**在main.cpp文件用extern"C"包含math.c文件中的你想用的函数。**

```
//math.c文件
```

```
    return x+y;
}
int sub(int x,int y)//减法
{
    return x-y;
}
int mult(int x,int y)//乘法
{
    return x*y;
}
int div(int x,int y) //除法
{
    return x/y;
}
```

---

main.cpp（C++文件）中，要使用math.c文件中的函数.

```
//main.cpp文件
#include<iostream>
using namespace std;
extern “C” //用extern“C”{ }声明math.c文件中的函数，以至于可以在main.cpp文件使用。
{
    int add (int x,int y);
    int sub(int x,int y);
}

int main()
{
    cout<<add(10,20)<<endl; //由于有声明该函数，所以访问成功，结果30
    cout<<sub(10,20)<<endl;//由于有声明该函数，所以访问成功，结果-10
    return 0;
}
```

第二种使用方式：

假如我有一个需求，我想在main.cpp文件，用math.c文件中的其他函数，比如除法函数，乘法函数；
那么我就只能在 main.cpp 中 extern"C"{ }的括号中，逐个的加上这两个函数；这好像也可以正常使用没问题，但是，**这很麻烦啊，别人使用你的.c库时候会先得非常麻烦。**所以我们一般**使用头文件的方式，即创建多一个头文件math.h，在math.h里声明函数，在math.c文件定义函数，在main.cpp，即要使用该库文件里面，用extern"C"{ } ,在括号里包含该math.h头文件即可**

看下math.h头文件内容：

```
// math.h文件的内容
int add (int x,int y);//加法
int sub(int x,int y);//减法
int mult(int x,int y);//乘法
int div(int x,int y); //除法
```

在math.c中定义函数

```
//math.c文件

int add (int x,int y)//加法
{
    return x+y;
}
int sub(int x,int y)//减法
{
    return x-y;
}
int mult(int x,int y)//乘法
{
    return x*y;
}
int div(int x,int y) //除法
{
    return x/y;
}
```

这时候我们只要在main.cpp文件中，用extern “C”{ }包含该头文件就可以使用了。

```cpp
//main.cpp文件
#include<iostream>
using namespace std;

extern"C"{ //包含头文件，该里面是C语言方式编译的。
    # include"math.h"
}

int main()
{
    cout<<add(10,20)<<endl; //由于有声明该函数，所以访问成功，结果30
    cout<<sub(10,20)<<endl;//由于有声明该函数，所以访问成功，结果-10
    cout<<mutl(10,20)<<endl;
    cout<<div(10,20)<<endl;

    return 0;
}
```

这样就可以正常使用了。

但是，上面的使用方式还是不太好，该方式还是有一定的问题，什么问题呢？**就是我在main.cpp文件使用 `extern "C"{ #include"math.h"}` ,即要多写一个 `extern"C"{ }` ,显得格外麻烦，再说了，假如还有其他Cpp文件要用这个C语言的库呢。都是这种方式， `extern"C"` 太麻烦啦** 。我们之前都是直接 `#include"math.h"` ,这样才是我们平时的使用习惯呢。

所以有了这**第三种使用方式**：（**为的是直接在main.cpp中，#include"math.h"就可以使用，不用多谢extern"C"**）

直接在 `math.h` 中，用 `extern"C"` 的方式修饰就行啦。那么我就可以在 `main.cpp` 直接 `#include"math.h"` 。
(代码就不写啦，文字可以理解就行)

上面的使用有并不是完全好的方式，如何说呢？假如我有一个 `.c` 文件，创建为 `other.c` ，我这个文件呢，也想使用 `math.c` 的函数，那么我就可以在 `other.c` 文件直接包含该 `math.c` 文件就可以了，可是事实却不如所想，这样会报错，报错的原因是，由于在 `math.h` 文件中，声明函数用了 `extern"C"` ，而other.c文件 `# include"math.h"` ，会包含里面的所有内容，包括里面的

```c
//other.c文件
# include"math.h"
//包含头文件，由于这是.c文件，不是.cpp文件。所以.c文件不认识extern"C",所以会报错

void test()
{
    int ret = add(10,20);//报错，不认识，add.
}
```

所以如何解决上面问题呢？ 核心问题是，我希望有一种使用 extern"C"的方式，在C语言文件中，能够使用该库，在C++文件中也可以使用该库。

要解决这个问题：
也是 **第四种使用方式**

由于编译器默认会在，**你创建的任何一个.cpp文件中，默认定义一个宏** `#define __cplusplus` **,这个宏是你看不到的，是编译器默认给每个.cpp文件创建的，而在.c文件，即C语言的编译器，是没有这个宏的**，所以我们可以借助它，来在 math.h文件中使用下面的代码：

看math.h的代码

```c
// math.h文件的内容

//意思是如果使用该头文件math.h的文件定义了__cplusplus,
//则下面代码到#endif都是有效的,在这里是 extern "C" { 有效
#ifdef __cplusplus
extern "C" {
#endif  //__cplusplus

int add (int x,int y);//加法
int sub(int x,int y);//减法
int mult(int x,int y);//乘法
int div(int x,int y); //除法

//意思是如果使用该头文件 math.h 的文件定义了 __cplusplus
```

```
}
#endif  //__cplusplus
```

在main.cpp文件你直接包含该头文件math.h就可以使用了；
在other.c文件中你直接包含该头文件math.h就可以使用了；

只要按上面的方式书写，那么就可以完美的解决了在C文件和C++文件中混用的问题；你既可以在.c文件使用该头文件，也可以在.cpp文件中使用该文件库。

第四种方式是最值得推荐的。

三. 总结

两者 extern 和 extern"C"有点区别：

1. extren 在C和C++中，都表示声明语句的意思；
2. 而 extern"C"是按C语言方式去编译文件；
3. extern"C"只能在C++文件使用，extern在C和C++都可以使用；

显示推荐内容

评论 7 条 >

写评论

可口也可樂、 热评 非常详细 直接收藏爱了爱了

呋喃吖 关注

19 20 7