告别cpp&h：hpp文件编写心得

hpp，顾名思义等于.h加上.cpp，在boost、Xerces等开源库中频繁出现，偶在机缘巧合之下，学得一招半式，遂记录如下，以供参考学习。

hpp，其实质就是将.cpp的实现代码混入.h头文件当中，定义与实现都包含在同一文件，则该类的调用者只需要include该hpp文件即可，无需再将cpp加入到project中进行编译。而实现代码将直接编译到调用者的obj文件中，不再生成单独的obj，采用hpp将大幅度减少调用project中的cpp文件数与编译次数，也不用再发布烦人的lib与dll，因此非常适合用来编写公用的开源库。

hpp的优点不少，但是编写中有以下几点要注意：

1、不可包含全局对象和全局函数。

由于hpp本质上是作为.h被调用者include，所以当hpp文件中存在全局对象或者全局函数，而该hpp被多个调用者include时，将在链接时导致符号重定义错误。要避免这种情况，需要去除全局对象，将全局函数封装为类的静态方法。

2、类之间不可循环调用。

在.h和.cpp的场景中，当两个类或者多个类之间有循环调用关系时，只要预先在头文件做被调用类的声明即可，如下：

```
class B;
class A{
public:
    void someMethod(B b);
};
class B{
public:
    void someMethod(A a);
};
```

在hpp场景中，由于定义与实现都已经存在于一个文件，调用者必需明确知道被调用者的所有定义，而不能等到cpp中去编译。因此hpp中必须整理类之间调用关系，不可产生循环调用。同理，对于当两个类A和B分别定义在各自的hpp文件中，形如以下的循环调用也将导致编译错误：

2008年08月(2)
2007年08月(2)
2006年12月(1)
2006年11月(1)
2006年10月(1)
2006年06月(1)
2006年03月(1)
2006年02月(3)

```
//a.hpp
#include "b.hpp"
class A{
public:
   void someMethod(B b);
};

//b.hpp
#include "a.hpp"
class B{
public:
   void someMethod(A a);
};
```

3、不可使用静态成员。

静态成员的使用限制在于如果类含有静态成员，则在hpp中必需加入静态成员初始化代码，当该hpp被多个文档include时，将产生符号重定义错误。唯一的例外是const static整型成员，因为在vs2003中，该类型允许在定义时初始化，如：

```
class A{
public:
  const static int intValue = 123;
};
```

由于静态成员的使用是很常见的场景，无法强制清除，因此可以考虑以下几种方式（以下示例均为同一类中方法）

1.类中仅有一个静态成员时，且仅有一个调用者时，可以通过局域静态变量模拟

```
//方法模拟获取静态成员
someType getMember()
{
  static someType value(xxx);//作用域内静态变量
  return value;
}
```

2.类中有多个方法需要调用静态成员，而且可能存在多个静态成员时，可以将每个静态成员封装一个模拟方法，供其他方法调用。

```
someType getMemberA()
{
  static someType value(xxx);//作用域内静态变量
  return value;
```

```
}
someType getMemberB()
{
  static someType value(xxx);//作用域内静态变量
  return value;
}
void accessMemberA()
{
  someType member = getMemberA();//获取静态成员
};
//获取两个静态成员
void accessStaticMember()
{
  someType a = getMemberA();//获取静态成员
  someType b = getMemberB();
};
```

3.第二种方法对于大部分情况是通用的，但是当所需的静态成员过多时，编写封装方法的工作量将非常巨大，在此种情况下，建议使用Singleton模式，将被调用类定义成普通类，然后使用Singleton将其变为全局唯一的对象进行调用。

如原h+cpp下的定义如下：

```
class A{
public:
   type getMember(){
     return member;
   }
   static type member;//静态成员
}
```

采用singleton方式，实现代码可能如下（singleton实现请自行查阅相关文档）

```
//实际实现类
class Aprovider{
public:
   type getMember(){
     return member;
   }
   type member;//变为普通成员
```

```
}

//提供给调用者的接口类
class A{
public:
    type getMember(){
        return Singleton<AProvider>::getInstance()->getMember();
    }
}
```

发表于 @ 2006年10月31日 01:05:00 | 评论(2) | 举报 | 收藏

旧一篇:com初学心得小记 | 新一篇:委托与事件的故事（For c++ developer）

iwillbeback008 □□□Tuesday, January 06, 2009 22:00:48 □□

□□S3C2440□□□□□□□□□□□□□□□□

diablojj □□□Tuesday, July 07, 2009 17:46:37 □□

□□□□□□□□□□C □□□□□□□~~

□ □□ ccdjk

□□□□ ZCAV □□□□□□□

提交留言