最近在读完陈硕大牛的《Linux多线程服务端编程》以及muduo源码后，对其中的一些实现细节有着十分深刻的印象，尤其是使用std::bind和std::function的回调技术。可以说，这两个大杀器简直就是现代C++的“任督二脉”，甚至可以解决继承时的虚函数指代不清的问题。在此详细叙述使用std::bind和std::function在C++对象之间的用法，用以配合解决事件驱动的编程模型。笔者才疏学浅，如果解释的不正确希望朋友们不吝赐教。
下面的所有讨论基于对象。

std::bind和std::function的基础用法

```
#include<iostream>
#include<functional>

typedef std::function<void()> Functor;

class Blas
{
    public:
        void add(int a,int b)
        {
            std::cout << a+b << std::endl;
        }

        static void addStatic(int a,int b)
        {
            std::cout << a+b << std::endl;
        }
};

int main(int argc,char** argv)
{
    Blas blas;

//使用bind绑定类静态成员函数
    Functor functor(std::bind(&Blas::addStatic,1,2));

//使用bind绑定类的chengyuan函数
    Functor functor(std::bind(&Blas::add,blas,1,2));

    functor();
    return 0;
}
```

上述代码中的区别是：如果不是类的静态成员函数，需要在参数绑定时，往绑定的参数列表中加入使用的对象。

使用std::function和std::bind实现回调功能

```cpp
#include<iostream>
#include<functional>

typedef std::function<void()> Functor;

class Blas
{
    public:
        void setCallBack(const Functor& cb)
        {functor = cb;};

        void printFunctor()
        {functor();};

    private:
        Functor functor;
};

class Atlas
{
    public:
        Atlas(int x_) : x(x_)
        {
            //使用当前类的静态成员函数
            blas.setCallBack(std::bind(&addStatic,x,2));

            //使用当前类的非静态成员函数
            blas.setCallBack(std::bind(&Atlas::add,this,x,2));
        }

        void print()
        {
            blas.printFunctor();
        }

    private:
        void add(int a,int b)
        {
            std::cout << a+b << std::endl;
        }

        static void addStatic(int a,int b)
        {
            std::cout << a+b << std::endl;
        }
        Blas blas;
        int x;
};


int main(int argc,char** argv)
{
    Atlas atlas(5);
    atlas.print();
    return 0;
}
```

在以上代码中的

```cpp
void add();
void addStatic();
```

两个函数在Atlas类中，并且可以自由操作Atlas的数据成员。尽管是将add()系列的函数封装成函数对象传入Blas中，并且在Blas类中调用，但是它们仍然具有操作Atlas数据成员的功能，在两个类之间形成了弱的耦合作用。但是如果要在两个类之间形成弱的耦合作用，必须在使用 `std::bind()` 封装时，向其中传入this指针：

```cpp
std::bind(&Atlas::add,this,1,2);
```

也就是说，要在两个类之间形成耦合作用，要使用非静态的成员函数（私有和公有都可以）。代码如下：

```cpp
#include<iostream>
#include<functional>

typedef std::function<void()> Functor;

class Blas
{
    public:
        void setCallBack(const Functor& cb)
        {functor = cb;};

        void printFunctor()
        {functor();};

    private:
        Functor functor;
};

class Atlas
{
    public:
        Atlas(int x_,int y_) : x(x_),y(y_)
        {
            //使用当前类的非静态成员函数
            blas.setCallBack(std::bind(&Atlas::add,this,x,2));
        }

        void print()
        {
            blas.printFunctor();
        }

    private:

        void add(int a,int b)
        {
            std::cout << y << std::endl;
            std::cout << a+b << std::endl;
        }
        Blas blas;
        int x,y;
};


int main(int argc,char** argv)
{
    Atlas atlas(5,10);
    atlas.print();
    return 0;
}
```

这样，便可以Atlas便可以在Blas类中注册一些函数对象，这些函数对象在处理Blas数据的同时(在std::bind中预留位置传入Blas的参数)，还可以回带处理Atlas的数据，形成回调作用。代码如下：

```cpp
#include<iostream>
#include<functional>

typedef std::function<void(int,int)> Functor;

class Blas
{
    public:
        void setCallBack(const Functor& cb)
        {functor = cb;};

        void printFunctor()
        {functor(x,y);};

    private:
        int x = 10;
        int y = 10;
        Functor functor;
};
```

```
20.
21. class Atlas
22. {
23.     public:
24.         Atlas(int x_,int y_) : x(x_),y(y_)
25.         {
26.             //使用当前类的非静态成员函数
27.             blas.setCallBack(std::bind(&Atlas::add,this,std::placeholders::_1,std::placeholders::_2
    ));
28.         }
29.
30.         void print()
31.         {
32.             blas.printFunctor();
33.         }
34.
35.         void printFunctor()
36.         {functor(x,y);};
37.
38.     private:
39.         int x = 10;
40.         int y = 1:;
41.         Functor functor;
42. };
43.
44. class Atlas
45. {
46.     public:
47.         Atlas(int x_,int y_) : x(x_),y(y_)
48.         {
49.             //使用当前类的非静态成员函数
50.             blas.setCallBack(std::bind(&Atlas::add,this,std::placeholders::_1,std::placeholders::_2
    ));
51.         }
52.
53.         void print()
54.         {
55.             blas.printFunctor();
56.         }
57.
58.     private:
59.
60.         void add(int a,int b)
61.         {
62.             std::cout << y << std::endl;
63.             std::cout << a+b << std::endl;
64.         }
65.         Blas blas;
66.         int x,y;
67. };
68.
69.
70. int main(int argc,char** argv)
71. {
72.     Atlas atlas(5,10);
73.     atlas.print();
74.     return 0;
75. }
```