Std::function 和 std::bind 的使用：

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

struct Foo {
    Foo(int num) : num_(num) {}
    void print_add(int i) const { std::cout << num_+i << '\n'; }
    int num_;
};
// store a call to a member function and object
    using std::placeholders::_1;
    std::function<void(int)> f_add_display2= std::bind( &Foo::print_add,
foo, _1 );
    f_add_display2(2);

    // store a call to a member function and object ptr
    std::function<void(int)> f_add_display3= std::bind( &Foo::print_add,
&foo, _1 );
    f_add_display3(3);
```

cocos2dx 中的 std::function,std::bind：

```
class CC_DLL MenuItemLabel : public MenuItem
{
public:
    /** Creates a MenuItemLabel with a Label, target and selector. */
    CC_DEPRECATED_ATTRIBUTE static MenuItemLabel * create(Node*label, Ref* target, SEL_MenuHandler selector);

    /** Creates a MenuItemLabel with a Label and a callback. */
    static MenuItemLabel * create(Node*label, const ccMenuCallback& callback);
```

```
40    typedef std::function<void(Ref*)> ccMenuCallback;
```

接收的是一个 function<void(Ref*)>参数

```
305  #define CC_CALLBACK_0(__selector__,__target__, ...) std::bind(&__selector__,__target__, ##__VA_ARGS__)
306  #define CC_CALLBACK_1(__selector__,__target__, ...) std::bind(&__selector__,__target__, std::placeholders::_1,
       ##__VA_ARGS__)
307  #define CC_CALLBACK_2(__selector__,__target__, ...) std::bind(&__selector__,__target__, std::placeholders::_1,
       std::placeholders::_2, ##__VA_ARGS__)
308  #define CC_CALLBACK_3(__selector__,__target__, ...) std::bind(&__selector__,__target__, std::placeholders::_1,
       std::placeholders::_2, std::placeholders::_3, ##__VA_ARGS__)
```

CC_CALLBACK_X 就是一个 std::bind

```
//menuItem
auto item1 = MenuItemLabel::create(menuLabel1, CC_CALLBACK_1(HelloWorld::moveEvent, this,'W'));
```

所以等价于：

```
    //menuItem
    std::function<void(Ref*)> myCallBack = std::bind(&HelloWorld::moveEvent, this, std::placeholders::_1, 'W');
    auto item1 = MenuItemLabel::create(menuLabel1, myCallBack);
```

如何使用 CC_CALLBACK_X:
上次作业中的触摸事件：

```
//add touch listener
EventListenerTouchOneByOne* listener = EventListenerTouchOneByOne::create();
listener->setSwallowTouches(true);
listener->onTouchBegan = CC_CALLBACK_2(GameSence::onTouchBegan, this);
Director::getInstance()->getEventDispatcher()->addEventListenerWithSceneGraphPriority(listener, this);
```

```
76      public:
77
78          typedef std::function<bool(Touch*, Event*)> ccTouchBeganCallback;
79          typedef std::function<void(Touch*, Event*)> ccTouchCallback;
80
81          ccTouchBeganCallback onTouchBegan;
82          ccTouchCallback onTouchMoved;
83          ccTouchCallback onTouchEnded;
84          ccTouchCallback onTouchCancelled;
85
```

我们可以看到 onTouchBegan 是一个 std::function<bool(Touch*,Event*)>,因此我们绑定的函数必然是
std::bind(&__selector__,__target__,std::placeholder::_1,std::placeholder::_2,##__VA_ARGS__);
需要 2 个占位符，因此使用 CC_CALLBACK_2.也能确定我们的回调函数最少应该包含这 2 个类型的参数。

```
bool GameSence::onTouchBegan(Touch *touch, Event *unused_event) {

    return true;
}
```

简而言之，CC_CALLBACK_X 的使用只需要根据对应的 std::function 中的参数个数，确定 X，并确定我们需要写的回调函数的参数格式。
更简单的方法：0-3 都使用一遍..那个没有编译错误就用哪一个。(这样做的缺陷在于，如果使用了错误的 CC_CALLBACK,有时候在 VS 上也能运行，但是到打包成 exe 或者 apk 的时候就会出现错误)

参考网站：
http://en.cppreference.com/w/cpp/utility/functional/function
http://en.cppreference.com/w/cpp/utility/functional/bind
http://blog.csdn.net/eclipser1987/article/details/24406203
http://www.cnblogs.com/skysand/p/4247823.html