一、 函数对象

1. function对象

> C++语言中有几种可调用对象： 函数 、 函数指针 、 lambda表达式 、 bind创建的对象 以及 重载了函数调用运算符的类 。
>
> 在前面讲解函数指针的时候，提过，函数指针也是有类型的，函数指针的类型，由函数的返回值类型和参数共同决定。而function的出现让这个定义更为简化。function是一个模板类，可以用来表示函数的类型，当然需要在定义的时候，表示函数的返回值类型以及参数的类型。 除了能使用函数指针来调用函数之外，其实也可以声明一个 function 的对象，接收某一个函数，直接调用function的对象，也等同于调用函数。

1.1. 函数指针回顾

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

using namespace std;

int add (int a ,int b){
    return a + b ;
}

int main(){
    //函数指针
    int (*padd)(int , int) =add;
    cout << padd(3,4) <<endl;

    return 0 ;
}
```

1.2. function 使用

1. 接收全局函数

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

using namespace std;

void print(int a , string b){
    cout <<a <<" = " <<b << endl;
}

int main(){

    //使用function对象接收函数 <void (int , string)>
    //前面的void 表示返回值  <>表示接收的参数类型，个数要对应上。

    function<void (int ,string)> f = print;
    f(3, "奥巴马");

    return 0 ;
}
```

2. 接收lambda表达式

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

int main (){

    function<int (int ,int)> f = [](int a ,int b){return a + b ;};
    cout << f(1 , 2 ) <<endl;
        return 0 ;
}
```

3. 接收静态成员函数

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


class stu{
public:
    static int run(string name ,int age){
        cout <<age <<" 的 "<< name << " 在跑步... "<< endl;
    }
};


int main (){

    function <int (string ,int)> f = stu::run;
    f("奥巴马" , 19);

    return 0 ;
}
```

4. 接收成员函数

> 接收成员函数时，需要额外提供，调用函数的对象实体，可以是对象，可以是引用，可以是指针。也就是参数的第一个位置写调用这个函数的对象实体

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

using namespace std;

class stu{
public:
    int run(string name ,int age){
        cout <<age <<" 的 "<< name << " 在跑步... "<< endl;
    }
};


int main (){

    //实际上我门想的应该是这样：  因为run 函数的返回值是int，接收两个参数(string ,int)
    //但是不能这么做，因为这个函数是非静态函数，也不是全局函数，它是成员函数，不能直接调用，
    //如果编译通过，那么直接使用f("aa" , 18) ; 显然是不允许的。
    //function <int ( string , int )> f0 =  &stu::run;   //错误写法


    function <int (stu , string , int )> f1 =  &stu::run; //正确写法 对象
    function <int (stu& , string , int )> f2 =  &stu::run; //正确写法 对象引用
    function <int (stu* , string , int )> f3 =  &stu::run; //正确写法 指针

    //真正调用run();
    stu s1;
    f1(s1 , "张三" , 18 ); //会发生拷贝 这里指的是stu发生拷贝，

    stu s2;
    f2(s2 , "张三" , 18 ); //避免发生拷贝 ，  这里指的是stu发生拷贝，，引用指向实体

    stu s3;
    f3(&s3 , "张三" , 18 );
    return 0 ;
}
```

2. bind

> c++ 11推出的一套机制 `bind` ， 它可以预先把指定可调用实体的某些参数绑定到已有的变量，产生一个新的可调用实体，这种机制常常出现在回调函数中。说的简单点就是，有一个可以直接调用的实体，但是可以对这个实体进行一下包装，包装出来的实体，也可以调用，那么调用这个包装的实体，等同于调用原来的实体。

- `bind` 到底绑的是什么呢?

> 可以理解为把函数 、函数的实参以及调用函数的对象打包绑定到一起。然后用一个变量来接收它，这个变量也就是这些个打包好的整体。

2.1. 绑定全局函数

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

using namespace std;

int add3(int a ,int b){
    cout <<"执行了add函数" << endl;
    return a + b;
}

int main (){

    //可以看成是把p 绑定到add3上， 以后使用p() 等同于 add3()
    //但是bind的一个好处是，参数不用在后续调用p()的时候传递了，因为在绑定的时候已经传递进来了。
    auto p1 = bind(add3 , 3 ,4 );
    int result1 = p1();
    cout <<"result1 = "<< result1 << endl;

    //也可以是用function来接收，但是由于参数的值已经在bind的时候传递进去了，所以接收的时候
    //function的参数可以使用() 表示无参。否则在调用p1的时候，还必须传递参数进来
    function<int ()> p2 = bind(add3 , 5 ,4 );
    int result2 =p2();
    cout <<"result2 = "<< result2 << endl;

    return 0 ;
}
```

2.2. 绑定成员函数

> bind 也可以 bind 类中的某个成员函数

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

using namespace std;


class stu{
public :
    int run(string name ,int age){
        cout <<age << " 的 " << name << "在跑步" << endl;
        return 10;
    }
};

int main (){
    stu s ;

    function<int ()> f= bind(&stu::run , s , "张三" ,18);
    f();

    //*************************************************
    //使用auto 完成类型推断。
    auto b = bind(&stu::run,s,"张三",18 );
    b();

        return 0 ;
}
```

3.3. 使用 placeholders占位

> 在绑定的时候，如果一时半会还不想传递实参，那么可以使用 `placeholders` 来占位，第一个占位的索引位置是1，第二个占位的索引是2，依次类推。

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

using namespace std;


class stu{
public :
    int run(string name ,int age){
        cout <<age << " 的 " << name << "在跑步" << endl;
        return 10;
    }
};


int main(){

    stu s ;
    //placeholders::_1 , 表示run的第一个参数，现在先不给，后面调用的时候再给。
    //占位的个数不限制，可以全部占位，后面在传递真正的实参。
    auto b = bind(&stu::run,s,placeholders::_1,18 );
    b("王五");

    //当然使用function来接收的话，就显得复杂些，所以平常建议使用auto
    //因为在绑定的时候，并没有传递run函数的第一个实参，那么表示这个实参在调用的时候在传
    //function在接收的时候，必须与函数完全比配，所以 f的参数必须带上 string
    //如果所有的数据都在后面给了，那么<>里面的()就可以空着了
    stu s2 ;
    function <int (string)> f = bind(&stu::run,s2, placeholders::_1,19 );
    f("李四");

    return 0 ;
}
```