函数调用运算符

这一部分内容是比较重要的内容，因此单开一章节。

> 类可以重载函数调用运算符。

```
struct absInt{
    int operator()(int val)const{
        return val<0?-val:val;
        // 更好的转换应该是转换为unsigned int
        // return val<0:-(unsigned int)val:(unsigned int)val;
    }
};
```

这种类的行为看起来和函数无异。任何该类的对象，都可以调用**函数调用运算符**，

```
absInt obj;
int ui=obj(i);// 等价于 obj.operator()(i);
```

> 定义了调用运算符的类，该类的对象称之为**函数对象（function object）。

lambda就是函数对象

lambda本质上就是函数对象，当我们编写一个lambda后，编译器将表达式翻译为一个**未命名**类的未命名对象。在lambda表达式产生的类中定义了一个重载的函数调用运算符。

```
auto f=[](const string &a,const string &b){return a.size()<b.size();};
```

其行为类似下面的类的一个未命名对象：

```
class Foo{
public:
    bool operator()(const string &s1,const string &s2)const{    // 注意这里有个const
        return s1.size()<s2.size();
    }
};
```

其中的函数调用运算符是`const`成员函数，因此，我们默认情况下，不能修改捕获的变量（需要使用`volatile`，此时函数调用运算符就不是`const`成员函数了）。即如果lambda被声明为`volatile`，则调用运算符就不是`const`。

> 应当注意使用这个类代替lambda表达式时，应当注意，显示创建一个对象，即

```
stable_sort(vec.begin(),vec.end(),Foo());    // 这里是Foo(),而不是Foo
```

最后的Foo()表明创建一个Foo对象。只有创建了对象，才可以调用非static成员函数。

表示lambda及相应捕获行为的类