|  |  |
| --- | --- |
| 1. 使用function生成一个可调用对象 2. 使用mem\_fn生成一个可调用对象 3. 使用bind生成一个可调用对象 | Function<bool(const string&) fcn = &string::empty  Mem\_fn(&string::empty)  Bind(&string::empty,\_1) |
| 1. 因为将成员函数指针当做算法的运算对象是不可的(普通的函数指针是可调用对象)   Fp(\*it)错误，成员函数指针不能有这样的参数,必须有.\*与->\*和对象，但是下面3个方法省去了这两个东西   1. 使用function生成一个可调用对象 2. 我们告诉fucntion，empty是一个接受string参数并返回bool值得函数 3. 当一个function对象包含有一个指向成员函数的指针时，function类知道它必须使用正确的指向成员的指针运算符来执行函数调用。 4. ((\*it).\*p)(),p是fcn内部的一个指向成员函数的指针，it是迭代器\*it获得对象   2）function中的参数必须根据实际对象是指针还是引用进行调整为\*或者&   1. 使用mem\_fn生成一个可调用对象 2. 让编译器负责推断成员的类型。 3. 这个可以通过对象调用，也可以通过指针调用，就无需像1一样要调整 4. 实际上有一对重载的函数，可以推断可调用对象的类型 5. 使用bind生成一个可调用对象 6. 和funciton类似，必须将函数中用于执行对象的隐式形参转换成显示的 7. 和mem\_fn类似，bind生成的可调用对象的第一个实参既可以是string的指针，也可以是string的调用   就是：  调用一个指向成员函数的指针需要表明哪个对象  (Myscreen.\*pmf)();  (myscreen->\*pmf)();  那么放入算法中，算法是把参数放入pmf中，不可，于是我们就将需解决能为上面两种方式的，使pmf(\*it)变成(\*it).\*pmf,(it->\*pmf)  关于it是哪种，看容器存储的类型是\*还是不带\*，  若带\*就是->,表示指针  若不带\*就是.\*，表示对象 | |