C++标准程序库读书笔记

CH5 STL标准模板库

STL组件：

- 容器***containers***
- 迭代器***iterator***
- 算法***algorithm***

> STL的基本观念是数据和操作分离，数据有容器类别加以管理，操作则由可定制的算法定义之。迭代器在两者之间充当粘合剂，使任何算法可以和任何容器交互运作。

STL的一个根本特性是，所有组件可以针对任意型别运作，所谓STL就是表示其内的所有组件都是可接受任意型别的templates,前提是这些型别必须能够执行必要操作。

容器

容器类别用来管理一组元素：

1. 序列式容器
2. 关联式容器

序列式容器

- vector
- deques
- lists

> lists并没有提供以operator（）直接存取元素的能力，因为lists不支持随机存取，如果采用operator[]会导致不良效能。

- strings
- arrays

关联式容器 通常关联式容器是有二叉树生成的。关联式容器的主要差别在于元素的类型以及处理重复元素时的方式。

- sets
- multisets
- maps
- multimaps

容器配接器

- stacks
- queues
- priority queues

> 其中的元素可以拥有不同的优先权，所谓优先权，乃是基于程序员提供的排序准则而定义。

迭代器

- operator*
- operator++
- operator== operator!=
- operator=
- begin()
- end()

> begin()和end()形成了一个半开区间，从第一个元素开始，到哦最后一个元素的下一位置结束。

迭代器分类

- 双向迭代器

  > 双向迭代器可以双向行进，以递增运算前进或者以递减运算后退。

- 随机存取迭代器

  > 随机存取不仅具备双向迭代器的所有属性，还具备随机访问能力。 只有随机存取迭代器支持operator<,双向迭代器支持operator!=

算法

算法并非容器类别的成员函数，而是一种搭配迭代器使用的全局函数。这里所阐述的并非面向对象思维模式，而是泛型函数式编程思维模式。

这种处理方式的缺点

- 用法有失直观
- 某些数据结构和算法之间并不兼容
- 某些容器和算法之间虽然勉强兼容，但是毫无用处。

配接器

1. insert iterator 安插型迭代器
2. stream iterator 流迭代器
3. reverse iterator 逆向迭代器

安插型迭代器

- 如果你对某个元素设值，会引发对其所属群集的安插操作
- 单步前进不会造成任何动静

> 1. back inserters
> 2. front inserters
> 3. general inserters

更易型算法

更易性算法用在关联式容器上市，会出现错误，它会改变某位置上的值，进而破坏其已序特性，那就推翻了关联式容器的基本原则：容器内的元素总是根据某个排序准则自动排序。为了保证这个原则，关联式容器的所有迭代器均被声明为指向常量。

如何从关联容器中删除元素： 调用他们的成员函数即可（比如erase()）

为了追求高效率，要永远优先选用成员函数

判断式

STL要求，面对相同的值，判断式必须得出相同的结果。

1. 一元判断式

   > 检查唯一参数的某项特性

2. 二元判断式

   > 比较两个参数的特定属性

仿函数

如果你定义了一个对象，行为像函数，它就可以被当作函数来用。

什么才算是拥有函数行为呢？ 是指可以”使用小括号传递参数，借以调用某个东西“

容器内的元素

容器元素的条件：

- 可以透过copy构造函数进行复制。副本与原本必须相等，即所有的相等测试的结果都必须显示，原本与副本的行为一致。
- 必须透过assignment操作符完成复制操作。容器和算法都是用assignment操作符，才能以新元素改写旧元素。
- 必须可以透过析构函数完成销毁动作。因此析构函数不能被设计为private，依C++惯例，析构函数绝不能抛出任何异常。
- 对序列式容器而言，元素的default构造函数必须可用。
- 对于某些动作而言，必须定义operator==以执行相等测试。
- 在关联式容器中，元素必须定义出排序准则，缺省情况下是operator<，透过仿函数less<>被调用。

**value语意VSreference语意

STL只支持value语意，不支持reference语意。好处是：

- 元素的拷贝简单
- 使用reference时容易导致错误。

缺点是：

- 拷贝元素可能导致不好的效能，有时甚至无法拷贝
- 无法在数个不同的容器中管理同一份对象。

STL内部的错误处理与异常处理

使用STL时，必须满足以下条件：

- 迭代器务必有效。
- 一个迭代器如果指向逾尾位置，它并不指向任何对象，因此不能对它调用operator* 或 operator->
- 区间（range）必须是合法的：

  1. 用以指出某个区间的前后迭代器，必须指向同一个位置。
  2. 从第一个迭代器出发，必须可以到达第二个迭代器所指位置。

- 如果涉及的区间不只一个，第二区间及后继各区间必须拥有至少和第一区间一样多的元素。
- 覆盖动作中的目标区间必须拥有足够元素，否则必须采用插入型迭代器。