**一．STL 组件**

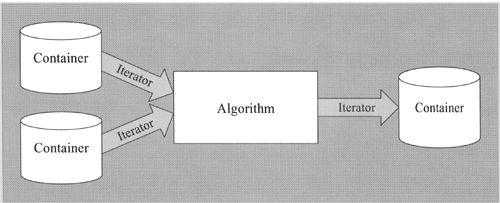
**1. 容器(Containers)**：用来管理某类对象的集合。

**2. 迭代器(Iterators)**：用来在一个对象群集的元素上进行遍历动件。迭代器的接口和一般指针差不多，以operator++累加，以operator\* 提取所指的值。

**3. 算法(Algorithms)**：用来处理群集内的元素。

STL 的基本观念就是将数据与操作分离。

数据由容器类加以管理，操作由算法定义，迭代器在两者之间充当粘合剂，使任何算法都可以和任何容器交互运作，如下图：

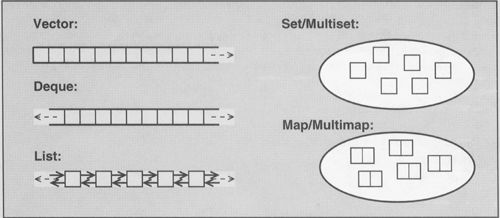


**二．容器**

容器可以分为两类：

**1. 序列式容器：vector， deque， list**

**2. 关联式容器：set， multiset， map， multimap**



**3. 容器配接器：stacks，queues， priority queues， string**

这是一些特别的容器，它们是根据基本容器类别实现来的。

**三．迭代器**

迭代器是一个“可以遍历 STL 容器内全部或部分元素”的对象。一个迭代器用来指出容器中的一个特定位置。

事实上每一种容器类型都有自己的迭代器，但它们对外的接口是相同的，这就是一种泛型程序设计。

**1. 迭代器的基本操作**

(1). **Operator\***

(2). **Operator++**

(3). **Operator==** 和 **operator!=**

(4). **Operator=**

**2. 获取迭代器**

所有容器都提供一些成员函数使我们获得迭代器，最重要的是：

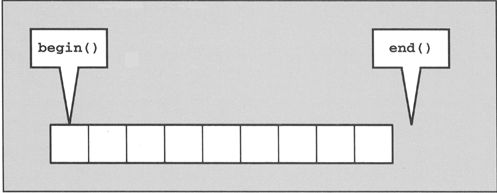
(1). **begin()** 返回一个迭代器，指向容器的起点，也就是第一个元素的位置。

(2). **end()** 返回一个迭代器，指向容器结束点，也就是最后一个元素之后的位置。

(1). **rbegin()** 返回一个迭代器，指向容器最后一个元素的位置。

(2). **rend()** 返回一个迭代器，指向容器第一个元素之前的位置。

注意：begin() 和 end() 形成了一个半开区间，如下图：



**3. 迭代器的类型**

任何容器都提供两种迭代器类形，读写模式和只读模式

(1). **Container::iterator** 读/写模式遍历元素。

(2).**Container::const\_iterator** 只读模式遍历元素。

**4. 迭代器的分类**

(1). **双向迭代器**

双向迭代器，可以以递增++运算前进或以递减--运算后退。

list，set，multiset，map，multimap 这些容器所提供的迭代器都属于此类。

(2). **随机存取迭代器**

随机存取不但具备双向迭代器的所有属性，还具备随机访问能力。

也就是它提供了“迭代器算法运算”必要的操作符，如：+n，-n，>，< 等。

vector，deque，string 这些容器的迭代器属于此类。

**注意：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/7301443)

1. list<**char**>::iterator pos;
2. for (pos = coll.begin(); pos != coll.end(); ++pos) {
3. \*pos = toupper(\*pos);
4. }
5. for (pos = coll.begin(); pos != coll.end(); pos++) {
6. ^^^^^ // OK, but slower
7. ...
8. }

list<char>::iterator pos;

for (pos = coll.begin(); pos != coll.end(); ++pos) {

\*pos = toupper(\*pos);

}

for (pos = coll.begin(); pos != coll.end(); pos++) {

^^^^^ // OK, but slower

...

}

++pos 要比 pos++ 效率高，因为后者需要一个额外的临时对象来存放迭代器的原来位置并将它返回，所以一般情况下最好用 ++pos。

**四．算法**

算法并非容器类形的成员函数，而是搭配迭代器使用全局函数。

这里体现的是泛型函数式编程思维模式，而不是OOP 模式，因为在OOP 概念里，数据和操作是一体的。

**注意：**

1. 所有的算法都用来处理一个或多个区间内的元素，所以我们必须将区间首尾当做两个参数传给算法。调用者必须保证这个区间的有效性。

2. 所有算法处理的都是半开区间 [begin, end )。

3. 在处理多个区间时，要保证第二个区间能够容纳第一个区间的所有元素。

**五. 迭代器配接器**

迭代器配接器，也就是一类特殊的迭代器。主要有以下三种：

1. 安插型迭代器 (Insert iterators)

2. 流迭代器 (Stream iterators)

3. 逆向迭代器 (Reverse iterators)

**六. 容器内的元素**

**容器元素的条件：**

STL 容器内的元素必须满足下面三个基本要求：

1. 必须是可以用拷贝构造函数进行复制的。

2. 必须是可以用 ＝操作符完成赋值操作的。

3. 必须是可以用析构函数完成销毁操作的。

**Value 与 Reference**

所有容器都会建立元素副本，并返回该副本。这意味着 STL 只提供 Value 语意。

STL 只支持 Value 语意，不支持 Reference 语意。如果你想要支持 Reference 语意，你可以用指针作为元素来实现，但是要非常谨慎。

**七. 错误与异常**

**1. 错误处理**

STL 的设计原则是效率优先，安全次之。而错误检查相当花时间，所以在 STL 里几乎没有错误处理。

所以对操作人员来说，要自己保证使用 STL 的正确性，或者自己增加一层包装来实现STL的安全性。

**2. 异常处理**

(1). 所有以节点实现的容器，如 list， set， multiset， mpa， multimap，如果节点构造失败，容器保持不变。

(2). 移除(removing)节点的动作保证不会失败。

(3). 对关联式容器插入(insert)多个元素，如果失败，则无法完全恢复原状。

(4). 对关联式容器插入单一个元素，要么成功，要么没有任何影响。

(5). 所有擦除(erase)操作，无论是针对单一元素或针对多重元素，肯定会成功。

(6). vector，deques类型的容器，在安插(insert)元素时如果失败，都不可能做到完全回复。