C++ Programming

标准模板库STL
Standard Template Library

2023年6月7日

学而不厌 诲人不倦

Chapter 8 标准模板库STL

- 8.1 类模板（复习）
- 8.2 STL与泛型编程
- 8.3 容器Containers
- 8.4 迭代器Iterators

类模板的作用

对于功能相同而只是数据类型不同的函数，可以定义**函数模板**。
对于功能相同的类而数据类型不同，不必定义出所有类，只要定义一个可对任何类进行操作的**类模板**。

```
//比较两个整数的类
class Compare_int
{private:
    int x,y;
public:
   Compare_int(int a,int b)
    {x=a;y=b;}
   int max()
    {return (x>y)?x:y;}
   int min()
    {return (x<y)?x:y;}
};
```

```
//比较两个浮点数的类
class Compare_float
 {private:
    float x,y;
public:
   Compare_float(float a,float b)
    {x=a;y=b;}
   float max()
    {return (x>y)?x:y;}
   float min()
    {return (x<y)?x:y;}
};
```

类模板的定义格式

```
template <class numtype>
class Compare
 { private:
    numtype x,y;
     public:
   Compare(numtype a,numtype b) //  构造函数
    {  x=a;y=b;}
   numtype max()
    {  return (x>y)?x:y;}
   numtype min()
    {  return (x<y)?x:y;}
  };
```

类模板的定义格式

```
template<class numtype>
class Compare
 { private:
    numtype x,y;
     public:
    //构造函数
    Compare(numtype a,numtype b)
     {  x=a;y=b;}
   numtype max()
     {  return (x>y)?x:y;}
   numtype min()
     {  return (x<y)?x:y;}
  };
```

类模板外定义max和min成员函数

```
numtype Compare< numtype > ::max()
{  return (x>y)?x:y;}
numtype Compare< numtype > ::min()
{  return (x<y)?x:y;}
```

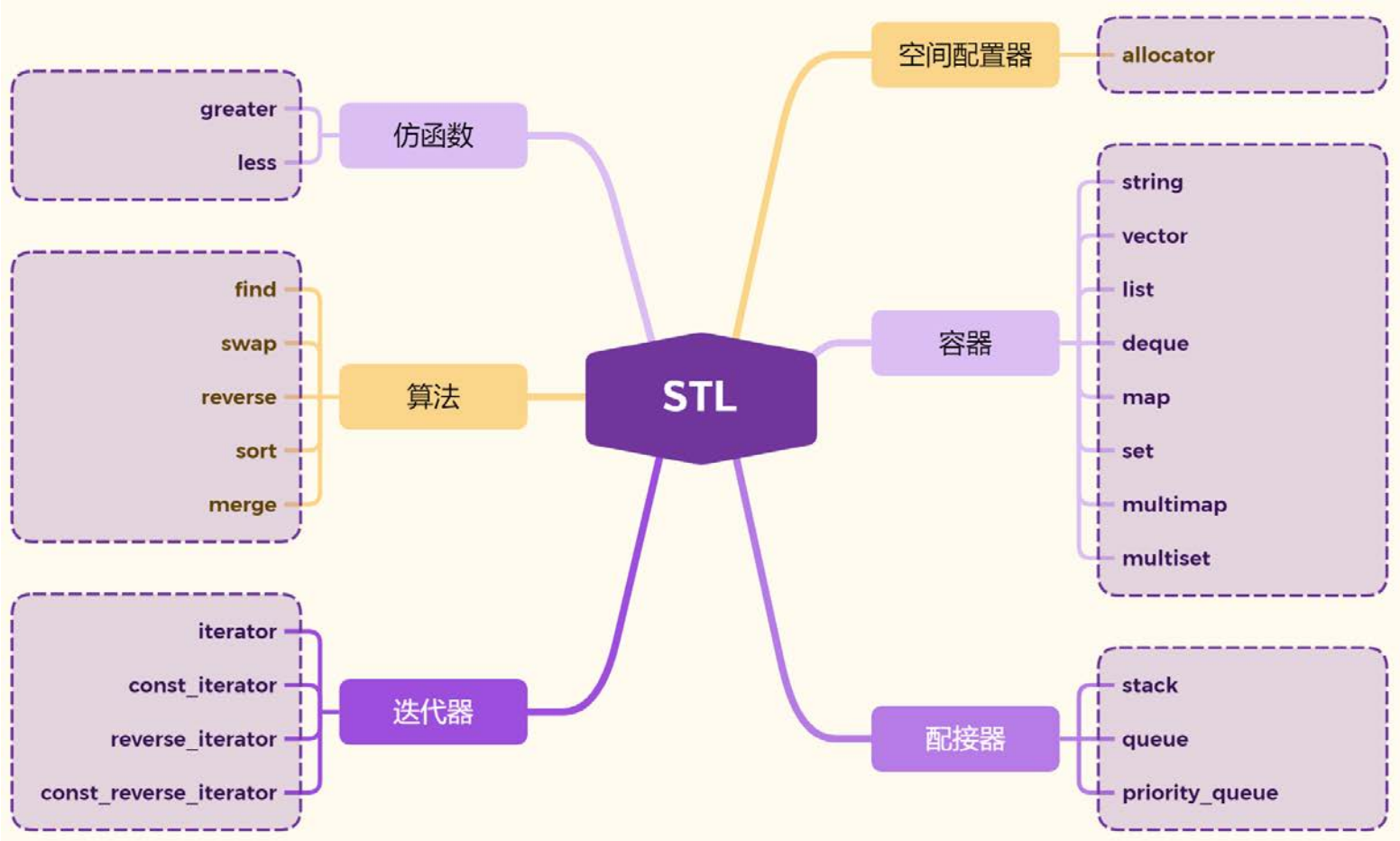
类模板的使用

```
int main()
{  Compare<int> cmp1(3,7);
   Compare<float> cmp2(45.78,93.6);
    Compare<char> cmp3('a','A');
    return 0;
}
```

8.2 STL与泛型编程

➢ STL 标准模板库 Standard Template Library

STL有六大组件，但主要包含容器、迭代器和算法三个部分。



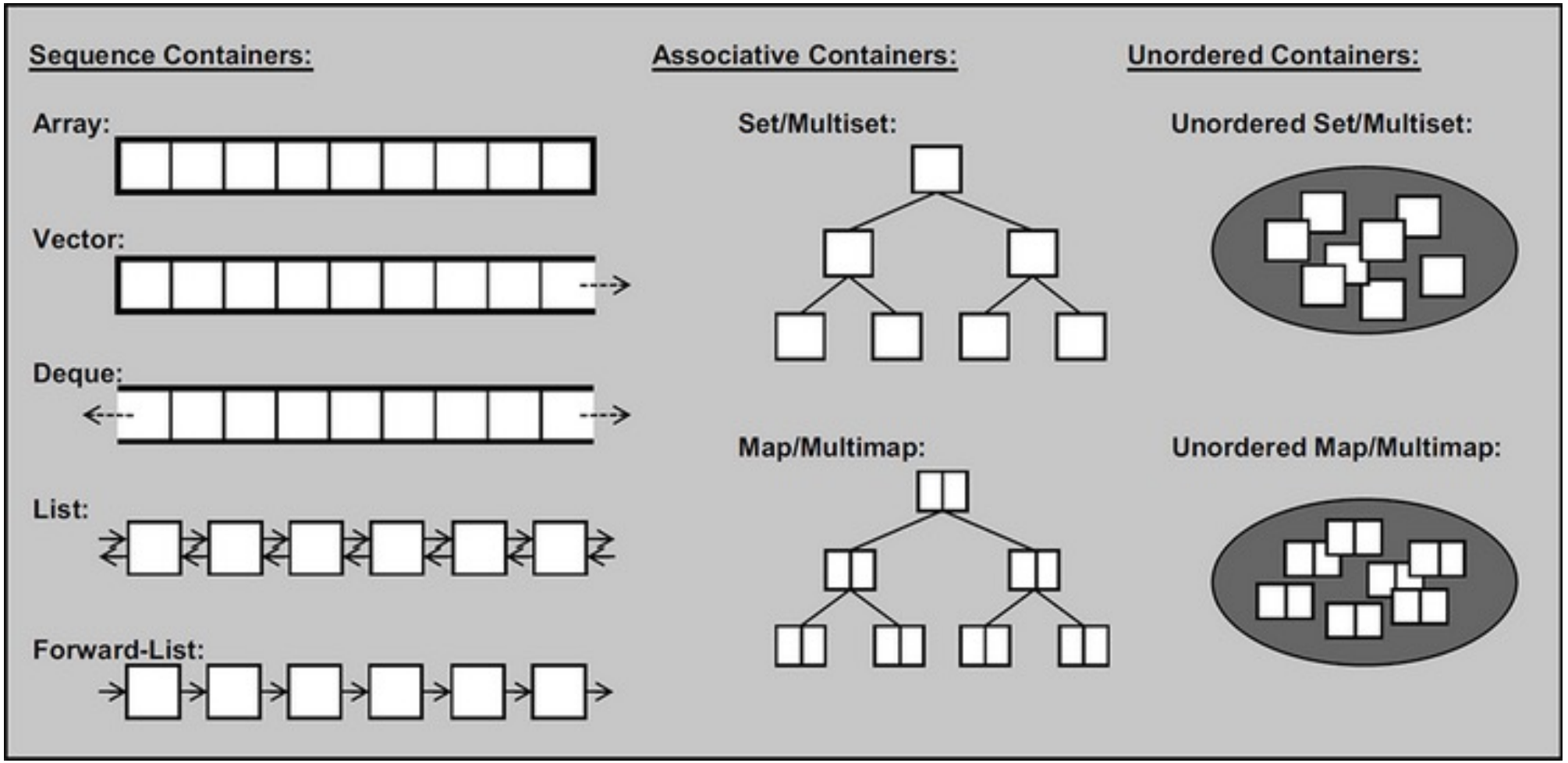
Cup<T>

Cup<Water>
Cup<Coffee>
Cup<Tea>

- 可扩展的应用框架，高度体现可复用性。
- **泛型化程序设计**的思想
- 类型参数化，基于模板

➢ STL 标准模板库

STL有六大组件，但主要包含容器、迭代器和算法三个部分。



➢ STL 标准模板库

C++常见操作：

1. 新申请一个较大的内存空间：

```
int * temp = new int[m];
```

2. 将原内存空间的数据全部复制到新申请的内存空间中：

```
memecpy(temp, p, sizeof(int)*n);
```

3. 将原来的堆空间释放：

```
delete [] p; p = temp;
```

➢ STL 标准模板库

```
vector <int> a; //定义 a 数组，当前数组长度为 0，但和普通数组不同的是，此数组 a 可以根据存储数据的数量自动变长。
//向数组 a 中添加 10 个元素
for (int i = 0; i < 10 ; i++)
a.push_back(i)
//还可以手动调整数组 a 的大小
a.resize(100);
a[90] = 100;
//还可以直接删除数组 a 中所有的元素，此时 a 的长度变为 0
a.clear();
//重新调整 a 的大小为 20，并存储 20 个 -1 元素。
a.resize(20, -1)
```

http://c.biancheng.net/view/6749.html

- 8.1 类模板（复习）
- 8.2 STL与泛型编程
- 8.3 容器 Containers
- 8.4 迭代器 Iterators

8.3 容器 Containers

➢ 为什么使用容器？

Think of your container as a file cabinet!

Organization

Related data can be packaged together!

Standardization

Common features are expected and implemented

Abstraction

Complex ideas made easier to utilize by clients

All containers are collections of objects…

```
for (initialization; termination condition; increment)
```

8.3 容器 Containers

➢ 容器的定义

Container: An object that allows us to collect other objects together and interact with them in some way.

```
vector <int> v;
deque <int> d;
list <int> l;
```

```
int num=20;
double value =1.0;
std::vector<double> values(num, value);

int array[]={1,2,3};
std::vector<int>values(array, array+2);//values 将保存{1,2}
std::vector<int>value1{1,2,3,4,5};
//value2保存{1,2,3}
std::vector<int>value2(std::begin(value1),std::begin(value1)+3);
```

8.3 容器 Containers

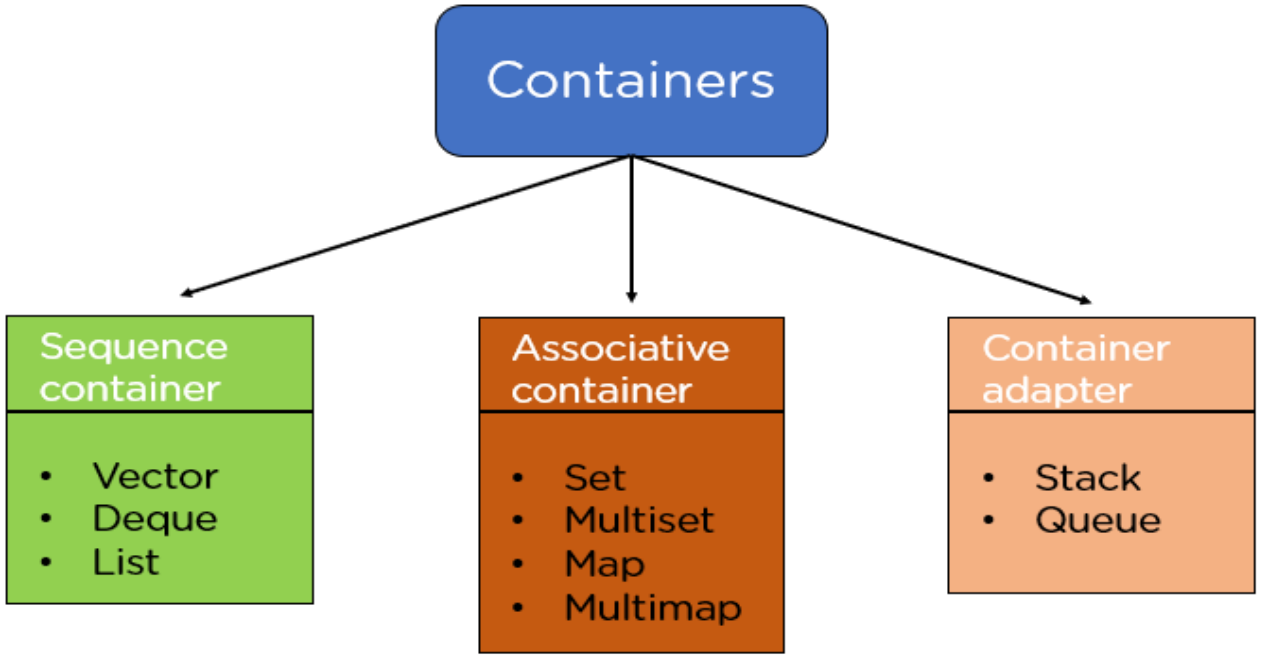
➢ 容器的分类

Sequence:

- Containers that can be accessed sequentially
- Anything with an inherent order goes here!

Associative

- Containers that don't necessarily have a sequential order
- More easily searched
- Maps and sets go here!



8.3 容器 Containers

➢ Vector容器

表 1 vector 容器的成员函数

函数成员	函数功能
begin()	返回指向容器中第一个元素的迭代器。
end()	返回指向容器最后一个元素所在位置后一个位置的迭代器，通常和 begin() 结合使用。
rbegin()	返回指向最后一个元素的迭代器。
rend()	返回指向第一个元素所在位置前一个位置的迭代器。
cbegin()	和 begin() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
cend()	和 end() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
crbegin()	和 rbegin() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
crend()	和 rend() 功能相同，只不过在其基础上，增加了 const 属性，不能用于修改元素。
size()	返回实际元素个数。
max_size()	返回元素个数的最大值。这通常是一个很大的值，一般是 $2^{32}-1$，所以我们很少会用到这个函数。
resize()	改变实际元素的个数。
capacity()	返回当前容量。
empty()	判断容器中是否有元素，若无元素，则返回 true；反之，返回 false。
reserve()	增加容器的容量。
shrink _to_fit()	将内存减少到等于当前元素实际所使用的大小。
operator[ ]	重载了 [ ] 运算符，可以向访问数组中元素那样，通过下标即可访问甚至修改 vector 容器中的元素。
at()	使用经过边界检查的索引访问元素。
front()	返回第一个元素的引用。
back()	返回最后一个元素的引用。
data()	返回指向容器中第一个元素的指针。
assign()	用新元素替换原有内容。
push_back()	在序列的尾部添加一个元素。
pop_back()	移出序列尾部的元素。
insert()	在指定的位置插入一个或多个元素。
erase()	移出一个元素或一段元素。
clear()	移出所有的元素，容器大小变为 0。
swap()	交换两个容器的所有元素。
emplace()	在指定的位置直接生成一个元素。
emplace_back()	在序列尾部生成一个元素。

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    //初始化一个空vector容量
    vector<char>value;
    //向value容器中的尾部依次添加 S、T、L 字符
    value.push_back('S');
    value.push_back('T');
    value.push_back('L');
    //调用 size() 成员函数容器中的元素个数
    printf("元素个数为: %d\n", value.size());
    //使用迭代器遍历容器
    for (auto i = value.begin(); i < value.end(); i++)
        cout << *i << " ";
    cout << endl;
    value.insert(value.begin(), 'C'); //开头插入字符
    cout << "首个元素为: " << value.at(0) << endl;
    return 0;
}
```

Vector implementation

We keep track of a few member variables:

- **_size** = number of elements in the vector
- **_capacity** = space allocated for elements

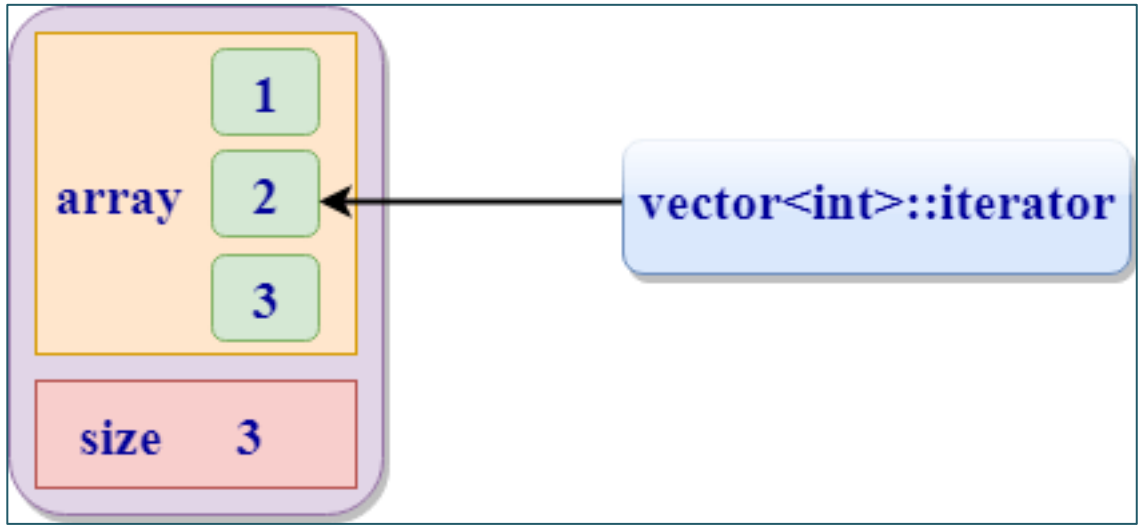
Don't confuse these two!

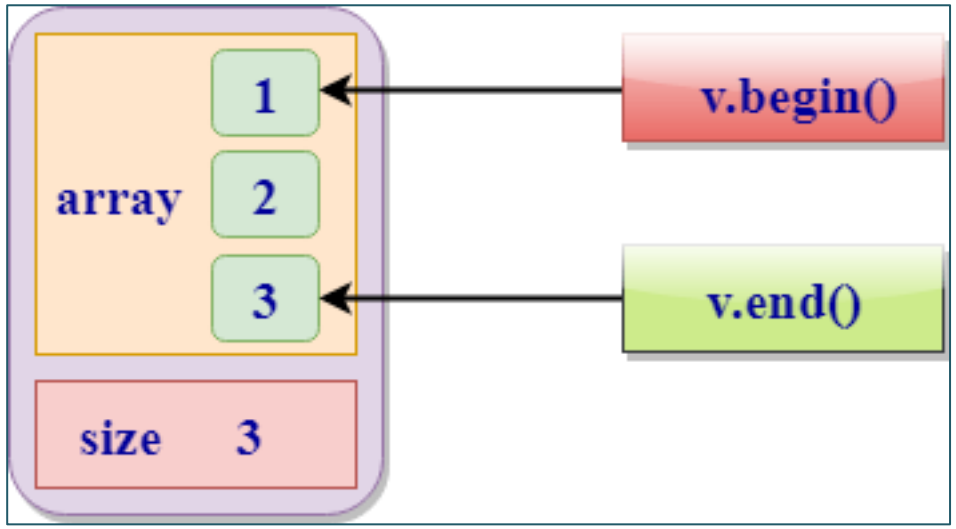
- 8.1 类模板（复习）
- 8.2 STL与泛型编程
- 8.3 容器 Containers
- 8.4 迭代器 Iterators

8.4 迭代器 Iterators

➢ 迭代器的定义和作用

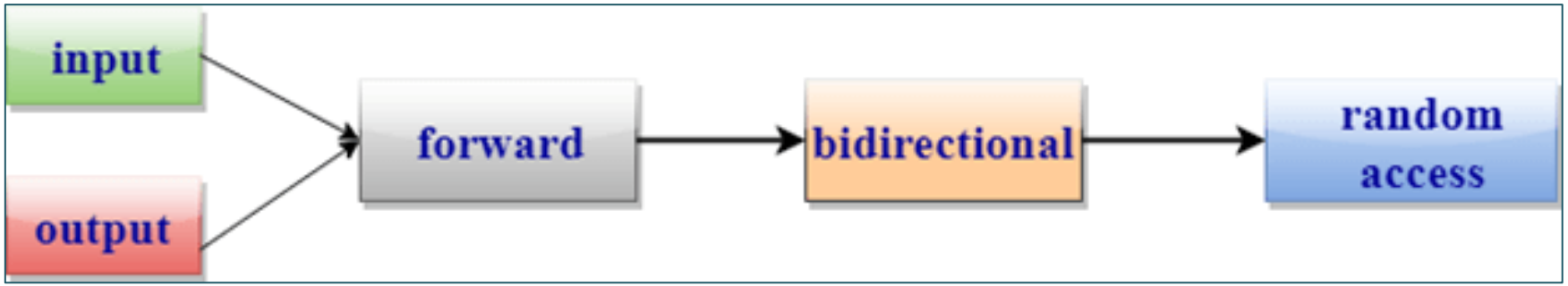
迭代器用于遍历对象集合的元素。这些集合可能是容器，也可能是容器的子集。
迭代器是类似指针的实体，用于访问容器中的各个元素。
迭代器从一个元素顺序移动到另一个元素。此过程称为遍历容器。

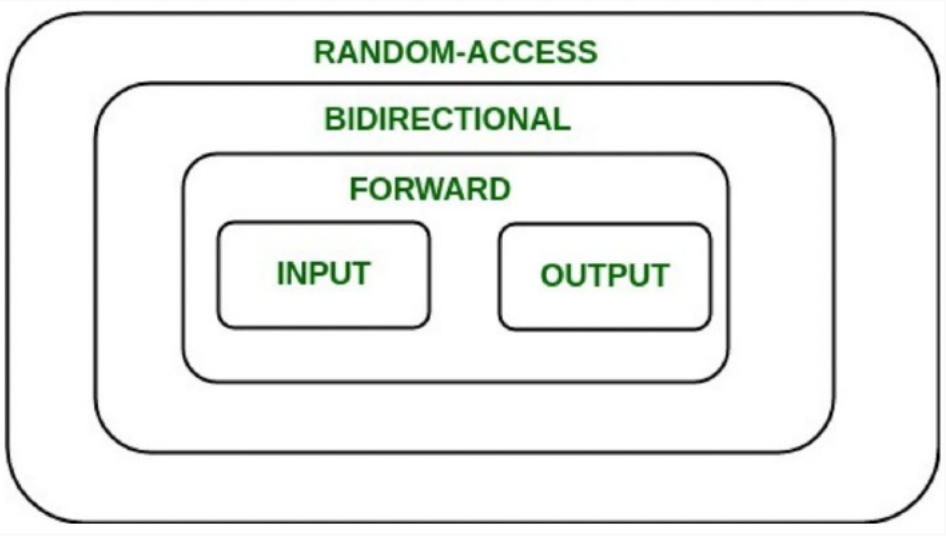




8.4 迭代器 Iterators

➢ 迭代器的定义和作用

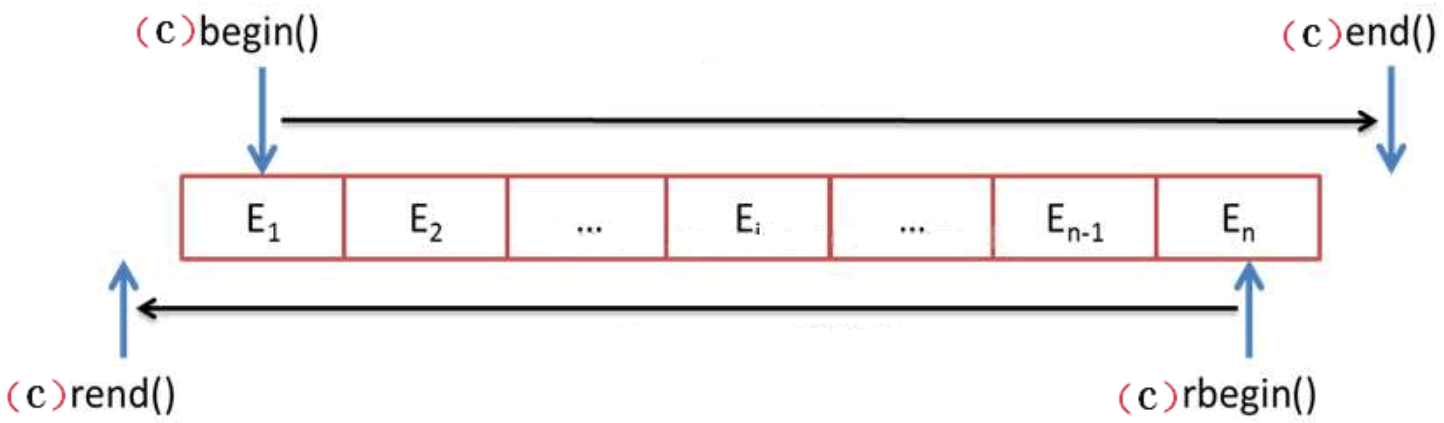




- All iterators implement a few shared operations:
  - Initializing → iter = s.begin();
  - Incrementing → ++iter;
  - Dereferencing → *iter;
  - Comparing → iter != s.end();
  - Copying → new_iter = iter;

8.4 迭代器 Iterators

➢ Vector容器的迭代器实例




```
vector<int>values{1,2,3,4,5};
auto first = values.cbegin();
auto end = values.cend();
while (first != end)
{
//*first = 10;不能修改元素
cout << *first << " ";
++first;
}
```

```
for (auto first = values.rbegin(); first != values.rend(); ++first) {
    cout << *first << " ";
}
```

注： C++11 赋予 auto 关键字新的含义，使用它来做自动类型推导。也就是说，使用了 auto 关键字以后，编译器会在编译期间自动推导出变量的类型。

8.4 迭代器 Iterators

➢ 迭代器实例

表 1 不同容器的迭代器

容器	对应的迭代器类型
array	随机访问迭代器
vector	随机访问迭代器
deque	随机访问迭代器
list	双向迭代器
set / multiset	双向迭代器
map / multimap	双向迭代器
forward_list	前向迭代器
unordered_map / unordered_multimap	前向迭代器
unordered_set / unordered_multiset	前向迭代器
stack	不支持迭代器
queue	不支持迭代器

表 2 迭代器的 4 种定义方式

迭代器定义方式	具体格式
正向迭代器	容器类名::iterator 迭代器名;
常量正向迭代器	容器类名::const_iterator 迭代器名;
反向迭代器	容器类名::reverse_iterator 迭代器名;
常量反向迭代器	容器类名::const_reverse_iterator 迭代器名;

```
#include <iostream>
#include <vector>//需要引入 vector 头文件
using namespace std;
int main() //遍历 vector 容器。
{
    vector<int> v{1,2,3,4,5,6,7,8,9,10}; //v被初始化成有10个元素
    cout << "第一种遍历方法: " << endl;
    for (int i = 0; i < v.size(); ++i) //size返回元素个数
        cout << v[i] <<" "; //像普通数组一样使用vector容器
    //创建一个正向迭代器，当然，vector也支持其他 3 种定义迭代器的方式
    cout << endl << "第二种遍历方法: " << endl;
    vector<int>::iterator i;
    for (i = v.begin(); i != v.end(); ++i) //用 != 比较两个迭代器
        cout << *i << " ";
    cout << endl << "第三种遍历方法: " << endl;
    for (i = v.begin(); i < v.end(); ++i) //用 < 比较两个迭代器
        cout << *i << " ";
    cout << endl << "第四种遍历方法: " << endl;
    i = v.begin();
    while (i < v.end()) { //间隔一个输出
        cout << *i << " ";
        i += 2; // 随机访问迭代器支持 "+= 整数"  的操作
    }
}
```

小结

- 8.1 类模板（复习）
- 8.2 STL与泛型编程
- 8.3 容器Containers
- 8.4 迭代器Iterators

Thank You !

Q & A