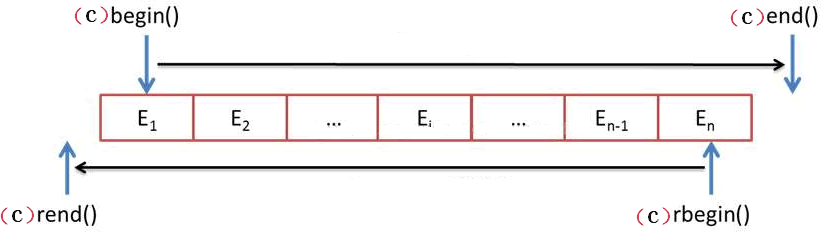
# 只有运用迭代器，才能访问 list 容器中存储的各个元素。list 模板类提供了

# 如表 1 所示的这些迭代器函数。

|  |  |
| --- | --- |
| 表 1 list 容器迭代器函数 | |
| **迭代器函数** | **功能** |
| begin() | 返回指向容器中第一个元素的双向迭代器（正向迭代器）。 |
| end() | 返回指向容器中最后一个元素所在位置的下一个位置的双向迭代器。（正向迭代器）。 |
| rbegin() | 返回指向最后一个元素的反向双向迭代器。 |
| rend() | 返回指向第一个元素所在位置前一个位置的反向双向迭代器。 |
| cbegin() | 和 begin() 功能相同，只不过在其基础上，正向迭代器增加了 const 属性，即不能用于修改元素。 |
| cend() | 和 end() 功能相同，只不过在其基础上，正向迭代器增加了 const 属性，即不能用于修改元素。 |
| crbegin() | 和 rbegin() 功能相同，只不过在其基础上，反向迭代器增加了 const 属性，即不能用于修改元素。 |
| crend() | 和 rend() 功能相同，只不过在其基础上，反向迭代器增加了 const 属性，即不能用于修改元素。 |

除此之外，[C++](http://c.biancheng.net/cplus/) 11 新添加的 begin() 和 end() 全局函数也同样适用于 list 容器。即当操作对象为 list 容器时，其功能分别和表 1 中的 begin()、end() 成员函数相同。

表 1 中各个成员函数的功能如图 2 所示。

  
图 2 list 容器迭代器的功能示意图

注意，list 容器的底层实现结构为双向链表，图 2 这种表示仅是为了方便理解各个迭代器函数的功能。

从图 2 可以看出，这些成员函数通常是成对使用的，即 begin()/end()、rbegin()/rend()、cbegin()/cend()、crbegin()/crend() 各自成对搭配使用。其中，begin()/end() 和 cbegin/cend() 的功能是类似的，同样 rbegin()/rend() 和 crbegin()/crend() 的功能是类似的。  
  
和array、vector、deque 容器的迭代器相比，list 容器迭代器最大的不同在于，其配备的迭代器类型为双向迭代器，而不再是随机访问迭代器。  
  
这意味着，假设 p1 和 p2 都是双向迭代器，则它们支持使用 ++p1、 p1++、 p1--、 p1++、 \*p1、 p1==p2 以及 p1!=p2 运算符，**但不支持以下操作（其中 i 为整数）**：

* **p1[i]：不能通过下标访问 list 容器中指定位置处的元素。**
* **p1-=i、 p1+=i、 p1+i 、p1-i：双向迭代器 p1 不支持使用 -=、+=、+、- 运算符。**
* **p1<p2、 p1>p2、 p1<=p2、 p1>=p2：双向迭代器 p1、p2 不支持使用 <、 >、 <=、 >= 比较运算符。**

下面这个程序演示了如何使用表 1 中的迭代器遍历 list 容器中的各个元素。

1. #include <iostream>
2. #include <list>
3. **u**[**sin**](http://c.biancheng.net/ref/sin.html)**g** **namespace** std;
4. int main()
5. {
6. //创建 list 容器
7. std::list<char> values{'h','t','t','p',':','/','/','c','.','b','i','a','n','c','h','e','n','g','.','n','e','t'};
8. //使用begin()/end()迭代器函数对输出list容器中的元素
9. **for** (std::list<char>::iterator it = values.begin(); it != values.end(); ++it) {
10. std::cout << \*it;
11. }
12. cout << endl;
13. //使用 rbegin()/rend()迭代器函数输出 lsit 容器中的元素
14. **for** (std::list<char>::reverse\_iterator it = values.rbegin(); it != values.rend();++it) {
15. std::cout << \*it;
16. }
17. **return** 0;
18. }

输出结果为：

http://c.biancheng.net  
ten.gnehcnaib.c//:ptth

注意，程序中比较迭代器之间的关系，用的是 != 运算符，因为它不支持 < 等运算符。另外在实际场景中，所有迭代器函数的返回值都可以传给使用 auto 关键字定义的变量，因为编译器可以自行判断出该迭代器的类型。

值得一提的是，list 容器在进行插入（insert()）、接合（splice()）等操作时，都不会造成原有的 list 迭代器失效，甚至进行删除操作，而只有指向被删除元素的迭代器失效，其他迭代器不受任何影响。  
  
举个例子：

1. #include <iostream>
2. #include <list>
3. **using** **namespace** std;
4. int main()
5. {
6. //创建 list 容器
7. std::list<char> values{'h','t','t','p',':','/','/','c','.','b','i','a','n','c','h','e','n','g','.','n','e','t'};
8. //创建 begin 和 end 迭代器
9. std::list<char>::iterator begin = values.begin();
10. std::list<char>::iterator end = values.end();
11. //头部和尾部插入字符 '1'
12. values.insert(begin, '1');
13. values.insert(end, '1');
14. **while** (begin != end)
15. {
16. std::cout << \*begin;
17. ++begin;
18. }
19. **return** 0;
20. }

运行结果为：

http://c.biancheng.net1

可以看到，在进行插入操作之后，仍使用先前创建的迭代器遍历容器，虽然程序不会出错，但由于插入位置的不同，可能会遗漏新插入的元素。