范围容器与视图

1.C++20添加了范围库,范围是一个可迭代对象的集合,支持begin()和end()迭代器的结构都是范围,大多数STL容器都是范围.

2."视图"是转换另一个基础范围的范围,视图是惰性的,只有在范围迭代时操作,视图从底层返回数据,不拥有数据,视图的运行时间复杂度时O(1).

3.视图适配器是一个对象,可接受一个范围,并返回一个视图对象.视图适配器可以用|类似管道操作符连接到其它视图适配器

`<ranges>`中定义了`std::ranges`和`std::ranges::view`命名空间,标准把`std::views`作为`std::ranges::view`的别名

将视图应用于范围,如下

```
    const std::vector nums{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    auto result = std::ranges::take_view(nums, 5);
    for (auto i: result) std::cout << i << " ";
```

输出结果为

```
1 2 3 4 5
```

`std::ranges::take_view(range,n)`:返回范围range的前n个元素的视图

视图适配器下的take_view();

```
    const std::vector nums{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    auto result = nums | std::views::take(5);
    for (auto i: result) std::cout << i << " ";
```

输出结果为

```
1 2 3 4 5
```

视图适配器位于std::ranges::views命名空间中,视图适配器从|左侧获取范围操作数,|操作符会从左往右求值

示例如下

```
    const std::vector nums{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    for (auto i: nums | std::views::take(5) | std::views::reverse) std::cout <<
i << " ";
```

输出结果为

```
5 4 3 2 1
```

filter()视图提供了一个谓词函数

```
    const std::vector nums{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    for (auto i: nums | std::views::take(5) | std::views::filter([](int x) {
return x & 1; })) {
        std::cout << i << " ";
    }
```

输出结果为

```
1 3 5
```

`std::views::filter()` 筛选出满足谓词约束的数据

transfrom()视图提供了一个转换函数

```
    const std::vector nums{1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    for (auto i: nums | std::views::take(5) | std::views::transform([](int x) {
return x * x; })) {
        std::cout << i << " ";
    }
```

输出结果为

```
1 4 9 16 25
```

iota(value, bound) 函数的作用是: 生成一个从 value 开始，到 bound 之前结束的序列。若省略了 bound，序列则为无穷大

```
    for (auto i: std::views::iota(1)|std::views::take(5)) {
        std::cout << i << " ";
    }
```

输出结果为

```
1 2 3 4 5
```

为了满足范围的基本要求，对象必须至少有两个迭代器 begin() 和 end()，其中 end() 迭代器是一 个哨兵，用于确定 Range 的端点。大多数 STL 容器都符合范围的要求，包括 string、vector、array、map 等。不过，容器适配器除外，如 stack 和 queue，因为它们没有起始迭代器和结束迭代器。 视图是一个对象，操作一个范围并返回一个修改后的范围。视图为惰性操作的，不包含自己的 数据。不保留底层数据的副本，只是根据需要返回底层元素的迭代器。

从**C++20**开始,`<algorithm>`头文件的大多数算法都会基于范围,在std::ranges命名空间中,将他们与传统算法区别开

```
    std::sort(vec.begin(), vec.end());
=>  std::ranges::sort(vec);
```

```
    std::sort(vec.begin()+3, vec.end());
=>  std::ranges::sort(std::views::drop(vec,3)) // drop(range,n)跳过range的开头n个元
素
```