

容器适配器

郭 炜 刘家瑛

北京大学



容器适配器

- ▲ 可以用某种顺序容器来实现
(让已有的顺序容器以栈/队列的方式工作)
- ▲ 1) stack: 头文件 `<stack>`
 - 栈 -- 后进先出
- ▲ 2) queue: 头文件 `<queue>`
 - 队列 -- 先进先出
- ▲ 3) priority_queue: 头文件 `<queue>`
 - 优先级队列 -- 最高优先级元素总是第一个出列



容器适配器

▲ 都有3个成员函数:

- **push**: 添加一个元素;
- **top**: 返回**栈顶部**或**队头**元素的引用
- **pop**: 删除一个元素

▲ 容器适配器上**没有迭代器**

→ STL中各种排序, 查找, 变序等算法都不适合容器适配器



stack

- stack 是**后进先出**的数据结构
- 只能插入, 删除, 访问栈顶的元素
- 可用 vector, list, deque来实现
 - 缺省情况下, 用deque实现
 - 用 vector和deque实现, 比用list实现性能好

```
template<class T, class Cont = deque<T> >  
class stack {  
  
    ...  
  
};
```



■ stack 中主要的三个成员函数:

- `void push(const T & x);`

将x压入栈顶

- `void pop();`

弹出(即删除)栈顶元素

- `T & top();`

返回栈顶元素的引用. 通过该函数, 可以读取栈顶元素的值, 也可以修改栈顶元素



queue

- 和stack 基本类似, 可以用 list和deque实现
- 缺省情况下用deque实现

```
template<class T, class Cont = deque<T> >  
class queue {  
.....  
};
```

- 同样也有push, pop, top函数
 - push发生在队尾
 - pop, top发生在队头, 先进先出



priority_queue

- ▲ 和 queue类似, 可以用vector和deque实现
- ▲ 缺省情况下用vector实现
- ▲ priority_queue 通常用堆排序技术实现, 保证最大的元素总是在最前面
 - 执行pop操作时, 删除的是最大的元素
 - 执行top操作时, 返回的是最大元素的引用
- ▲ 默认的元素比较器是 less<T>



```
#include <queue>
#include <iostream>
using namespace std;
int main() {
    priority_queue<double> priorities;
    priorities.push(3.2);
    priorities.push(9.8);
    priorities.push(5.4);
    while( !priorities.empty() ) {
        cout << priorities.top() << " ";
        priorities.pop();
    }
    return 0;
} //输出结果: 9.8 5.4 3.2
```