|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 名称和迭代器 | 迭代器失效 | 插入 | 删除 | 查找 | 场景 |
| vector | 向量、顺序、随机访问 | 插入和删除都会失效 | 尾端O（1）非尾端P：O（N-P） | 尾端O（1）非尾端P：O（N-P） | O（1） | 需要快速查找，不需要频繁插入/删除 |
| string | 字符串、顺序、随机访问 | 插入失效，删除不会 | 尾端O（1）非尾端P：O（N-P） | 尾端O（1）非尾端P：O（N-P） | O（1） | 类似vector，但是string删除元素不会释放空间（为了下次操作方便） |
| array | 数组、顺序 | 固定长度 | 无 | 无 | O（1） | 类似vector，比数组更安全（不担心越界），但是内容在栈上，且属于定长容器。 |
| deque | 双向队列、顺序、随机访问 | 插入失效。删除头和尾元素，指向被删除节点迭代器失效，而删除中间元素会使所有迭代器失效 | 首尾端：O（1）非首尾P：O（min（p， N-P）） | 首尾端：O（1）非首尾P：O（min（p， N-P）） | O（1） | 头尾增删元素很快，随机访问比vector慢一点，因为内部处理堆跳转。中间插入和删除效率交较高。因为他是list和vector综合的样子。使用较少 |
| forward\_list | 前向列表、顺序、单向 | 插入不失效，被删除节点自身失效。 | O（1） | O（1） | O（N） | 需要list的优势，但只要向前迭代 |
| list | 列表容器、顺序、双向 | 插入不失效，被删除节点自身失效。 | O（1） | O（1） | O（N） | 需要频繁插入/删除，不需要快速查找 |
| queue | 队列（容器适配器） | 不支持迭代器 | 只能尾端入：O（1） | 只能首端删除：O（1） | 不支持 | FIFO（先进先出）。底层容器可以是list或deque。 |
| priority\_queue | 优先、队列（容器适配器） | 不支持迭代器 | 只能尾端入：O（1） | 只能首端删除：O（1） | 不支持 | FIFO（先进先出）。底层容器可以是vector或deque。 |
| stack | 栈（容器适配器 | 不支持迭代器 | 只能尾端入：O（1） | 只能尾端删除：O（1） | 不支持 | FILO（先进后出）底层容器可以是list或vector或deque。 |
| set/multiset | 集合/多重集合（有序关联）双向 | 插入不失效。删除时只是被删除节点的迭代器失效，但迭代器返回void，所以需要保存删除前迭代器位置。 | O(logN) | O(logN) | O(logN) | 需要元素有序，查找/删除/插入性能一样。红黑树效率都是O(logN)。即使是几个亿的内容，最多也查几十次。 |
| map/multimap | 映射/多重映射有序关联）双向 | 同↑ | O(logN) | O(logN) | O(logN) | 需要key有序将值关联到key，查找/删除/插入性能一样 |
| unordered\_map /multimap | 无序集合（无序关联/哈希表）单向 | 插入删除失效 | 平均情况：O（1）最差情况：O（N） | 平均情况：O（1）最差情况：O（N） | 平均情况：O（1）最差情况：O（N） | 内存使用比有序的高一些，但是查找速度更快。只有哈希函数太差或者发生哈希重建才会退化为O（N）。但是一般很少发生，均摊还是O（1）。 |
| unordered\_set /multiset | 无序映射（无序关联/哈希表）单向 | 插入删除失效 | 平均情况：O（1）最差情况：O（N） | 平均情况：O（1）最差情况：O（N） | 平均情况：O（1）最差情况：O（N） | 内存使用比有序的高一下，但是查找速度更快。只有哈希函数太差或者发生哈希重建才会退化为O（N）。但是一般很少发生，均摊还是O（1）。 |
| bitset |  |  | 无 | 无 | O（1） | 需要标志集合时 |