2.2 向量-可扩充向量

cpp实现链接	https://dsa.cs.tsinghua.edu.cn/~deng/ds/src_link/vector/vector_expand.h.htm
备注	
学习日	@2020/03/11
视频起始	https://www.bilibili.com/video/av75509584?p=39
讲义	02.Vector.B.extendable_vector.pdf

▼ 02-B-1 可扩充向量

- 现有向量方案不具有可扩充性 ← 静态空间管理，容量_capacity固定，有明显不足：
  1. 上溢（overflow）：_elem[]不足以存放所有元素
  2. 下溢（underflow）：_elem[]中的元素寥寥无几

$$\lambda = \_size/\_capacity << 50\%$$

- 一般应用环境难以预测空间的需求量 → 动态调整容量

▼ 02-B-2 动态空间管理

- 即将上溢 → 适当扩大数组容量

  当向量A接近上限时，则创建一个比原来容量更大的向量B，然后将原有A的内容进行复制到B后对A进行释放！

- 扩容算法实现 expand.cpp

```
template <typename T> void Vector<T>::expand() { //向量空间不足时扩容
   if ( _size < _capacity ) return; //尚未满员时，不必扩容
   if ( _capacity < DEFAULT_CAPACITY ) _capacity = DEFAULT_CAPACITY; //不低于最小容量
   T* oldElem = _elem;  _elem = new T[_capacity <<= 1]; //容量加倍
   for ( int i = 0; i < _size; i++ )
```

```
        _elem[i] = oldElem[i]; //复制原向量内容（T为基本类型，或已重载赋值操作符'='）
    delete [] oldElem; //释放原空间
}
```

`<<= 1` 左移一位，代表整体数量直接翻倍

▼ 02-B-3 递增型扩容

不同于上一节的翻倍扩容操作，这一节则是主要使用按照固定增量`INCREMENT`扩容：

```
T* oldElem = _elem;  _elem = new T[_capacity <<= 1]; //加倍时扩容
T* oldElem = _elem;  _elem = new T[_capacity += INCREMENT]; //递增型扩容
```

每经过i次插入操作，进行一次扩容

最坏的情况：连续插入n = m*I >> 2个元素 ⇒ 因此，在1, I+1, 2I+1,...次插入都需要扩容

不计申请空间成本，扩容对应的时间成本是：0, I, 2I, 3I, (m-1)I，算术级数的总体耗时：

$$I * (m-1) * m/2 = \mathcal{O}(n^2)$$

n次操作过后，每次扩容分摊成本为O(n)

▼ 02-B-4 加倍型扩容

- 加倍式扩容分析

  ```
  T* oldElem = _elem;  _elem = new T[_capacity <<= 1]; //加倍时扩容
  T* oldElem = _elem;  _elem = new T[_capacity += INCREMENT]; //递增型扩容
  ```

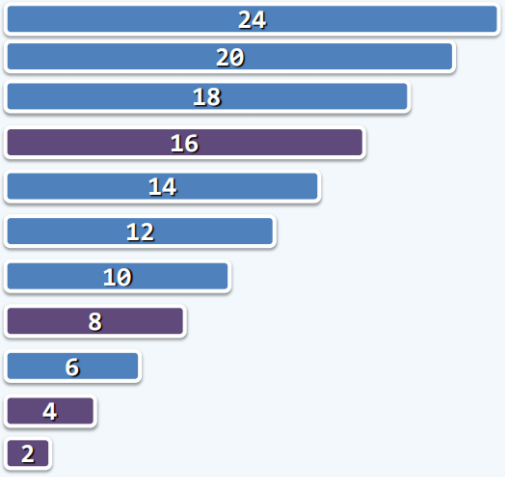
  最坏的情况：连续插入n = m*I >> 2个元素 ⇒ 因此，在1, 2 4,...次插入都需要扩容

  由之前的知识可知，复制原向量的成本为O(n)；对应地，每次扩容分摊成本为O(1)

- 算法对比

  随着数据规模增大，在运行速度的差距可能相差巨大，空间方面，递增型扩容相对利用率较好

  加倍式算法牺牲空间以换取时间操作的快速



	递增策略	倍增策略
累计 增容时间	$O(n^2)$	$O(n)$
分摊 增容时间	$O(n)$	$O(1)$
装填因子	≈ 100%	> 50%

▼ 02-B-5 分摊复杂度

平均分析vs分摊分析

- 平均分析

  将操作视为独立的事件，割裂了相关性何连贯性，往往不能准确判断算法的真实性能

- 分摊分析

  从实际可行的角度，对一系列操作整体考虑

❖ 平均复杂度或期望复杂度（average/expected complexity）

根据数据结构各种操作出现概率的分布，将对应的成本加权平均

各种可能的操作，作为独立事件分别考查

割裂了操作之间的相关性和连贯性

往往不能准确地评判数据结构和算法的真实性能

❖ 分摊复杂度（amortized complexity）

对数据结构连续地实施足够多次操作，所需总体成本分摊至单次操作

从实际可行的角度，对一系列操作做整体的考量

更加忠实地刻画了可能出现的操作序列

可以更为精准地评判数据结构和算法的真实性能