02-B-2 动态空间管理 & 02-B-3 递增式扩容

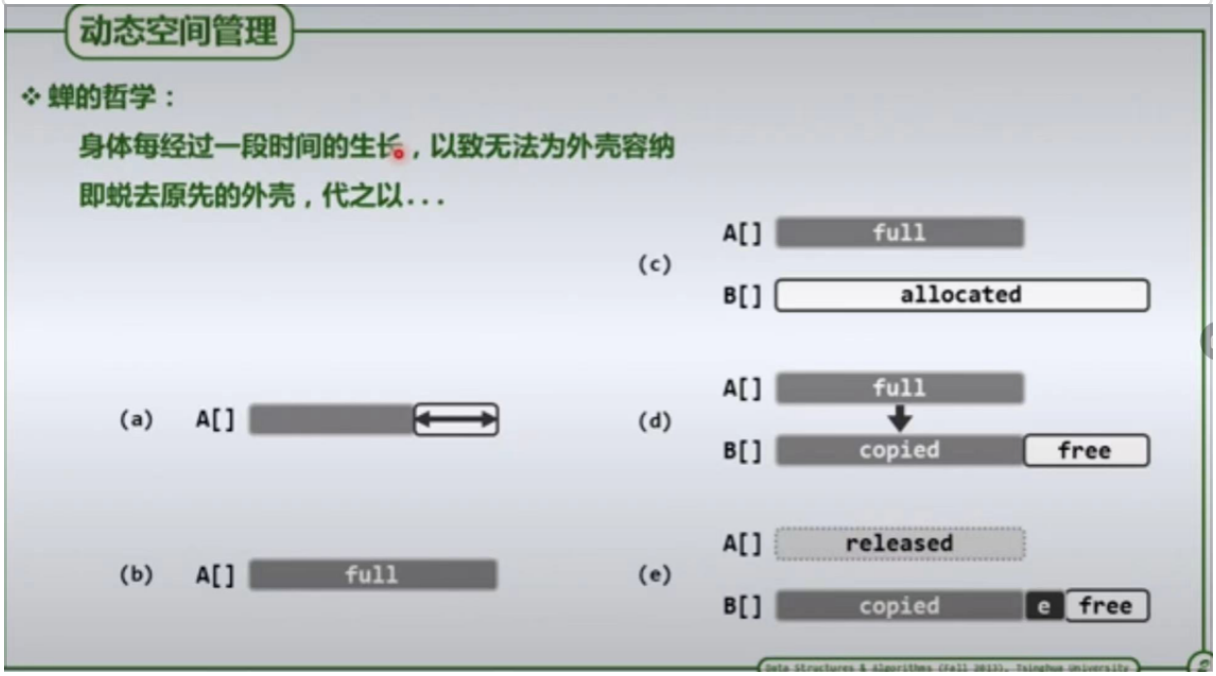
#数据结构邓神

动态空间管理

蝉的哲学：

身体每经过一段时间的增长，以至于外壳无法容纳

即蝉褪去原本的外壳，代之以...



在即将发生上溢出的时候

适当的扩大内部数组的容量

拓容算法的实现

```
template <typename T>
void Vector<T>::expand(){

    if (_size < _capacity){ // 是否即将上溢出
        return;
    }
```

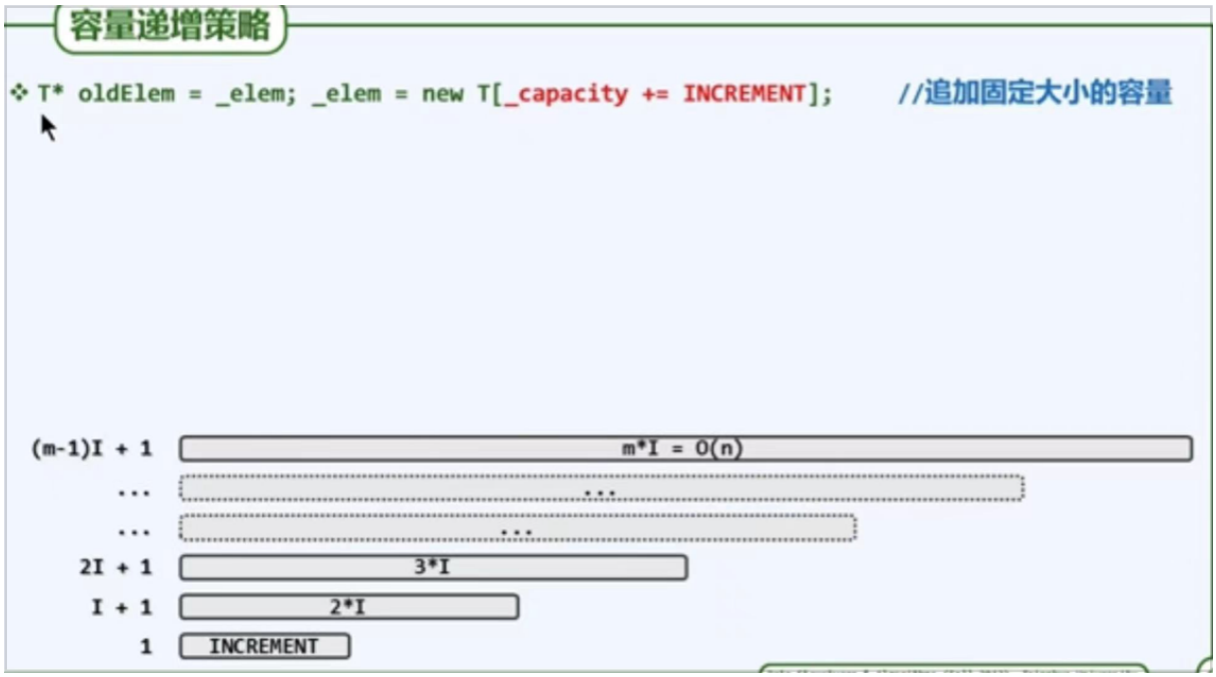
```
    _capacity = max(_capacity, DEFAULT_CAPACITY); // 不低于最低容量
    T* oldElem = _elem; _elem = new T[_capacity <<= 1]; // 容量加倍

    for (int i = 0; i < _size; ++i) {
        _elem[i] = oldElem[i];
    }
    delete [] oldElem;
}
```

得益于向量的封装，尽管扩容后的数据的物理地址有所改变，但是却不至于出现野指针

Q：为何采用容量加倍的策略呢?

其他策略是否可行?

容量递增策略



在每次容量到达上限的时候知识增加一个固定的数额

最坏情况： 在初始容量为空的一个向量： 连续插入 n = m*I >> 2个元素

于是在 1 , I+1 , 2*I+1 , 3*I + 1 次插入的时候都需要拓展容量

即使不计算申请空间的操作，每次扩容过程复制原向量的时间成本为

0,I,2I,...,(m−1)I // 算数级数

总体耗时 = I∗(m−1)∗m/2 总成本为O(n^2) 每次扩容的分摊成本为 O(n)