java ArrayQueue 增长因子为什么是1.5

小圆滚滚

1 增长因子公式

公式:

$c\sum_{i=0}^{n} 2^i = c(2^{n+1} - 1) < c2^{n+1}$

也就是说，之前分配的内存空间不可能被使用。这样对于缓存并不友好。最好把增长因子设为$1 < k < 2$，例如 Folly 采用 1.5，RapidJSON 也是跟随采用 1.5.

```
GenericValue& PushBack(GenericValue& value, Allocator& allocator) {
    RAPIDJSON_ASSERT(IsArray());
    if (data_.a.size >= data_.a.capacity)
        Reserve(data_.a.capacity == 0 ? kDefaultArrayCapacity : (data_.a.
            capacity + (data_.a.capacity + 1) / 2), allocator);
    data_.a.elements[data_.a.size++].RawAssign(value);
    return *this;
}
```

```
private void grow(int minCapacity) {
        // overflow-conscious code
        int oldCapacity = elementData.length;   //将扩充前的elementData大小
            给oldCapacity
        int newCapacity = oldCapacity + (oldCapacity >> 1);//newCapacity就
            是1.5倍的oldCapacity
        if (newCapacity - minCapacity < 0)//这句话就是适应于elementData就空数组
            的时候，length=0，那么oldCapacity=0，newCapacity=0，所以这个判断成立，在这
            里就是真正的初始化elementData的大小了，就是为10.前面的工作都是准备工作。
            newCapacity = minCapacity;
        if (newCapacity - MAX_ARRAY_SIZE > 0)//如果newCapacity超过了最大的容量
            限制，就调用hugeCapacity，也就是将能给的最大值给newCapacity
            newCapacity = hugeCapacity(minCapacity);
        // minCapacity is usually close to size, so this is a win:
    //新的容量大小已经确定好了，就copy数组，改变容量大小咯。
        elementData = Arrays.copyOf(elementData, newCapacity);
    }
```

比较内存分配的情况:

```
k = 2, c = 4
```

```
0123
    01234567
            012345789ABCDEF
                        0123456789ABCDEF0123456789ABCDEF
                                                    012345...

k = 1.5, c = 4
0123
    012345
          012345678
                    0123456789ABCD
                                0123456789ABCDEF0123
0123456789ABCDEF0123456789ABCD
                              0123456789ABCDEF0123456789ABCDEF...
```

可以看到，k = 1.5 在几次扩展之后，可以重用之前的内存空间。