

[JavaGuide](#) / [docs](#) / [java](#) / [collection](#) / **ArrayList-Grow.md** Snailclimb Update ArrayList-Grow.md

b853a3c on 29 Dec 2019

1 contributor

Raw	Blame	History
-----	-------	---------



358 lines (276 sloc) | 13.4 KB

一 先从 ArrayList 的构造函数说起

ArrayList有三种方式来初始化，构造方法源码如下：

[illegible]

```

/**
 *构造包含指定collection元素的列表，这些元素利用该集合的迭代器按顺序返回
 *如果指定的集合为null，throws NullPointerException。
 */
public ArrayList(Collection<? extends E> c) {
    elementData = c.toArray();
    if ((size = elementData.length) != 0) {
        // c.toArray might (incorrectly) not return Object[] (see 6260652)
        if (elementData.getClass() != Object[].class)
            elementData = Arrays.copyOf(elementData, size, Object[].class);
    } else {
        // replace with empty array.
        this.elementData = EMPTY_ELEMENTDATA;
    }
}

```

细心的同学一定会发现：**以无参数构造方法创建 ArrayList 时，实际上初始化赋值的是一个空数组。当真正对数组进行添加元素操作时，才真正分配容量。即向数组中添加第一个元素时，数组容量扩为10。**下面在我们分析 ArrayList 扩容时会讲到这一点内容！

二 一步一步分析 ArrayList 扩容机制

这里以无参构造函数创建的 ArrayList 为例分析

1. 先来看 add 方法

```

/**
 * 将指定的元素追加到此列表的末尾。
 */
public boolean add(E e) {
    //添加元素之前，先调用ensureCapacityInternal方法
    ensureCapacityInternal(size + 1); // Increments modCount!!
    //这里看到ArrayList添加元素的实质就相当于为数组赋值
    elementData[size++] = e;
    return true;
}

```

2. 再来看看 ensureCapacityInternal() 方法

可以看到 add 方法 首先调用了 ensureCapacityInternal(size + 1)

```

//得到最小扩容量
private void ensureCapacityInternal(int minCapacity) {
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        // 获取默认的容量和传入参数的较大值
        minCapacity = Math.max(DEFAULT_CAPACITY, minCapacity);
    }
}

```

```
    }  
  
    ensureExplicitCapacity(minCapacity);  
}
```

当要 add 进第1个元素时，minCapacity为1，在Math.max()方法比较后，minCapacity为10。

3. ensureExplicitCapacity() 方法

如果调用 ensureCapacityInternal() 方法就一定会进过（执行）这个方法，下面我们来研究一下这个方法的源码！

```
//判断是否需要扩容  
private void ensureExplicitCapacity(int minCapacity) {  
    modCount++;  
  
    // overflow-conscious code  
    if (minCapacity - elementData.length > 0)  
        //调用grow方法进行扩容，调用此方法代表已经开始扩容了  
        grow(minCapacity);  
}
```

我们来仔细分析一下：

- 当我们要 add 进第1个元素到 ArrayList 时，elementData.length 为0（因为还是一个空的 list），因为执行了 ensureCapacityInternal() 方法，所以 minCapacity 此时为10。此时，minCapacity - elementData.length > 0 成立，所以会进入 grow(minCapacity) 方法。
- 当add第2个元素时，minCapacity 为2，此时elementData.length(容量)在添加第一个元素后扩容成 10 了。此时，minCapacity - elementData.length > 0 不成立，所以不会进入（执行）grow(minCapacity) 方法。
- 添加第3、4...到第10个元素时，依然不会执行grow方法，数组容量都为10。

直到添加第11个元素，minCapacity(为11)比elementData.length（为10）要大。进入grow方法进行扩容。

4. grow() 方法

```
/**  
 * 要分配的最大数组大小  
 */  
private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;  
  
/**  
 * ArrayList扩容的核心方法。  
 */
```

```

private void grow(int minCapacity) {
    // oldCapacity为旧容量, newCapacity为新容量
    int oldCapacity = elementData.length;
    //将oldCapacity 右移一位, 其效果相当于oldCapacity /2,
    //我们知道位运算的速度远远快于整除运算, 整句运算式的结果就是将新容量更新为旧容
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    //然后检查新容量是否大于最小需要容量, 若还是小于最小需要容量, 那么就把最小需要
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;
    // 如果新容量大于 MAX_ARRAY_SIZE, 进入(执行) `hugeCapacity()` 方法来比较 minC
    //如果minCapacity大于最大容量, 则新容量则为`Integer.MAX_VALUE`, 否则, 新容量
    if (newCapacity - MAX_ARRAY_SIZE > 0)
        newCapacity = hugeCapacity(minCapacity);
    // minCapacity is usually close to size, so this is a win:
    elementData = Arrays.copyOf(elementData, newCapacity);
}

```

int newCapacity = oldCapacity + (oldCapacity >> 1),所以 ArrayList 每次扩容之后容量都会变为原来的 1.5 倍 ! (JDK1.6版本以后) JDK1.6版本时, 扩容之后容量为 1.5 倍 +1 ! 详情请参考源码

">>" (移位运算符) : >>1 右移一位相当于除2, 右移n位相当于除以 2 的 n 次方。这里 oldCapacity 明显右移了1位所以相当于oldCapacity /2。对于大数据的2进制运算,位移运算符比那些普通运算符的运算要快很多,因为程序仅仅移动一下而已,不去计算,这样提高了效率,节省了资源

我们再来通过例子探究一下 grow() 方法 :

- 当add第1个元素时, oldCapacity 为0, 经比较后第一个if判断成立, newCapacity = minCapacity(为10)。但是第二个if判断不会成立, 即newCapacity 不比 MAX_ARRAY_SIZE大, 则不会进入 hugeCapacity 方法。数组容量为10, add方法中 return true,size增为1。
- 当add第11个元素进入grow方法时, newCapacity为15, 比minCapacity (为11) 大, 第一个if判断不成立。新容量没有大于数组最大size, 不会进入hugeCapacity方法。数组容量扩为15, add方法中return true,size增为11。
- 以此类推……

这里补充一点比较重要, 但是容易被忽视掉的知识点 :

- java 中的 length 属性是针对数组说的,比如说你声明了一个数组,想知道这个数组的长度则用到了 length 这个属性。
- java 中的 length() 方法是针对字符串说的,如果想看这个字符串的长度则用到 length() 这个方法。
- java 中的 size() 方法是针对泛型集合说的,如果想看这个泛型有多少个元素,就调用此方法来查看!

5. hugeCapacity() 方法。

从上面 grow() 方法源码我们知道：如果新容量大于 MAX_ARRAY_SIZE,进入(执行) hugeCapacity() 方法来比较 minCapacity 和 MAX_ARRAY_SIZE，如果minCapacity大于最大容量，则新容量则为 Integer.MAX_VALUE，否则，新容量大小则为 MAX_ARRAY_SIZE 即为 Integer.MAX_VALUE - 8。

```
private static int hugeCapacity(int minCapacity) {
    if (minCapacity < 0) // overflow
        throw new OutOfMemoryError();
    //对minCapacity和MAX_ARRAY_SIZE进行比较
    //若minCapacity大，将Integer.MAX_VALUE作为新数组的大小
    //若MAX_ARRAY_SIZE大，将MAX_ARRAY_SIZE作为新数组的大小
    //MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;
    return (minCapacity > MAX_ARRAY_SIZE) ?
        Integer.MAX_VALUE :
        MAX_ARRAY_SIZE;
}
```

三 System.arraycopy() 和 Arrays.copyOf() 方法

阅读源码的话，我们就会发现 ArrayList 中大量调用了这两个方法。比如：我们上面讲的扩容操作以及 add(int index, E element)、toArray() 等方法中都用到了该方法！

3.1 System.arraycopy() 方法

```
/**
 * 在此列表中的指定位置插入指定的元素。
 *先调用 rangeCheckForAdd 对index进行界限检查；然后调用 ensureCapacityInternal
 *再将从index开始之后的所有成员后移一个位置；将element插入index位置；最后size加1
 */
public void add(int index, E element) {
    rangeCheckForAdd(index);

    ensureCapacityInternal(size + 1); // Increments modCount!!
    //arraycopy()方法实现数组自己复制自己
    //elementData:源数组;index:源数组中的起始位置;elementData: 目标数组; index
    System.arraycopy(elementData, index, elementData, index + 1, size - index)
    elementData[index] = element;
    size++;
}
```

我们写一个简单的方法测试以下：

```
public class ArraycopyTest {
```

```

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] a = new int[10];
        a[0] = 0;
        a[1] = 1;
        a[2] = 2;
        a[3] = 3;
        System.arraycopy(a, 2, a, 3, 3);
        a[2]=99;
        for (int i = 0; i < a.length; i++) {
            System.out.println(a[i]);
        }
    }
}

```

结果：

```
0 1 99 2 3 0 0 0 0 0
```

3.2 Arrays.copyOf() 方法

```

/**
 * 以正确的顺序返回一个包含此列表中所有元素的数组（从第一个到最后一个元素）；返回的数
 */
public Object[] toArray() {
    //elementData: 要复制的数组; size: 要复制的长度
    return Arrays.copyOf(elementData, size);
}

```

个人觉得使用 `Arrays.copyOf()` 方法主要是为了给原有数组扩容，测试代码如下：

```

public class ArrayscopyOfTest {

    public static void main(String[] args) {
        int[] a = new int[3];
        a[0] = 0;
        a[1] = 1;
        a[2] = 2;
        int[] b = Arrays.copyOf(a, 10);
        System.out.println("b.length"+b.length);
    }
}

```

结果：

3.3 两者联系和区别

联系：

看两者源代码可以发现 `copyOf()` 内部实际调用了 `System.arraycopy()` 方法

区别：

`arraycopy()` 需要目标数组，将原数组拷贝到你自己定义的数组里或者原数组，而且可以选择拷贝的起点和长度以及放入新数组中的位置 `copyOf()` 是系统自动在内部新建一个数组，并返回该数组。

四 `ensureCapacity` 方法

`ArrayList` 源码中有一个 `ensureCapacity` 方法不知道大家注意到没有，这个方法 `ArrayList` 内部没有被调用过，所以很显然是提供给用户调用的，那么这个方法有什么作用呢？

```
/**
 * 如有必要，增加此 ArrayList 实例的容量，以确保它至少可以容纳由minimum capacity参数
 *
 * @param minCapacity 所需的最小容量
 */
public void ensureCapacity(int minCapacity) {
    int minExpand = (elementData != DEFAULTCAPACITY_EMPTY_ELEMENTDATA)
        // any size if not default element table
        ? 0
        // larger than default for default empty table. It's already
        // supposed to be at default size.
        : DEFAULT_CAPACITY;

    if (minCapacity > minExpand) {
        ensureExplicitCapacity(minCapacity);
    }
}
```

最好在 add 大量元素之前用 `ensureCapacity` 方法，以减少增量重新分配的次数

我们通过下面的代码实际测试以下这个方法的效果：

```
public class EnsureCapacityTest {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<Object>();
        final int N = 10000000;
        long startTime = System.currentTimeMillis();
        for (int i = 0; i < N; i++) {
```

```

        list.add(i);
    }
    long endTime = System.currentTimeMillis();
    System.out.println("使用ensureCapacity方法前: " + (endTime - startTi

    }
}

```

运行结果：

使用ensureCapacity方法前: 2158

```

public class EnsureCapacityTest {
    public static void main(String[] args) {
        ArrayList<Object> list = new ArrayList<Object>();
        final int N = 10000000;
        list = new ArrayList<Object>();
        long startTime1 = System.currentTimeMillis();
        list.ensureCapacity(N);
        for (int i = 0; i < N; i++) {
            list.add(i);
        }
        long endTime1 = System.currentTimeMillis();
        System.out.println("使用ensureCapacity方法后: " + (endTime1 - startTime1));
    }
}

```

运行结果：

使用ensureCapacity方法前: 1773

通过运行结果，我们可以看出向 ArrayList 添加大量元素之前最好先使用 ensureCapacity 方法，以减少增量重新分配的次数。