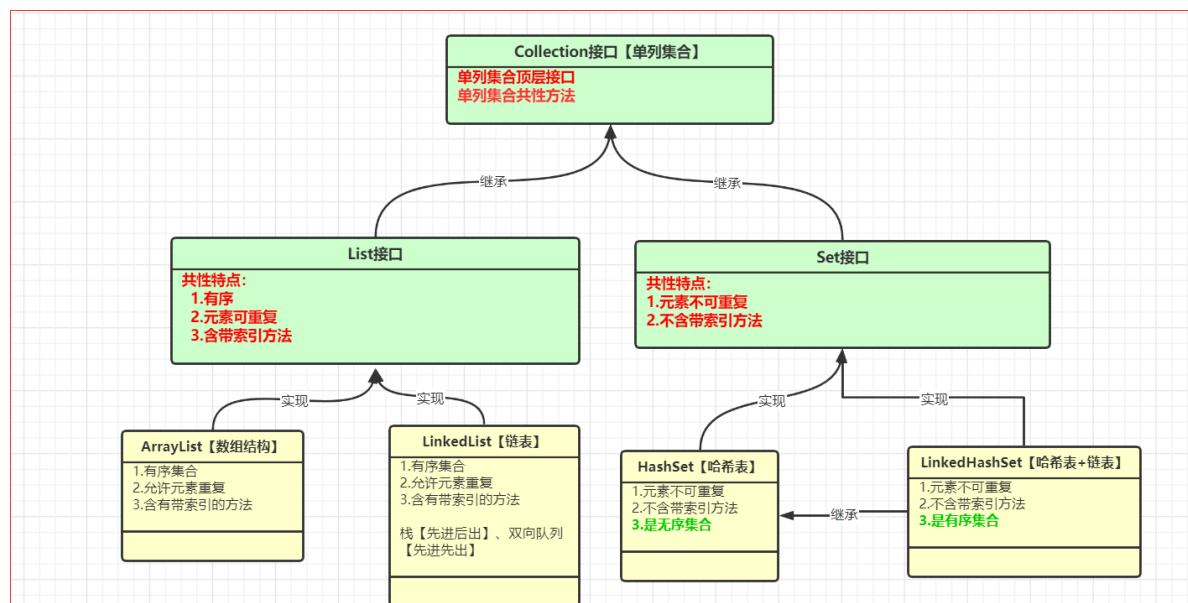


第一章 LinkedList源码分析

目标:

- 理解LinkedList的底层数据结构
- 深入源码掌握LinkedList查询慢，新增快的原因

一、LinkedList的简介

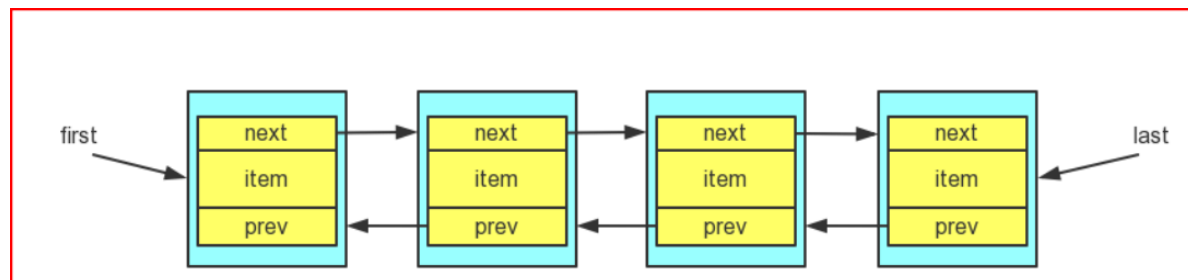


`List` 接口的链接列表实现。实现所有可选的列表操作，并且允许所有元素（包括 `null`）。除了实现 `List` 接口外，`LinkedList` 类还为在列表的开头及结尾 `get`、`remove` 和 `insert` 元素提供了统一的命名方法。这些操作允许将链接列表用作堆栈、队列或双端队列。

特点：

- 有序性：存入和取出的顺序是一致的
- 元素可以重复：
- 含有带索引的方法
- 独有特点：数据结构是链表，可以作为栈、队列或者双端队列！

`LinkedList`是一个双向的链表结构，双向链表的长相，如下图！



二、LinkedList原理分析

2.1 LinkedList的数据结构

LinkedList是一个双向链表!

底层数据结构的源码

```
1 public class LinkedList<E>{
2     transient int size = 0;
3     //双向链表的头结点
4     transient Node<E> first;
5     //双向链表的最后一个节点
6     transient Node<E> last;
7     //节点类【内部类】
8     private static class Node<E> {
9         E item;//数据元素
10        Node<E> next;//下一个节点
11        Node<E> prev;//上一个节点
12        //节点的构造方法
13        Node(Node<E> prev, E element, Node<E> next) {
14            this.item = element;
15            this.next = next;
16            this.prev = prev;
17        }
18    }
19    //...
20 }
```

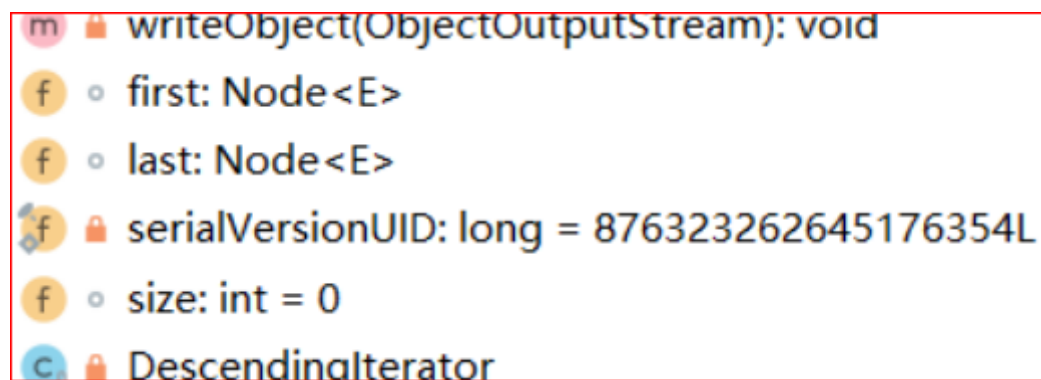
LinkedList是双向链表，在代码中是一个Node类。内部并没有数组的结构。双向链表肯定存在一个头节点和一个尾部节点。node节点类，是以内部类的形式存在于LinkedList中的。Node类都有两个成员变量：

- prev : 当前节点上一个节点，头节点的上一个节点是null
- next : 当前节点下一个节点，尾结点的下一个节点是null

链表数据结构的特点：查询慢，增删快！

- 链表数据结构基本构成，是一个node类
- 每个node类中，有上一个节点【prev】和下一个节点【next】
- 链表一定存在至少两个节点，first和last节点
- 如果LinkedList没有数据，irst和last都是为null

2.2 LinkedList默认容量&最大容量



```
m writeObject(ObjectOutputStream): void
f first: Node<E>
f last: Node<E>
f serialVersionUID: long = 876323262645176354L
f size: int = 0
c Descendiniterator
```

没有默认容量，也没有最大容量

2.3 LinkedList扩容机制

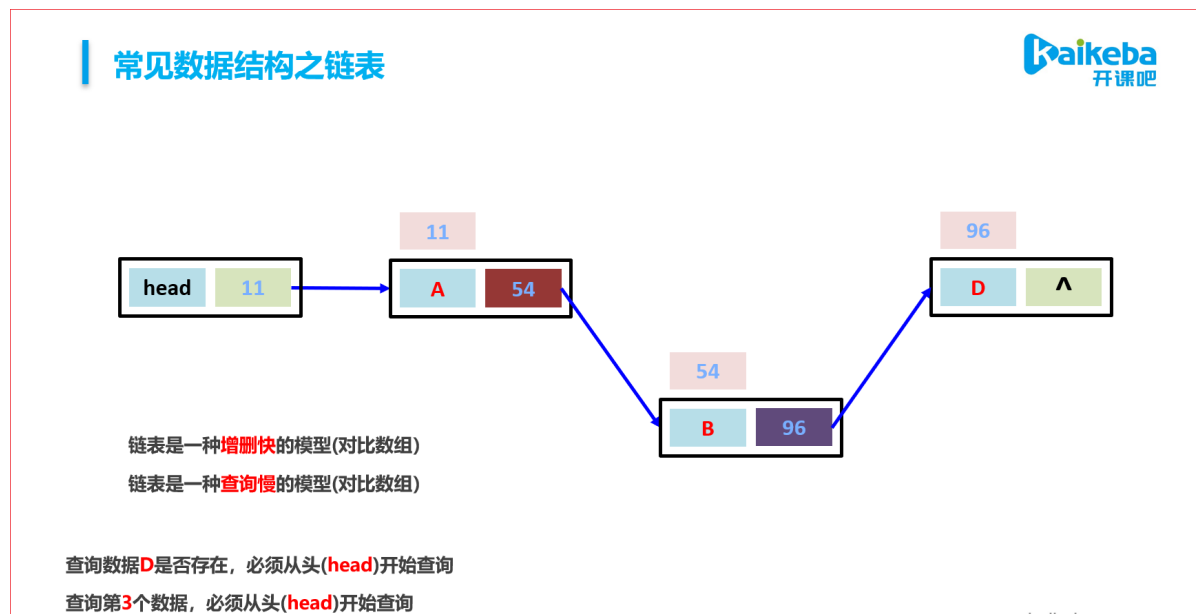
无需扩容机制，只要你的内存足够大，可以无限制扩容下去。前提是不考虑查询的效率。

2.4 为什么LinkedList查询慢，增删快？

LinkedList的数据结构的特点，链表的数据结构就是这样的特点！

- 链表是一种**查询慢**的结构【相对于数组来说】
- 链表是一种**增删快**的结构【相对于数组来说】

上动画



2.5 LinkedList源码剖析-为什么增删快？

新增add

```
1 //想LinkedList添加一个元素
2 public boolean add(E e) {
3     //连接到链表的末尾
4     linkLast(e);
5     return true;
6 }
7 //连接到最后一个节点上去
8 void linkLast(E e) {
9     //将全局末尾节点赋值给l
10    final Node<E> l = last;
11    //创建一个新节点 : (上一个节点, 当前插入元素, null)
12    final Node<E> newNode = new Node<>(l, e, null);
13    //将当前节点作为末尾节点
14    last = newNode;
15    //判断l节点是否为null
16    if (l == null)
17        //既是尾结点也是头节点
18        first = newNode;
19    else
20        //之前的末尾节点, 下一个节点时末尾节点!
21        l.next = newNode;
22    size++; //当前集合的元素数量+1
23    modCount++; //操作集合数+1。modCount属性是修改技术器
```

```

24 }
25 //-----
26 //向链表中部添加
27 //参数1, 添加的索引位置, 添加元素
28 public void add(int index, E element) {
29     //检查索引位是否符合要求
30     checkPositionIndex(index);
31     //判断当前所有是否是存储元素个数
32     if (index == size) //true, 最后一个元素
33         linkLast(element);
34     else
35         //连接到指定节点的后面【链表中部插入】
36         linkBefore(element, node(index));
37 }
38 //根据索引查询链表中节点!
39 Node<E> node(int index) {
40     // 判断索引是否小于 已经存储元素个数的1/2
41     if (index < (size >> 1)) { //二分法查找 : 提高查找节点效率
42         Node<E> x = first;
43         for (int i = 0; i < index; i++)
44             x = x.next;
45         return x;
46     } else {
47         Node<E> x = last;
48         for (int i = size - 1; i > index; i--)
49             x = x.prev;
50         return x;
51     }
52 }
53 //将当前元素添加到指定节点之前
54 void linkBefore(E e, Node<E> succ) {
55     // 取出当前节点的前一个节点
56     final Node<E> pred = succ.prev;
57     //创建当前元素的节点 : 上一个节点, 当前元素, 下一个节点
58     final Node<E> newNode = new Node<>(pred, e, succ);
59     //为指定节点上一个节点重新值
60     succ.prev = newNode;
61     //判断当前节点的上一个节点是否为null
62     if (pred == null)
63         first = newNode; //当前节点作为头部节点
64     else
65         pred.next = newNode; //将新插入节点作为上一个节点的下个节点
66     size++; //新增元素+1
67     modCount++; //操作次数+1
68 }

```

remove删除指定索引元素

```

1 //删除指定索引位置元素
2 public E remove(int index) {
3     //检查元素索引
4     checkElementIndex(index);
5     //删除元素节点,
6     //node(index) 根据索引查到要删除的节点
7     //unlink()删除节点
8     return unlink(node(index));
9 }

```

```

10 //根据索引查询链表中节点!
11 Node<E> node(int index) {
12     // 判断索引是否小于 已经存储元素个数的1/2
13     if (index < (size >> 1)) { //二分法查找 : 提高查找节点效率
14         Node<E> x = first;
15         for (int i = 0; i < index; i++)
16             x = x.next;
17         return x;
18     } else {
19         Node<E> x = last;
20         for (int i = size - 1; i > index; i--)
21             x = x.prev;
22         return x;
23     }
24 }
25 //删除一个指定节点
26 E unlink(Node<E> x) {
27     //获取当前节点中的元素
28     final E element = x.item;
29     //获取当前节点的上一个节点
30     final Node<E> next = x.next;
31     //获取当前节点的下一个节点
32     final Node<E> prev = x.prev;
33     //判断上一个节点是否为null
34     if (prev == null) {
35         //如果为null, 说明当前节点为头部节点
36         first = next;
37     } else {
38         //上一个节点, 的下一个节点改为下下节点
39         prev.next = next;
40         //将当前节点的上一个节点置空
41         x.prev = null;
42     }
43     //判断下一个节点是否为null
44     if (next == null) {
45         //如果为null, 说明当前节点为尾部节点
46         last = prev;
47     } else {
48         //下一个节点的上节点, 改为上上节点
49         next.prev = prev;
50         //当前节点的上节点置空
51         x.next = null;
52     }
53     //删除当前节点内的元素
54     x.item = null;
55     size--; //集合中的元素个数-1
56     modCount++; //当前集合操作数+1。modCount计数器, 记录当前集合操作次数
57     return element; //返回删除的元素
58 }

```

2.6 LinkedList源码剖析-为什么查询慢?

查询快和慢是一个相对概念! 相对于数组来说

```

1 //根据索引查询一个元素
2 public E get(int index) {
3     //检查索引是否存在

```

```

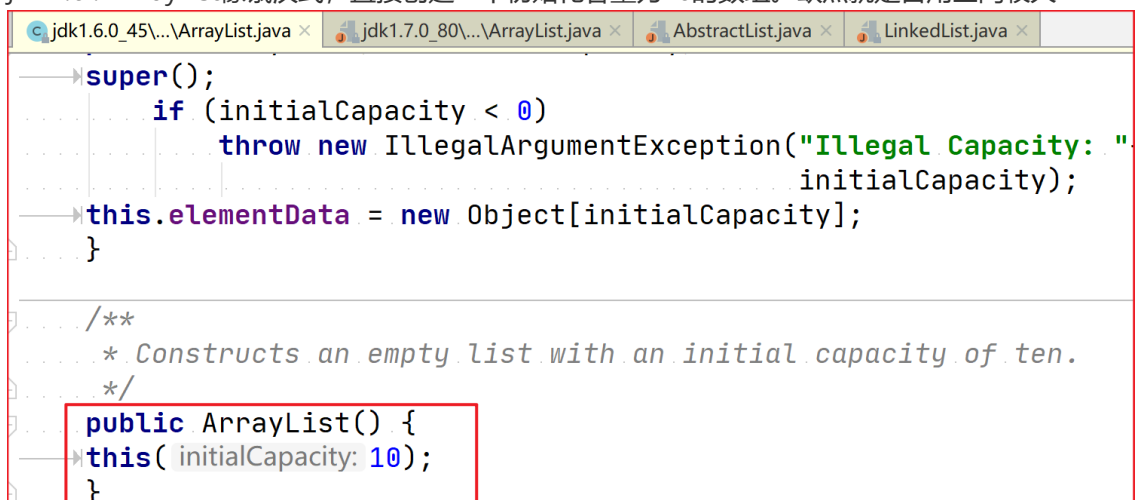
4     checkElementIndex(index);
5     // node(index)获取索引对应节点，获取节点中的数据item
6     return node(index).item;
7 }
8 //根据索引获取对应节点对象
9 Node<E> node(int index) {
10    //二分法查找索引对应的元素
11    if (index < (size >> 1)) {
12        Node<E> x = first;
13        //前半部分查找【遍历节点】
14        for (int i = 0; i < index; i++)
15            x = x.next;
16        return x;
17    } else {
18        Node<E> x = last;
19        //后半部分查找【遍历】
20        for (int i = size - 1; i > index; i--)
21            x = x.prev;
22        return x;
23    }
24 }
25 //查看ArrayList里的数组获取元素的方式
26 public E get(int index) {
27     rangeCheck(index); //检查范围
28     return elementData(index); //获取元素
29 }
30 E elementData(int index) {
31     return (E) elementData[index]; //一次性操作
32 }

```

第二章 经典大厂面试题

1、ArrayList的JDK1.8之前与之后的实现区别？

- JDK1.6：ArrayList像饿汉式，直接创建一个初始化容量为10的数组。缺点就是占用空间较大



```

→ super();
    if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal Capacity: ".
            initialCapacity);
→ this.elementData = new Object[initialCapacity];
    }

    /**
     * Constructs an empty list with an initial capacity of ten.
     */
    public ArrayList() {
→ this(initialCapacity: 10);
    }

```

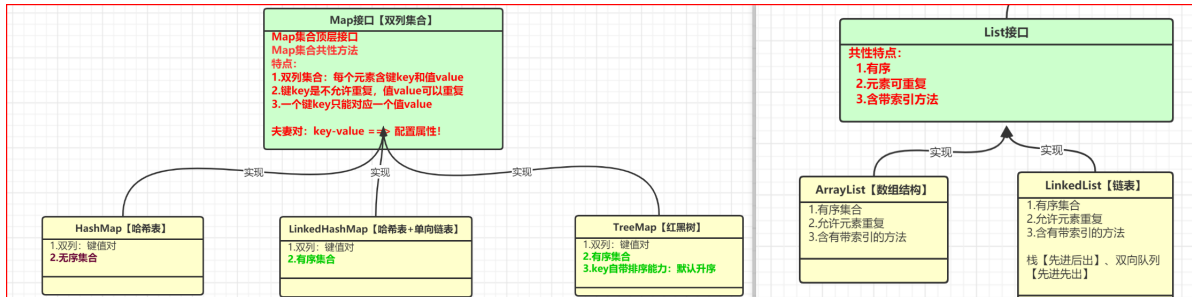
- JDK1.7 & JDK1.8：ArrayList像懒汉式，一开始创建一个长度为0的数组，当添加第一个元素时再创建一个初始容量为10的数组

```

1 private static final Object[] EMPTY_ELEMENTDATA = {};
2
3 public ArrayList() {
4     super();
5     this.elementData = EMPTY_ELEMENTDATA;
6 }

```

2、List 和 Map 区别？



Map集合

- 双列集合：一次存一对
- key是不允许重复的，value可以重复
- 一个key只能对应一个值value
- Map集合三兄弟：HashMap【无序集合】、LinkedHashMap【有序集合】、TreeMap【有序集合，自带排序能力】

List集合

- 单列集合：一次存一个
- 有序集合
- 元素可以重复
- 带索引
- List集合主要有两个实现类：ArrayList和LinkedList

3、Array 和 ArrayList 有何区别？什么时候更适合用 Array？

区别：

- Array可以容纳基本类型和对象，而ArrayList只能容纳对象【底层是一个对象数组】。
- Array指定大小的固定不变，而ArrayList大小是动态的，可自动扩容。
- Array没有ArrayList方法多。

尽管ArrayList明显是更好的选择，但也有些时候Array比较好用，比如下面的三种情况。

- 1、如果列表的大小已经指定，大部分情况下是存储和遍历它们
- 2、基本数据类型使用Array更合适。

4、ArrayList 与 LinkedList 区别？

ArrayList

- 优点：ArrayList是实现了基于动态数组的数据结构，因为地址连续，一旦数据存储好了，查询操作效率会比较高（在内存里是连着放的）。查询快，增删相对慢
- 缺点：因为地址连续，ArrayList要移动数据，所以插入和删除操作效率比较低。

LinkedList

- 优点：LinkedList 基于链表的数据结构，地址是任意的，所以在开辟内存空间的时候不需要等一个连续的地址。对于新增和删除操作 add 和 remove，LinkedList 比较占优势。LinkedList 适用于要头尾操作或插入指定位置的场景。
- 缺点：因为 LinkedList 要移动指针，所以查询操作性能比较低。查询慢，增删快

适用场景分析：

- 当需要对数据进行对随机访问的情况下，选用 ArrayList。
- 当需要对数据进行多次增加删除修改时，采用 LinkedList。
- 当然，绝大数业务的场景下，使用 ArrayList 就够了。主要是，注意：最好避免 ArrayList 扩容，以及非顺序的插入。

ArrayList 是如何扩容的？

参考第一章原理分析中的扩容原理讲解

- 如果通过无参构造的话，初始数组容量为 0，当真正对数组进行添加时，才真正分配容量。每次按照 1.5 倍（位运算）的比率通过 `copyOf` 的方式扩容。

重点是 1.5 倍扩容，这是和 HashMap 2 倍扩容不同的地方。

5、ArrayList 集合加入 10万条数据，应该怎么提高效率？

ArrayList 的默认初始容量为 10，要插入大量数据的时候需要不断扩容，而扩容是非常影响性能的。因此，现在明确了 10 万条数据了，我们可以直接在初始化的时候就设置 ArrayList 的容量！

这样就可以提高效率了~

6、ArrayList 与 Vector 区别？

ArrayList 和 Vector 都是用数组实现的，主要有这么三个区别：

- 1、Vector 是多线程安全的，线程安全就是说多线程访问同一代码，不会产生不确定的结果，而 ArrayList 不是。这个可以从源码中看出，Vector 类中的方法很多有 `synchronized` 进行修饰，这样就导致了 Vector 在效率上无法与 ArrayList 相比。

Vector 是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用。

- 2、两个都是采用的线性连续空间存储元素，但是当空间不足的时候，两个类的增加方式是不同。
- 3、Vector 可以设置增长因子，而 ArrayList 不可以，ArrayList 集合没有增长因子。

适用场景分析：

- 1、Vector 是线程同步的，所以它也是线程安全的，而 ArrayList 是线程无需同步的，是不安全的。如果不考虑到线程的安全因素，一般用 ArrayList 效率比较高。

实际场景下，如果需要多线程访问安全的数组，使用 `CopyOnWriteArrayList`。