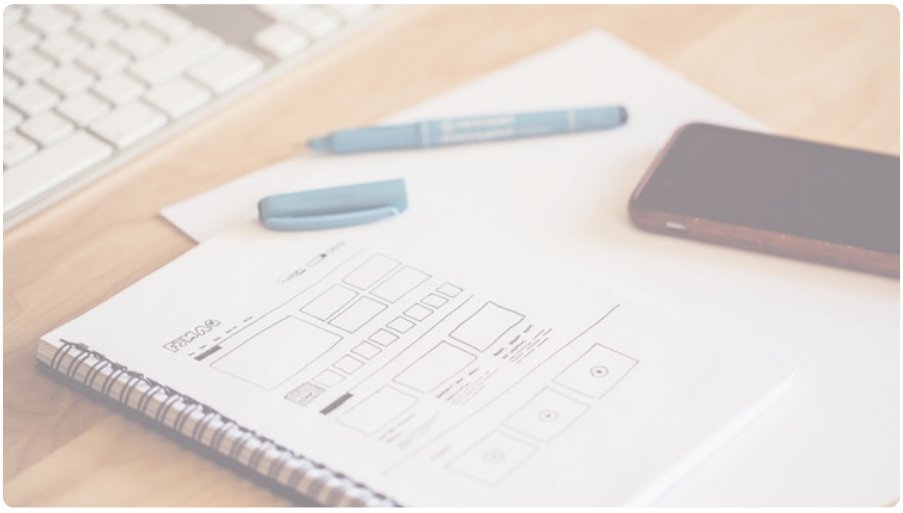
目录

06 LinkedList 源码解析

更新时间：2019-11-26 09:44:59



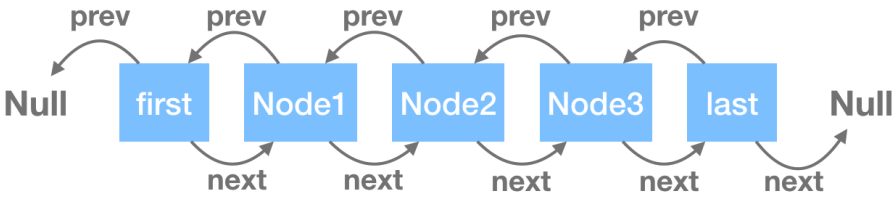
> 智慧，不是死的默念，而是生的沉思。
>
> ——斯宾诺莎

引导语

LinkedList 适用于集合元素先入先出和先入后出的场景，在队列源码中被频繁使用，面试也经常问到，本小节让我们通过源码来加深对 LinkedList 的了解。

1 整体架构

LinkedList 底层数据结构是一个双向链表，整体结构如下图所示：



上图代表了一个双向链表结构，链表中的每个节点都可以向前或者向后追溯，我们有几个概念如下：

- 链表每个节点我们叫做 Node，Node 有 prev 属性，代表前一个节点的位置，next 属性，代表后一个节点的位置；
- first 是双向链表的头节点，它的前一个节点是 null。
- last 是双向链表的尾节点，它的后一个节点是 null；
- 当链表中没有数据时，first 和 last 是同一个节点，前后指向都是 null；
- 因为是个双向链表，只要机器内存足够强大，是没有大小限制的。

链表中的元素叫做 Node，我们看下 Node 的组成部分：

```
    Node<E> next; // 指向的下一个节点
    Node<E> prev; // 指向的前一个节点

    // 初始化参数顺序分别是：前一个节点、本身节点值、后一个节点
    Node(Node<E> prev, E element, Node<E> next) {
        this.item = element;
        this.next = next;
        this.prev = prev;
    }
}
```

2 源码解析

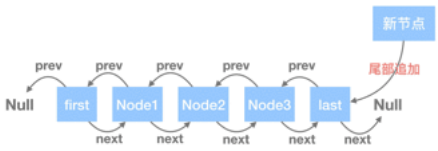
2.1 追加（新增）

追加节点时，我们可以选择追加到链表头部，还是追加到链表尾部，add 方法默认是从尾部开始追加，addFirst 方法是从头部开始追加，我们分别来看下两种不同的追加方式：

从尾部追加（add）

```
// 从尾部开始追加节点
void linkLast(E e) {
    // 把尾节点数据暂存
    final Node<E> l = last;
    // 新建新的节点，初始化入参含义：
    // l 是新节点的前一个节点，当前值是尾节点值
    // e 表示当前新增节点，当前新增节点后一个节点是 null
    final Node<E> newNode = new Node<>(l, e, null);
    // 新建节点追加到尾部
    last = newNode;
    //如果链表为空（l 是尾节点，尾节点为空，链表即空），头部和尾部是同一个节点，都是新建的节
    if (l == null)
        first = newNode;![图片描述](//img.mukewang.com/5d5fc69600013e4803600240.gif)
    //否则把前尾节点的下一个节点，指向当前尾节点。
    else
        l.next = newNode;
    //大小和版本更改
    size++;
    modCount++;
}
```


从源码上来看，尾部追加节点比较简单，只需要简单地把指向位置修改下即可，我们做个动图来描述下整个过程：



从头部追加（addFirst）

目录

```
    // 头节点赋值给临时变量
    final Node<E> f = first;
    // 新建节点，前一个节点指向null，e 是新建节点，f 是新建节点的下一个节点，目前值是头节点的
    final Node<E> newNode = new Node<>(null, e, f);
    // 新建节点成为头节点
    first = newNode;
    // 头节点为空，就是链表为空，头尾节点是一个节点
    if (f == null)
        last = newNode;
    //上一个头节点的前一个节点指向当前节点
    else
        f.prev = newNode;
    size++;
    modCount++;
}
```

头部追加节点和尾部追加节点非常类似，只是前者是移动头节点的 prev 指向，后者是移动尾节点的 next 指向。

2.2 节点删除

节点删除的方式和追加类似，我们可以选择从头部删除，也可以选择从尾部删除，删除操作会把节点的值，前后指向节点都置为 null，帮助 GC 进行回收。

从头部删除


```
//从头删除节点 f 是链表头节点
private E unlinkFirst(Node<E> f) {
    // 拿出头节点的值，作为方法的返回值
    final E element = f.item;
    // 拿出头节点的下一个节点
    final Node<E> next = f.next;
    //帮助 GC 回收头节点
    f.item = null;
    f.next = null;
    // 头节点的下一个节点成为头节点
    first = next;
    //如果 next 为空，表明链表为空
    if (next == null)
        last = null;
    //链表不为空，头节点的前一个节点指向 null
    else
        next.prev = null;
    //修改链表大小和版本
    size--;
    modCount++;
    return element;
}
```

从尾部删除节点代码也是类似的，就不贴了。

从源码中我们可以了解到，链表结构的节点新增、删除都非常简单，仅仅把前后节点的指向修改下就好了，所以 LinkedList 新增和删除速度很快。

2.3 节点查询

目录

```
// 根据链表索引位置查询节点
Node<E> node(int index) {
    // 如果 index 处于队列的前半部分，从头开始找，size >> 1 是 size 除以 2 的意思。
    if (index < (size >> 1)) {
        Node<E> x = first;
        // 直到 for 循环到 index 的前一个 node 停止
        for (int i = 0; i < index; i++)
            x = x.next;
        return x;
    } else {// 如果 index 处于队列的后半部分，从尾开始找
        Node<E> x = last;
        // 直到 for 循环到 index 的后一个 node 停止
        for (int i = size - 1; i > index; i--)
            x = x.prev;
        return x;
    }
}
```

从源码中我们可以发现，LinkedList 并没有采用从头循环到尾的做法，而是采取了简单二分法，首先看看 index 是在链表的前半部分，还是后半部分。如果是前半部分，就从头开始寻找，反之亦然。通过这种方式，使循环的次数至少降低了一半，提高了查找的性能，这种思想值得我们借鉴。

2.4 方法对比

LinkedList 实现了 Queue 接口，在新增、删除、查询等方面增加了很多新的方法，这些方法在平时特别容易混淆，在链表为空的情况下，返回值也不太一样，我们列一个表格，方便大家记录：

方法含义	返回异常	返回特殊值	底层实现
新增	add(e)	offer(e)	底层实现相同
删除	remove()	poll(e)	链表为空时，remove 会抛出异常，poll 返回 null。
查找	element()	peek()	链表为空时，element 会抛出异常，peek 返回 null。

PS：Queue 接口注释建议 add 方法操作失败时抛出异常，但 LinkedList 实现的 add 方法一直返回 true。

LinkedList 也实现了 Deque 接口，对新增、删除和查找都提供从头开始，还是从尾开始两种方向的方法，比如 remove 方法，Deque 提供了 removeFirst 和 removeLast 两种方向的使用方式，但当链表为空时的表现都和 remove 方法一样，都会抛出异常。

2.5 迭代器

因为 LinkedList 要实现双向的迭代访问，所以我们使用 Iterator 接口肯定不行了，因为 Iterator 只支持从头到尾的访问。Java 新增了一个迭代接口，叫做：ListIterator，这个接口提供了向前和向后的迭代方法，如下所示：

迭代顺序	方法
从尾到头迭代方法	hasPrevious、previous、previousIndex
从头到尾迭代方法	hasNext、next、nextIndex

目录

```
// 双向迭代器
private class ListItr implements ListIterator<E> {
    private Node<E> lastReturned;//上一次执行 next() 或者 previos() 方法时的节点位置
    private Node<E> next;//下一个节点
    private int nextIndex;//下一个节点的位置
    //expectedModCount：期望版本号；modCount：目前最新版本号
    private int expectedModCount = modCount;
    .....
}
```

我们先来看下从头到尾方向的迭代：

```
// 判断还有没有下一个元素
public boolean hasNext() {
    return nextIndex < size;// 下一个节点的索引小于链表的大小，就有
}

// 取下一个元素
public E next() {
    //检查期望版本号有无发生变化
    checkForComodification();
    if (!hasNext())//再次检查
        throw new NoSuchElementException();
    // next 是当前节点，在上一次执行 next() 方法时被赋值的。
    // 第一次执行时，是在初始化迭代器的时候，next 被赋值的
    lastReturned = next;
    // next 是下一个节点了，为下次迭代做准备
    next = next.next;
    nextIndex++;
    return lastReturned.item;
}
```


上述源码的思路就是直接取当前节点的下一个节点，而从尾到头迭代稍微复杂一点，如下：

```
// 如果上次节点索引位置大于 0，就还有节点可以迭代
public boolean hasPrevious() {
    return nextIndex > 0;
}
// 取前一个节点
public E previous() {
    checkForComodification();
    if (!hasPrevious())
        throw new NoSuchElementException();
    // next 为空场景：1:说明是第一次迭代，取尾节点(last);2:上一次操作把尾节点删除掉了
    // next 不为空场景：说明已经发生过迭代了，直接取前一个节点即可(next.prev)
    lastReturned = next = (next == null) ? last : next.prev;
    // 索引位置变化
    nextIndex--;
    return lastReturned.item;
}
```

这里复杂点体现在需要判断 next 不为空和为空的场景，代码注释中有详细的描述。

迭代器删除

LinkedList 在删除元素时，也推荐通过迭代器进行删除，删除过程如下：

目录

```
    // lastReturned 是本次迭代需要删除的值，分以下空和非空两种情况：
    // lastReturned 为空，说明调用者没有主动执行过 next() 或者 previos()，直接报错
    // lastReturned 不为空，是在上次执行 next() 或者 previos()方法时赋的值
    if (lastReturned == null)
        throw new IllegalStateException();
    Node<E> lastNext = lastReturned.next;
    //删除当前节点
    unlink(lastReturned);
    // next == lastReturned 的场景分析：从尾到头递归顺序，并且是第一次迭代，并且要删除最后一
    // 这种情况下，previous() 方法里面设置了 lastReturned = next = last,所以 next 和 lastReturne
    if (next == lastReturned)
        // 这时候 lastReturned 是尾节点，lastNext 是 null，所以 next 也是 null，这样在 previous()
        next = lastNext;
    else
        nextIndex--;
    lastReturned = null;
    expectedModCount++;
}
```

总结

LinkedList 适用于要求有顺序、并且会按照顺序进行迭代的场景，主要是依赖于底层的链表结构，在面试中的频率还是蛮高的，相信理清楚上面的源码后，应对面试应该没有问题。

← 05 ArrayList 源码解析和设计思路 07 List 源码会问哪些面试题 →

精选留言 12

欢迎在这里发表留言，作者筛选后可公开显示

rq_conquer

有一个问题，LinkedList的add我看源码只返回true，老师说的add失败返回false是从哪里来的，是我看漏了什么吗?

0 回复 2019-10-16

文贺 回复 **rq_conquer**

感谢提醒，你是对的，已订正了，谢谢。

回复 2019-10-17 19:14:46

威先森

问题：调用next()方法时，next变量为什么有值？ 原因：当调用list.iterator()生成迭代器的时候，都会调用LinkeList继承的AbstractSequentialList的iterator()方法，iterator()方法则会调用AbstractSequentialList父类的listIterator()方法，listIterator()方法会调用listIterator(final int index)，由于LinkedList重写了listIterator(final int index)，所以最后调用链又回到LinkedList中了，LinkedList中的listIterator方法默认调用有参构造new了ListItr这个类，在构造方法中给next和nextIndex都赋值了，老师分析的应该是对的吧，有问题希望老师指正出来

目录

文贺 回复 **威无森**

是的，没问题，给你点赞。

回复 2019-10-14 12:45:35

风舞炫动

老师可否对2.4方法对比表格中的方法给出demo，有点理解不了，空链表指的不是直接new出来的吗？add也不会抛出异常呀，还有element()、poll()、peek()这些方法也是没有参数的呀（问的low的话别介意，实在是有点看不懂，(#^.^#)）

0 回复 2019-10-10

文贺 回复 **风舞炫动**

感谢提醒，问题很好，我写的不好，我重新修改了下，谢谢。

回复 2019-10-11 19:43:36

文贺 回复 **文贺**

麻烦同学再看看，能否看懂了。

回复 2019-10-11 19:44:26

风舞炫动 回复 **文贺**

明 白 啦 ， 谢 谢 老 师

回复 2019-10-12 11:00:19

汤圆

请老师更快点啊请老师更快点啊

1 回复 2019-09-14

文贺 回复 **汤圆**

努力努力努力，加油加油加油。

回复 2019-10-11 19:13:29

大胖晴

看完了这个LinkedList源码解析之后，真的是处处都是数据结构和算法,辛亏链表看过了，要不然感觉看着一脸懵逼,看来数据结构还得带着看了;

1 回复 2019-09-10

文贺 回复 **大胖晴**

哈哈，集合中的确是有很多数据结构，但对数据结构大体了解的话，再看源码应该都还好吧，看源码可以先扫过，看看这坨代码大体是干啥的，了解下整体架构，然后再慢慢深入，实在还是看不懂，建议多多 debug，debug 很奇妙，不懂的代码，debug 一下全都明白。

回复 2019-09-10 19:15:56

慕神9346227

表格里是抛出异常还是跑出异常

0 回复 2019-09-04

目录

回复 2019-09-05 17:42:07

慕神4112825

这么长时间了为啥还不更新啊

0 回复 2019-09-01

初一 回复 **慕神4112825**

同学你好，本专栏是每周二、周四更新的呦^ ^ 感谢支持

回复 2019-09-02 10:29:42

Elylic

非常好的专栏！！第一次订阅！感觉超棒！

3 回复 2019-08-30

文贺 回复 **Elylic**

感谢支持，一起进步。

回复 2019-09-03 16:02:04

慕虎6508060

从前往后的迭代中，第一次使用Next，next为空的话怎么lastReturned也为空，怎么继续迭代呀

1 回复 2019-08-29

文贺 回复 **慕虎6508060**

第一次迭代的时候 next 不为空哈，LinkedList 的 listIterator 方法有无参的，也有有参数的，无参数的底层默认传0的，有参数的方法如下，index 代表第一次迭代的位置。 public ListIterator listIterator(int index) { checkPositionIndex(index); return new ListItr(index); } ListItr 初始化的时候，会给 next 赋值的，next 肯定不为空，但 lastReturned 是空的

回复 2019-08-30 20:29:22

慕虎6508060

想问一下linkedLis的双向迭代，从前往后迭代的过程中next和lastReturned的初始情况是空吗

0 回复 2019-08-29

文贺 回复 **慕虎6508060**

lastReturned为空，但 next 不为空，原因见上一条回复

回复 2019-08-30 20:29:55

窝兔与草

请问老师，更新速度是怎样的呢?

初一 回复 莴兔与草

目录

点击展开剩余评论

千学不如一看，千看不如一练