**Java LinkedList**

链表（Linked list）是一种常见的基础数据结构，是一种线性表，但是并不会按线性的顺序存储数据，而是在每一个节点里存到下一个节点的地址。

链表可分为单向链表和双向链表。

一个单向链表包含两个值: 当前节点的值和一个指向下一个节点的链接。

一个双向链表有三个整数值: 数值、向后的节点链接、向前的节点链接。

Java LinkedList（链表） 类似于 ArrayList，是一种常用的数据容器。

与 ArrayList 相比，LinkedList 的增加和删除的操作效率更高，而查找和修改的操作效率较低。

以下情况使用 ArrayList :

频繁访问列表中的某一个元素。

只需要在列表末尾进行添加和删除元素操作。

以下情况使用 LinkedList :

你需要通过循环迭代来访问列表中的某些元素。

需要频繁的在列表开头、中间、末尾等位置进行添加和删除元素操作。

LinkedList 继承了 AbstractSequentialList 类。

LinkedList 实现了 Queue 接口，可作为队列使用。

LinkedList 实现了 List 接口，可进行列表的相关操作。

LinkedList 实现了 Deque 接口，可作为队列使用。

LinkedList 实现了 Cloneable 接口，可实现克隆。

LinkedList 实现了 java.io.Serializable 接口，即可支持序列化，能通过序列化去传输。

LinkedList 类位于 java.util 包中，使用前需要引入它，语法格式如下：

// 引入 LinkedList 类

import java.util.LinkedList;

LinkedList<E> list = new LinkedList<E>(); // 普通创建方法

或者

LinkedList<E> list = new LinkedList(Collection<? extends E> c); // 使用集合创建链表

创建一个简单的链表实例：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

System.out.println(sites);

}

}

以上实例，执行输出结果为：

[Google, Runoob, Taobao, Weibo]

更多的情况下我们使用 ArrayList 访问列表中的随机元素更加高效，但以下几种情况 LinkedList 提供了更高效的方法。

在列表开头添加元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

// 使用 addFirst() 在头部添加元素

sites.addFirst("Wiki");

System.out.println(sites);

}

}

以上实例，执行输出结果为：

[Wiki, Google, Runoob, Taobao]

在列表结尾添加元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

// 使用 addLast() 在尾部添加元素

sites.addLast("Wiki");

System.out.println(sites);

}

}

以上实例，执行输出结果为：

[Google, Runoob, Taobao, Wiki]

在列表开头移除元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

// 使用 removeFirst() 移除头部元素

sites.removeFirst();

System.out.println(sites);

}

}

以上实例，执行输出结果为：

[Runoob, Taobao, Weibo]

在列表结尾移除元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

// 使用 removeLast() 移除尾部元素

sites.removeLast();

System.out.println(sites);

}

}

以上实例，执行输出结果为：

[Google, Runoob, Taobao]

获取列表开头的元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

// 使用 getFirst() 获取头部元素

System.out.println(sites.getFirst());

}

}

以上实例，执行输出结果为：

Google

获取列表结尾的元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

// 使用 getLast() 获取尾部元素

System.out.println(sites.getLast());

}

}

以上实例，执行输出结果为：

Weibo

迭代元素

我们可以使用 for 配合 size() 方法来迭代列表中的元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

for (int size = sites.size(), i = 0; i < size; i++) {

System.out.println(sites.get(i));

}

}

}

size() 方法用于计算链表的大小。

以上实例，执行输出结果为：

Google

Runoob

Taobao

Weibo

也可以使用 for-each 来迭代元素：

实例

// 引入 LinkedList 类

import java.util.LinkedList;

public class RunoobTest {

public static void main(String[] args) {

LinkedList<String> sites = new LinkedList<String>();

sites.add("Google");

sites.add("Runoob");

sites.add("Taobao");

sites.add("Weibo");

for (String i : sites) {

System.out.println(i);

}

}

}

以上实例，执行输出结果为：

Google

Runoob

Taobao

Weibo

常用方法

方法 描述

public boolean add(E e) 链表末尾添加元素，返回是否成功，成功为 true，失败为 false。

public void add(int index, E element) 向指定位置插入元素。

public boolean addAll(Collection c) 将一个集合的所有元素添加到链表后面，返回是否成功，成功为 true，失败为 false。

public boolean addAll(int index, Collection c) 将一个集合的所有元素添加到链表的指定位置后面，返回是否成功，成功为 true，失败为 false。

public void addFirst(E e) 元素添加到头部。

public void addLast(E e) 元素添加到尾部。

public boolean offer(E e) 向链表末尾添加元素，返回是否成功，成功为 true，失败为 false。

public boolean offerFirst(E e) 头部插入元素，返回是否成功，成功为 true，失败为 false。

public boolean offerLast(E e) 尾部插入元素，返回是否成功，成功为 true，失败为 false。

public void clear() 清空链表。

public E removeFirst() 删除并返回第一个元素。

public E removeLast() 删除并返回最后一个元素。

public boolean remove(Object o) 删除某一元素，返回是否成功，成功为 true，失败为 false。

public E remove(int index) 删除指定位置的元素。

public E poll() 删除并返回第一个元素。

public E remove() 删除并返回第一个元素。

public boolean contains(Object o) 判断是否含有某一元素。

public E get(int index) 返回指定位置的元素。

public E getFirst() 返回第一个元素。

public E getLast() 返回最后一个元素。

public int indexOf(Object o) 查找指定元素从前往后第一次出现的索引。

public int lastIndexOf(Object o) 查找指定元素最后一次出现的索引。

public E peek() 返回第一个元素。

public E element() 返回第一个元素。

public E peekFirst() 返回头部元素。

public E peekLast() 返回尾部元素。

public E set(int index, E element) 设置指定位置的元素。

public Object clone() 克隆该列表。

public Iterator descendingIterator() 返回倒序迭代器。

public int size() 返回链表元素个数。

public ListIterator listIterator(int index) 返回从指定位置开始到末尾的迭代器。

public Object[] toArray() 返回一个由链表元素组成的数组。

public T[] toArray(T[] a) 返回一个由链表元素转换类型而成的数组。