链滴

java【链表实现】

作者：haxLook
原文链接：https://ld246.com/article/1619773555606
来源网站：链滴

一般我们在学习集合之前，都会去了解了解数据结构的相关知识，最常见的莫过于链表，在List集合面，linkedList这个子类就是使用的链表的数据结果进行实现的，

链表的优势：新增数据，删除数据，会很容易

动态数组优势：检索快

下面的有一个关于链表的数据结构，是我大学学习期间写的一个测试，便于我们加深对链表结构的掌。

```
package ITaljavaT3;

import org.junit.jupiter.api.Test;

/*
 * 链表的的增加
 */
interface Ilink<E>{
    public abstract void add(E date);//进行元素的添加
    public abstract int size();//进行长度的的获取
    public abstract boolean isEmpty();//进行判断是否为空集合
    public abstract Object[] toArray();//根据元素集合对应给数组获取数据
    public abstract E getdate (int index);//根据指定出的索引获取数据
    public abstract void setdate(int index,E date);//修改指定处索引处的值
    public abstract boolean judge(E date);//判断链表中数据是否存在
    public abstract void removedate(E date);//进行数据的删除
    public abstract void clean();//清空集合数据
}
class impLink<E> implements Ilink<E>{

    /**
     * 节点内部类
     * @author 米饭饭一族
     *
     */
    private class Node{
        /**
         * 内部类成员变量
         */
        private E date;

        private Node next;

        public Node(E date) {
            this.date = date;
        }
        //第一次调用；this=impLink.root
        //第二次调用；this=impLink.root.next
        //第三次调用；this=impLink.root.next.next
        public void addnode(Node newnode) {//保存新的NODE数据
            if(this.next == null) {//当前节点的下一个节点为null，
                this.next = newnode;//保存当前节点
            }else {
                /*
```

```
         * 在第一次循环中这个地方this是 impLink.root
         * 第二次的时候就是implink.root.next.addnode(newnode);
         * 第三次的时候就是implink.root.next.next.addnode(newnode);
         */
        this.next.addnode(newnode);//将这个节点（newnode）添加到上一个数据的节点中，
    }
}
//第一次调用 this=implink.root;根节点
//第二次调用 this=implink.root.next;根节点
//第一三调用 this=implink.root.next.next;根节点
public void returndate() {
    //外部类的中当前对象
    impLink.this.returndate[impLink.this.foot ++]=this.date;
    if(this.next!=null) {
        this.next.returndate();//递归中出口
    }
}
//根据索引获取相应的数据
public E getNode(int index) {
    if(impLink.this.foot++==index) {
        return this.date;
    }
    return this.next.getNode(index);
}

//根据索引修改相应的数据
public void setNode(int index,E date ) {
    if(impLink.this.foot++ ==index) {
        this.date=date;
    }else {
        this.next.setNode(index,date);
    }
}
//判断date数据是否存在
public boolean judgeNode(E date) {
    if(date.equals(this.date)) {
        return true;
    }else {
        if(this.next==null) {
            return false;
        }else {
            return this.next.judgeNode(date);
        }
    }
}
//进行数据的删除(把节点的下一个节点赋值给节点的上一个节点)
public void removeNode(Node before,E date) {
    if(date.equals(this.date)) {
        before.next=this.next;
    }else {
        this.next.removeNode(this, date);
    }
}
```

```
}
//-----------------------linK类的成员变量-----------------
private Node root;//保存根元素
private int count;
private int foot;
private Object[] returndate;
//-----------------------linK类的成员方法-----------------
//Node node=this.root;
@Override
public void add(E date) {
    if(date==null) {//保存数据为空
        return;//方法调用直接返回
    }
    //date是一个数据，数据本身是没，要想实现关联性就必须使用Node来实现引用之间的 传递
    Node newnode = new Node(date);//创建一个新的节点
    if(this.root==null) {//根节点为空
        this.root = newnode;//将newnode设定为根节点
    }else {//节点存在
        this.root.addnode(newnode);//将新的节点保存在合适的位置
        //this.root为跟节点，跟节点调用方法
    }
    this.count++;

}
//进行长度的的获取
public int size() {
    return this.count;
}
//进行判断是否为空集合
public boolean isEmpty() {
    return this.count==0;
    //return this.root==null;
}
//数据的返回
public Object[] toArray() {
    //先判断是否为空
    if(this.isEmpty()) {
        return null;
    }
    this.foot=0;
    this.returndate = new Object[this.count];
    //用node类去获取数据
    this.root.returndate();
    return this.returndate;
}
//根据索引获取指定的数据
public E getdate(int index) {
    if(index>=count) {
        return null;
    }
    this.foot=0;
    return this.root.getNode(index);
}
//修改指定处的数据
```

```
    public void setdate(int index,E date) {
        if(index >= count) {
            return;
        }else {
            this.foot=0;
            this.root.setNode(index, date);
        }
    }
    //判断数据是否存在
    public boolean judge(E date) {
        if(date == null) {
            return false;
        }
        return this.root.judgeNode(date);
    }
    //进行数据的删除
    public void removedate(E date) {
        if(this.judge(date)) {//判断date是否存在
            if(date.equals(this.root.date)) {
                this.root = this.root.next;
            }else {
                this.root.removeNode(this.root, date);
            }
            this.count--;
        }
    }
    //进行集合的清理
    public void clean() {
        this.root = null;//根节点为空
        this.count = 0;//索引清除
    }

}
public class datascope {

    /**
     * 链表结构的测试操作
     */
    @Test
    public void testdataScope() {

            Ilink<String> link = new impLink<String>();

            System.out.println(String.format("开始 【长度】%s,【值】%s,【是否为空】%s", link.size(
,link.toString(),link.isEmpty()));
            link.add("第一个");
            link.add("第二个");
            link.add("第三个");
            link.add("第四个");
            link.add("第五个");
            System.out.println(String.format("中间 【长度】%s,【值】%s,【是否为空】%s", link.size(),
ink.toString(),link.isEmpty()));
            //删除一个节点
            link.removedate("第一个");
```

```
            link.setdate(3, "修改第三个数据成功");
            //进行数据清空
            System.out.println(String.format("修改 【长度】%s,【值】%s,【是否为空】%s", link.size(),
ink.toString(),link.isEmpty()));

            link.clean();

            System.out.println(String.format("结束 【长度】%s,【值】%s,【是否为空】%s", link.size(),
ink.toString(),link.isEmpty()));

            Object[] result = link.toArray();
            if(result!=null) {
                for(Object temp:result) {
                    System.out.println(temp);
                }
            }
            System.out.println("----------------指定数据的获取（索引）---------------");
            if(!link.isEmpty()) {
                System.out.println(link.getdate(1));
                System.out.println(link.getdate(2));
                System.out.println(link.getdate(3));
                System.out.println(link.getdate(19));
            }
            System.out.println("----------------判断数据是否存在）---------------");
            if(!link.isEmpty()) {
                System.out.println(link.judge("第一个"));
                System.out.println(link.judge("aaa"));
            }

        }

}
```

将上述的链表进行扩展，实现一个购物车，写一个购物车的实体类小demo

```
package ITaljavaT4;
/*
 * 链表的的增加
 */
interface Ilink<E>{
    public abstract void add(E date);//进行元素的添加
    public abstract int size();//进行长度的的获取
    public abstract boolean isEmpty();//进行判断是否为空集合
    public abstract Object[] toArray();//根据元素集合对应给数组获取数据
    public abstract E getdate (int index);//根据指定出的索引获取数据
    public abstract void setdate(int index,E date);//修改指定处索引处的值
    public abstract boolean judge(E date);//判断链表中数据是否存在
    public abstract void removedate(E date);//进行数据的删除
    public abstract void clean();//清空集合数据
}
class impLink<E> implements Ilink<E>{
    private class Node{
        private E date;
```

```
private Node next;
public Node(E date) {
    this.date=date;
}
//第一次调用；this=impLink.root
//第二次调用；this=impLink.root.next
//第三次调用；this=impLink.root.next.next
public void addnode(Node newnode) {//保存新的NODE数据
    if(this.next==null) {//当前节点的下一个节点为null，
        this.next=newnode;//保存当前节点
    }else {
        /*
         * 在第一次循环中这个地方this是 impLink.root
         * 第二次的时候就是implink.root.next.addnode(newnode);
         * 第三次的时候就是implink.root.next.next.addnode(newnode);
         *
         */
        this.next.addnode(newnode);//将这个节点（newnode）添加到上一个数据的节点中，
    }
}
//第一次调用 this=implink.root;根节点
//第二次调用 this=implink.root.next;根节点
//第一三调用 this=implink.root.next.next;根节点
public void returndate() {
    //外部类的中当前对象
    impLink.this.returndate[impLink.this.foot ++]=this.date;
    if(this.next!=null) {
        this.next.returndate();//递归中出口
    }
}
//根据索引获取相应的数据
public E getNode(int index) {
    if(impLink.this.foot+++==index) {
        return this.date;
    }
    return this.next.getNode(index);
}

//根据索引修改相应的数据
public void setNode(int index,E date ) {
    if(impLink.this.foot++ ==index) {
        this.date=date;
    }else {
        this.next.setNode(index,date);
    }
}
//判断date数据是否存在
public boolean judgeNode(E date) {
    if(date.equals(this.date)) {
        return true;
    }else {
        if(this.next==null) {
            return false;
        }else {
```

```
                return this.next.judgeNode(date);
            }
        }
    }
    //进行数据的删除(把节点的下一个节点赋值给节点的上一个节点)
    public void removeNode(Node before,E date) {
        if(date.equals(this.date)) {
            before.next=this.next;
        }else {
            this.next.removeNode(this, date);
        }
    }

}
//------------------------linK类的成员变量-----------------
private Node root;//保存根元素
private int count;
private int foot;
private Object[] returndate;
//------------------------linK类的成员方法-----------------
//Node node=this.root;
@Override
public void add(E date) {
    if(date==null) {//保存数据为空
        return;//方法调用直接返回
    }
    //date是一个数据，数据本身是没，要想实现关联性就必须使用Node来实现引用之间的 传递
    Node newnode=new Node(date);//创建一个新的节点
    if(this.root==null) {//根节点为空
        this.root=newnode;//将newnode设定为根节点
    }else {//节点存在
        this.root.addnode(newnode);//将新的节点保存在合适的位置
        //this.root为跟节点，跟节点调用方法
    }
    this.count++;

}
//进行长度的的获取
public int size() {
    return this.count;
}
//进行判断是否为空集合
public boolean isEmpty() {
    return this.count==0;
    //return this.root==null;
}
//数据的返回
public Object[] toArray() {
    //先判断是否为空
    if(this.isEmpty()) {
        return null;
    }
    this.foot=0;
    this.returndate=new Object[this.count];
```

```
        //用node类去获取数据
        this.root.returndate();
        return this.returndate;
    }
    //根据索引获取指定的数据
    public E getdate(int index) {
        if(index>=count) {
            return null;
        }
        this.foot=0;
        return this.root.getNode(index);
    }
    //修改指定处的数据
    public void setdate(int index,E date) {
        if(index>=count) {
            return;
        }else {
            this.foot=0;
            this.root.setNode(index, date);
        }
    }
    //判断数据是否存在
    public boolean judge(E date) {
        if(date==null) {
            return false;
        }
        return this.root.judgeNode(date);
    }
    //进行数据的删除
    public void removedate(E date) {
        if(this.judge(date)) {//判断date是否存在
            if(date.equals(this.root.date)) {
                this.root=this.root.next;
            }else {
                this.root.removeNode(this.root, date);
            }
            this.count--;
        }
    }
    //进行集合的清理
    public void clean() {
        this.root=null;//根节点为空
        this.count=0;//索引清除
    }

}
/*
 * 写一个购物车的实现
 * 1.收银台
 *  包括一个计算总价的方法，一个获取商品商品数量的方法
 * 2.购物车的接口标准
 * 包括一个 添加商品，包括删除商品，包括获取所有的商品
 * 3.商品类
 * 包括一个商品的价格，名字
```

```
 */
//商品标准
interface Igoods{
    public String getName();
    public double getprice();
}
//购物车的标准
interface IShop{//购物车的接口
    public void addshop(Igoods goods);
    public void delete(Igoods goods);
    public Object[] getall();
}
class Shop implements IShop{
    private Igoods goods;
    private Ilink<Igoods> shopgoogs =new impLink<Igoods>();
//    public Shop(Igoods goods) {
//        this.goods=goods;
//    }
    @Override
    //添加商品
    public void addshop(Igoods goods) {
        // TODO Auto-generated method stub
        this.shopgoogs.add(goods);
    }

    @Override
    //删除商品
    public void delete(Igoods goods) {
        // TODO Auto-generated method stub
        this.shopgoogs.removedate(goods);
    }

    @Override
    //获取所有的商品
    public Object[] getall() {
        // TODO Auto-generated method stub
    return this.shopgoogs.toArray();
    }

}
class bag implements Igoods{
    private String name;
    private double price;
    public bag(String name,double price ) {
        this.name=name;
        this.price=price;
    }
    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return this.name;
    }

    @Override
```

```
    public double getprice() {
        // TODO Auto-generated method stub
        return this.price;
    }
    public String toString() {
        return "[商品信息：]"+this.name+" -- "+"[商品的价格：]"+this.price;
    }
    public boolean equals(Object obj) {
        if(obj==null) {
            return false;
        }
        if(!(obj instanceof bag)) {
            return false;
        }
        if(obj==this) {
            return true;
        }
        bag other=(bag)obj;
        return this.name.equals(other.name) && this.price==other.price;
    }
}
class book implements Igoods{
    private String name;
    private double price;
    public book(String name,double price ) {
        this.name=name;
        this.price=price;
    }
    @Override
    public String getName() {
        // TODO Auto-generated method stub
        return this.name;
    }

    @Override
    public double getprice() {
        // TODO Auto-generated method stub
        return this.price;
    }
    public String toString() {
        return "[商品信息：]"+this.name+" -- "+"[商品的价格：]"+this.price;
    }
    public boolean equals(Object obj) {
        if(obj==null) {
            return false;
        }
        if(!(obj instanceof book)) {
            return false;
        }
        if(obj==this) {
            return true;
        }
        book other=(book)obj;
        return this.name.equals(other.name) && this.price==other.price;
```

```
    }
}
//收银台
class Cashier{
    //结算总的价钱
    public double allprice(IShop shop) {
        double allprice=0.0;
        Object obj[]=shop.getall();
        for(Object temp:obj) {
            Igoods goods=(Igoods)temp;
            allprice+=goods.getprice();
        }
        return allprice;
    }
    //结算商品的总数
    public int allcount(IShop shop) {
        return shop.getall().length;
    }
}

    public class ITaljava01 {
        public static void main(String[] args) {
        //添加商品
        IShop shop =new Shop();
        shop.addshop(new bag("超级书包",18.9));
        shop.addshop(new bag("中级书包",18.9));
        shop.addshop(new bag("低级书包",11.9));
        shop.addshop(new bag("低级书包",11.9));
        shop.addshop(new book("c++",19.9));
        shop.addshop(new book("java",88.9));
        //所有的信息
        Object object1[] =shop.getall();
        for(Object temp: object1) {
            System.out.println(temp);
        }
        //删除一个信息
        shop.delete(new bag("低级书包",11.9));
        //在此过去所有的删除以后的数据
        System.out.println("---------删除以后获取数据--------------");
        Object object2[] =shop.getall();
        for(Object temp: object2) {
            System.out.println(temp);
        }
        System.out.println("--------计算购物车里面的价钱---------------");
        Cashier cas=new Cashier();
        System.out.println(cas.allprice(shop));
        System.out.println(cas.allcount(shop));
        }
}
```

有兴趣的可以分析一下代码，也比较简单，主要是理解集合里面的一些数据结构。