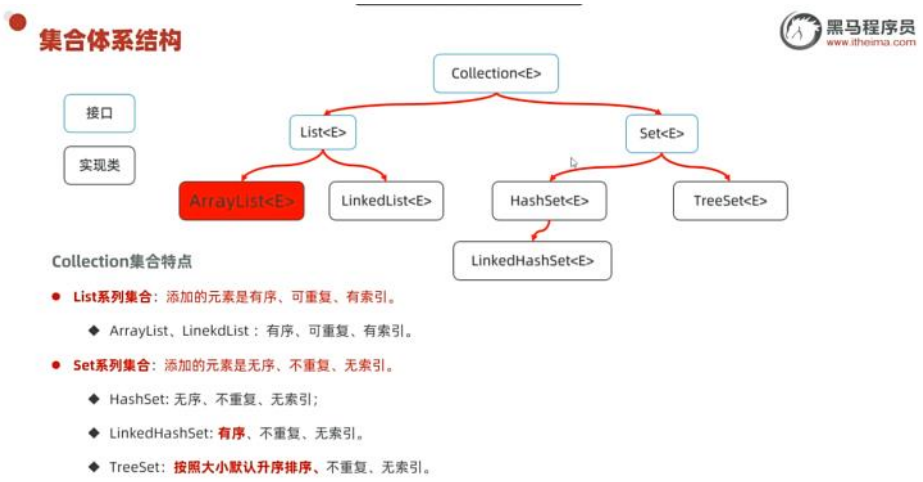
Collection

2024年11月5日 13:43

1.

2.



3.

方法名	说明
public boolean add(E e)	把给定的对象添加到当前集合中
public void clear()	清空集合中所有的元素
public boolean remove(E e)	把给定的对象在当前集合中删除
public boolean contains(Object obj)	判断当前集合中是否包含给定的对象
public boolean isEmpty()	判断当前集合是否为空
public int size()	返回集合中元素的个数。
public Object[] toArray()	把集合中的元素，存储到数组中

Collection的遍历方式一：迭代器遍历

- 迭代器是用来遍历集合的专用方式(数组没有迭代器)，在Java中迭代器的代表是Iterator。

4.

Collection集合获取迭代器的方法

方法名称	说明
Iterator<E> iterator()	返回集合中的迭代器对象，该迭代器对象默认指向当前集合的第一个元素

Iterator迭代器中的常用方法

方法名称	说明
boolean hasNext()	询问当前位置是否有元素存在，存在返回true ，不存在返回false
E next()	获取当前位置的元素，并同时将迭代器对象指向下一个元素处。

5.

```
public Iterator<E> iterator() {
    return new Itr();
}
```

An optimized version of AbstractList.Itr

```
private class Itr implements Iterator<E> {
    int cursor       // index of next element to return
    int lastRet = -1; // index of last element returned; -1 if no such
    int expectedModCount = modCount;

    // prevent creating a synthetic constructor
    Itr() {}

    public boolean hasNext() {
        return cursor != size;
    }
```

Collection的遍历方式二：增强for循环

6.

格式：

```
for (元素的数据类型 变量名 : 数组或者集合) {

}
```

```
Collection<String> c = new ArrayList<>();
...
for(String s : c) {
    System.out.println(s);
}
```

- 增强for可以用来遍历集合或者数组。
- 增强for遍历集合，本质就是迭代器遍历集合的简化写法。

```
String[] array = collection.toArray(new String[0]);
```

需要使用Collection的如下方法来完成

7.

方法名称	说明
default void forEach(Consumer<? super T> action)	结合lambda遍历集合

```
names.forEach(new Consumer<String>() {
    @Override
    public void accept(String s) {
        System.out.println(s);
    }
});
```

```
default void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
}
```

解决并发修改异常问题的方案

① 如果集合支持索引，可以使用for循环遍历，每删除数据后做i--；或者可以倒着遍历

8.

② 可以使用迭代器遍历，并用迭代器提供的删除方法删除数据。

注意：增强for循环/Lambda遍历均不能解决并发修改异常问题，因此增它们只适合做数据的遍历，不适合同时做增删操作。

Iterator.remove();

1、List系列集合的特点是什么？

- ArrayList、LinekdList：有序，可重复，有索引。

2、List提供了哪些独有的方法？

9.

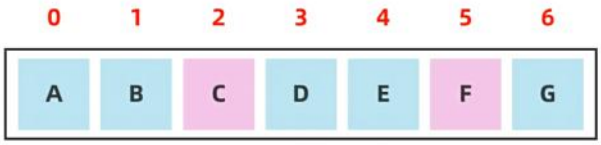
方法名称	说明
void add(int index,E element)	在此集合中的指定位置插入指定的元素
E remove(int index)	删除指定索引处的元素，返回被删除的元素
E set(int index,E element)	修改指定索引处的元素，返回被修改的元素
E get(int index)	返回指定索引处的元素

ArrayList的底层原理

- ArrayList底层是基于数组存储数据的。
- LinkedList底层是基于链表存储数据的。

数组的特点

10.



- **查询速度快**（注意：是根据索引查询数据快）： 查询数据通过地址值和索引定位，查询任意数据耗时相同。
- **增删数据效率低：** 可能需要把后面很多的数据进行前移。

The default initial capacity of an ArrayList is 10. When an ArrayList is created, it starts out empty. After the first element is added, the ArrayList is extended to a capacity of 10. If the ArrayList reaches its full capacity, it increases its size by 1.5 times the original capacity.

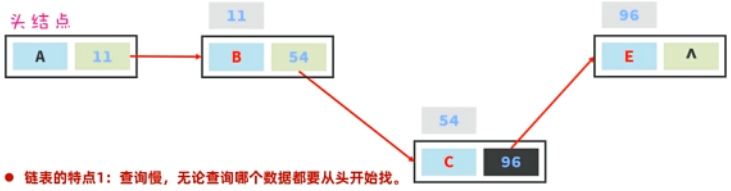
LinkedList的底层原理

- LinkedList底层是基于链表存储数据的。

链表的特点

- 链表中的数据是一个一个独立的结点组成的，结点在内存中是不连续的，每个结点包含数据值和下一个结点的地址。

11.



- **链表的特点1：查询慢，无论查询哪个数据都要从头开始找。**
- **链表的特点2：链表增删相对快**

删除数据CE之间的数据D
①数据C对应的下一个数据地址指向数据E
②数据D删除

LinkedList基于双链表实现

LinkedList集合的底层原理

- 基于双链表实现的。
- 特点：查询慢，增删相对较快，**但对首尾元素进行增删改查的速度是极快的。**

LinkedList新增了：很多首尾操作的特有方法。

12.

方法名称	说明
public void addFirst(E e)	在该列表开头插入指定的元素
public void addLast(E e)	将指定的元素追加到此列表的末尾
public E getFirst()	返回此列表中的第一个元素
public E getLast()	返回此列表中的最后一个元素
public E removeFirst()	从此列表中删除并返回第一个元素
public E removeLast()	从此列表中删除并返回最后一个元素

Set系列集合特点：无序：添加数据的顺序和获取出的数据顺序不一致； 不重复； 无索引；

13.

- HashSet：无序、不重复、无索引。
- LinkedHashSet：**有序**、不重复、无索引。
- TreeSet：**排序**、不重复、无索引。

什么是哈希值？对象的哈希值有什么特点？

- **对象调用Object的hashCode()方法得到的一个随机值**

HashSet集合的底层原理是什么样的？

14.

- **基于哈希表实现的。**
- **JDK8之前的，哈希表：底层使用数组+链表组成**
- **JDK8开始后，哈希表：底层采用数组+链表+红黑树组成。**

哈希表存储数据的详细流程

① 创建一个默认长度16，默认加载因为0.75的数组，数组名table

② 根据元素的哈希值跟数组的长度计算出应存入的位置

③ 判断当前位置是否为null，如果是null直接存入，如果位置不为null，表示有元素，则调用equals方法比较属性值，如果一样，则不存，如果不一样，则存入数组。

④ 当数组存满到16*0.75=12时，就自动扩容，每次扩容成原先的两倍

JDK8开始，当链表长度超过8，且数组长度>=64时，自动将链表转成红黑树

如果希望Set集合认为2个内容相同的对象是重复的应该怎么办？

- **重写对象的hashCode和equals方法。**

15.

```
// 结论：TreeSet集合默认不能 给自定义对象排序啊，因为不知道大小规则。
// 一定要能解决怎么办？两种方案。
// 1. 对象类实现一个Comparable比较接口，重写compareTo方法，指定大小比较规则
// 2. public TreeSet（Comparator c）集合自带比较器Comparator对象，指定比较规则
```

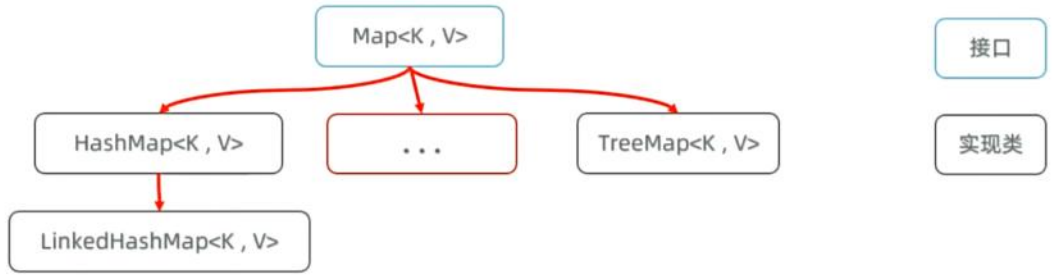
```java
    @Override
    public int compareTo(Teacher o) {
        // 按照年龄升序
//        if(this.getAge() > o.getAge()) return 1;
//        if(this.getAge() < o.getAge()) return -1;
//        return 0;
       // return this.getAge() - o.getAge(); // 升序
        return o.getAge() - this.getAge(); // 降序
    }
```

```java
Set<Teacher> teachers = new TreeSet<>(new Comparator<Teacher>() {
    @Override
    public int compare(Teacher o1, Teacher o2) {
        // return o2.getAge() - o1.getAge(); //降序
    }
}); // 排序，不重复，无索引
```

当集合的比较器与对象的比较器冲突时，优先使用集合的比较器

16.

Map集合的体系



Map集合体系的特点

注意：Map系列集合的特点都是由键决定的，值只是一个附属品，值是不做要求的

- HashMap（由键决定特点）：无序、不重复、无索引；（用的最多）
- LinkedHashMap（由键决定特点）:由键决定的特点：有序、不重复、无索引。
- TreeMap（由键决定特点）:按照大小默认升序排序、不重复、无索引。

17.

方法名称	说明
public V put(K key,V value)	添加元素
public int size()	获取集合的大小
public void clear()	清空集合
public boolean isEmpty()	判断集合是否为空，为空返回true ，反之
public V get(Object key)	根据键获取对应值
public V remove(Object key)	根据键删除整个元素
public boolean containsKey(Object key)	判断是否包含某个键
public boolean containsValue(Object value)	判断是否包含某个值
public Set<K> keySet()	获取全部键的集合
public Collection<V> values()	获取Map集合的全部值

18.

```java
for (Map.Entry<String, Double> entry : entries) {
    String key = entry.getKey();
    double value = entry.getValue();
    System.out.println(key + "====>" + value);
}
```

```
map.put("紫霞", 31);
System.out.println(map); // {嫦娥=28, 铁扇公主=38, 紫霞=31, 女九
// 1、直接调用Map集合的forEach方法完成遍历
map.forEach(new BiConsumer<String, Integer>() {
    @Override
    public void accept(String key, Integer value) {
        System.out.println(key + "=" + value);
    }
});
```

```
default void forEach(BiConsumer<? super K, ? super V> action) {
    Objects.requireNonNull(action);
    for (Map.Entry<K, V> entry : entrySet()) {
        K k;
        V v;
        try {
            k = entry.getKey();
            v = entry.getValue();
        } catch (IllegalStateException ise) {
            // this usually means the entry is no longer in the map
            throw new ConcurrentModificationException(ise);
        }
        action.accept(k, v);
    }
}
```

19.

int到Integer:

int a=3;

[redacted]

或:

20. Integer A=Integer.valueOf(a);

Integer到int:

[redacted]

int a=A.intValue();

至于Integer.parseInt(String str)则是将String类型转为int类型。

21. &&和&都是表示与，区别是&&只要第一个条件不满足，后面条件就不再判断。而&要对所有的条件都进行判断。||和|都是表示“或”，区别是||只要满足第一个条件，后面的条件就不再判断，而|要对所有的条件进行判断。