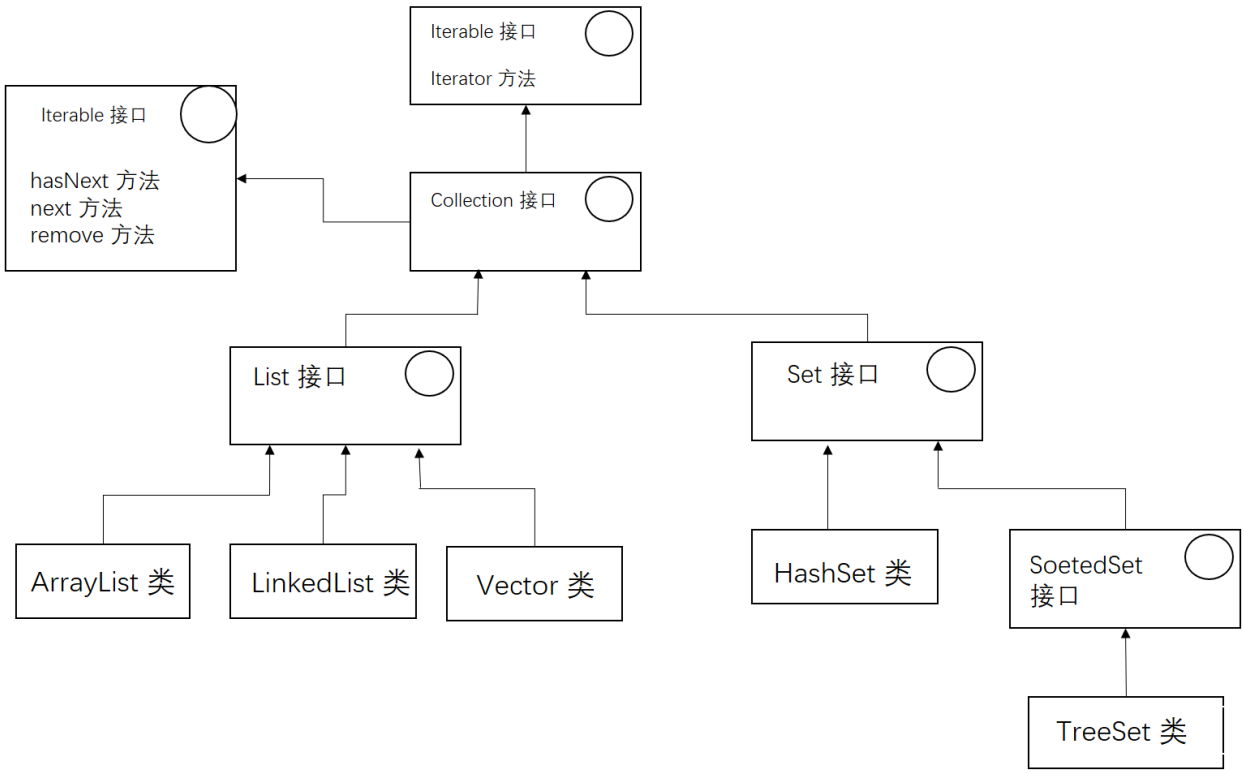
Collection接口

Java中的集合分为两类：

- 一类是单个方式存取元素，这一类的集合中的超级父接口是：java.util.collection；
- 一类是以键值对的方式存取元素，这一类的集合中的超级父接口是：java.util.map；



集合中只能存放引用数据类型。

所有实现Collecrion接口的类都是可遍历的可迭代的。

方法	返回值	功能
hasNext()	boolean	查看集合中有下一个元素没有
next()	-	获取一个集合对象
remove()	boolean	删除一个集合元素

1 List接口

List集合的特点：有序可重复，List集合的实现类都有下标。

- 有序：存进去是这个顺序取出来还是这个顺序，并不会自动排序。有序是因为元素有下标，从零开始以一递增。
- 可重复：可以存进去相同元素。

1.1 ArrayList类

1.1.1 ArrayList

- ArrayList底层是数组，可自动扩容被称为动态数组，是非线程安全的；
- ArrayList默认数组长度是10，每次扩容1.5倍，如果扩容的出小数则取整数位舍弃小数位；
- ArrayList集合在内存中需要大片连续的完整内存；
- ArrayList集合因为是连续的并且有下标支持，所有它的监所效率和修改效率极高 。

构造方法	返回值	作用
new ArrayList()	ArratList	使用默认初始化容量（10）
new ArrayList(20)	ArratList	设置初始化容量（20），不会进行自动扩容

方法（实例方法）	返回值类型	作用
add()	void	加入元素到集合中
get()	Object	获取对应下标的元素
instanceof()	关键字boolean	判断引用是否是那个类型

```
// ArrayList list = new ArrayList();
// 父类的引用 = 子类的对象
List list = new ArrayList(); //推荐这样写

//Object o1 = Integer = 1;
list.add(1); //0
list.add(1.2); //1
list.add("wangyang"); //2
list.add(true); //3

System.out.println(list);

//存进去的是个Int类型，那么取出来的应该也是Int类型
//然而实际情况是你存进去的是个int，取出来的是Object
if (list.get(1) instanceof Integer) {
    Integer i = (Integer) list.get(0);
    System.out.println(i + 1);
} else if (list.get(1) instanceof Double) {
    Double d = (Double) list.get(1);
    System.out.println(d + 2);
}
```

1.1.2 泛型机制

泛型：规范类型。

集合中只要是引用数据类型都能存放，这就让集合中的元素十分混乱不好管理，所以java推出了泛型机制。泛型规范了一个集合对象能存放的类型。

规范写法

```
List<String> list = new ArrayList<String>();
```

缺省写法（推荐使用）

```
List<String> list = new ArrayList<>();
```

```
List<String> list = new ArrayList<String>();
//因为限制了只能保存String类型，那么再取数据的时候就只能获取到String类型，不需要再去类型转换了
list.add("1");
list.add("2");
list.add("3");
System.out.println(list.get(0) instanceof String); //true

List<String> list2 = new ArrayList<>();

//如果泛型是基本数据类型，那么可以使用其对应的包装类
List<Integer> list3 = new ArrayList<>();
list3.add(1); //0
list3.add(2); //1
list3.add(3); //2
System.out.println(list3.get(0) + 1); //2
```

1.1.3 ArrayList常用方法

方法（实例方法）	返回值类型	作用
add(E e)	boolean	向列表的尾部添加指定的元素
add(int index, E element)	void	在列表的指定位置插入指定元素
clear()	void	清空列表中元素
contains(Object o)	boolean	判断列表中是否包含指定元素，如果包含，返回true
get(int index)	E	获取指定位置的元素
isEmpty()	boolean	判断列表中是否存在元素，如果为空，返回true
iterator()	`Iterator<E>`	返回迭代器对象，用来遍历集合中元素。
listIterator()	ListIterator	返回迭代器对象，用来遍历集合中元素
remove(int index)	E	删除指定位置的元素，并返回删除的元素
remove(Object obj)	boolean	删除列表中第一次出现的指定对象，存在true
set(int index , E element)	E	替换列表中指定位置的元素，返回被替换的元素
size()	int	获取列表长度
toArray(T[] a)	`<T> T[]`	把列表转成指定类型的数组，数组长度等于列表长度

```
List<String> list = new ArrayList<>();

list.add("a");
list.add("b");
list.add("c");
list.add("d");
System.out.println(list); //[a, b, c, d]

list.add(1, "e");
list.add(2, "f");
System.out.println(list); //[a, e, f, b, c, d]

list.set(3, "g");
System.out.println(list); //[a, e, f, g, c, d]

int size = list.size();
System.out.println(size); //6

list.remove(3);
System.out.println(list); //[a, e, f, c, d]

String i5 = list.get(4);
System.out.println(i5); //d
```

```
list.clear();
System.out.println(list); //[]

boolean flag = list.contains("a");
System.out.println(flag); //false
```

1.1.4 ArrayList循环遍历

1. for方法可获取元素下标如果要通过下标操作数据建议使用。（只有List集合可以用这种方式）
2. for each增强for循环可以遍历任何集合与数组，方便实用，如果不需要获取下标建议使用。
3. 迭代器方式，所有集合都有迭代器都可以使用迭代器遍历，但是如果集合增加或者删除元素了则得重新获取迭代器对象。通过迭代器的方法修改集合则不用重新获取。

```
List<String> list = new ArrayList<>();
list.add("a");
list.add("b");
list.add("c");
list.add("d");

System.out.println("No1.普普通通for循环");
for (int i = 0; i < list.size(); i++) {
    String s = list.get(i);
    System.out.println(s);
}

System.out.println("No2.迭代器");
for (Iterator<String> itr = list.iterator(); itr.hasNext(); ) {
    String s = itr.next();
    System.out.println(s);
}

System.out.println("No3.增强for循环（for each循环）");
for (String s : list) {
    System.out.println(s);
}
```

1.2 LinkedList类

LinkedList集合底层是一个双向链表数据结构，不需要大片连续空间，每个链表的节点保存一个元素，同时也保存上一个节点的内存地址，和下一个节点的内存地址。如果一个节点上一个节点的内存地址是null那么这个节点就是头节点，如果一个节点的下一个节点的内存地址是null那么他就是尾节点。

LinkedList是一个链表结构，具备一下特点：

1. 分配内存空间不是必须是连续的；
2. 插入、删除操作很快，只要修改前后指针就OK了，时间复杂度为O(1)；
3. 访问比较慢，必须得从第一个元素开始遍历，时间复杂度为O(n)；

LinkedList特有方法

方法名（实例方法）	返回值类型	作用
getFirst()	E	返回头节点存储的元素
getLast()	E	返回尾节点存储的元素

```
List<String> list = new LinkedList<>();
```

1.3 Vector类

Vector底层也是数组结构，与基本ArrayList相同，只不过Vector是线程安全的，不过现在我们已经找到了效率更高的保护线程安全的方法，所有Vector集合不常用了。

Vector 类可以实现可增长的对象数组。与数组一样，它包含可以使用整数索引进行访问的组件。与ArrayList相比，是线程安全的，但是运行速度比数组还慢。

1.4 总结

ArrayList底层是数组结构，修改和查询效率快，添加和删除慢；

LinkedList底层是链表结构，添加和删除效率快，修改和查询慢。

2 Set接口

无序不可重复，Set集合没有下标

- 无序：存进去的东西和取出来的东西顺序不一样。
- 不可重复：不能存储相同元素。

2.1 HashSet类

HashSet底层是（HashMap）哈希表数据结构，默认初始化长度16，扩容加载因子是0.75，每次扩容2倍。

2.1.1 HashSet常用方法

方法名（实例方法）	返回值类型	作用
isEmpty()	boolean	判断集合中是否有元素
size()	int	获取集合元素个数
remove()	boolean	通过元素内容删除元素
contains()	boolean	查看集合中是否包含某个元素
clear()	void	清空集合中的元素
add()	void	往集合中加元素

```
Set<Integer> set = new HashSet<>();

set.add(6);
set.add(6);
set.add(7);
set.add(1);
set.add(521);
set.add(231);
set.add(14154);
set.add(61111);
set.add(52112);
set.add(23121);

System.out.println(set.toString()); //[52112, 1, 23121, 6, 7, 231, 61111, 521, 14154]

System.out.println(set.isEmpty()); //false
System.out.println(set.size()); //9
System.out.println(set.contains(6)); //true
set.remove(6);
System.out.println(set.toString()); //[52112, 1, 23121, 7, 231, 61111, 521, 14154]

set.clear();
System.out.println(set); //[]
```

2.1.2 HashSet遍历方法

因为Set集合没有下标只能通过迭代器与for each的方式遍历。

```
Set<String> set = new HashSet<>();

String s = "";
for (int i = 0; i < 6; i++) {
    s = s + i;
    set.add(s);
}

//增强for循环  for each
for (String ss : set) {
    System.out.println(ss);
}
System.out.println("----------------------");

//迭代器
for (Iterator<String> itr = set.iterator(); itr.hasNext(); ) {
    System.out.println(itr.next());
}
```

2.1.3 HashSet去重

HashSet如何判断元素是否相同？

1. 自动调用泛型类型的hashcode，来判断地址是否相同。如果地址不同，那么直接不重复；

2. 如果hashcode值一样，再去自动调用equals方法，来判断内存空间是否一样，如果一样则认为是同一个元素，不能重复保存反之，如果不一样，则判断元素不重复；
3. 我们可以通过重写泛型类型中的equals和hashcode方法来改变set的判断重复规则，不仅可以判断内存空间，还可以判断数据内容。

注意：如果hashset判断为重复数据，那么只会保存第一个数据，其他重复数据没有机会进入集合中！！！

```
@Override
public boolean equals(Object o)
{
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;
    Student student = (Student) o;
    return id == student.id;
}

@Override
public int hashCode()
{
    return Objects.hash(id);
}
```

通过重写Student类的equals()和hashCode()方法来改变判断重复规则。只要输入对象的ID相同就认为是同一对象，不能重复添加。

2.2 SoetedSet接口

特点：无序不可重复，可自动排序

2.3 TreeSet类

2.3.1 TreeSet常用方法

TreeSet是SortedSet接口的实现类，正如SortedSet名字暗示的，TreeSet可以确保集合元素处于排序状态。

TreeSet采用红黑树的数据结构来存储集合元素，对应的类型必须实现Comparable接口特点：默认自然排序（升序）。

```
Set<Integer> set = new TreeSet<>();

set.add(1);
set.add(3);
set.add(5);
set.add(6);
set.add(8);

System.out.println("自然顺序（升序）"+set); //自然顺序（升序）[1, 3, 5, 6, 8]

Set<String> set2 = new TreeSet<>();

set2.add("A");

set2.add("a");
```

```
set2.add("c");
set2.add("D");
set2.add("d");

System.out.println(set2); //[A, D, a, c, d]

System.out.println(set2.size()); //5
System.out.println(set2.isEmpty()); //false
System.out.println(set2.contains("A")); //true
set2.remove("D");
System.out.println(set2); //[A, a, c, d]
set2.clear();
System.out.println(set2); //[]
```

2.3.2 TreeSet循环遍历

```
Set<String> set = new TreeSet<>();
set.add("A");
set.add("a");
set.add("B");
set.add("D");
set.add("d");

//迭代器
for (Iterator<String> itr = set.iterator(); itr.hasNext(); ) {
    String s = itr.next();
    System.out.println(s);
}
System.out.println("------------------");

//增强for循环  for each
for (String s : set) {
    System.out.println(s);
}
```

2.3.3 内部比较器

```
//如果比较条件全部相同，那TreeSet就认为2者是同一个对象，不能存放
@Override
public int compareTo(Teacher o)
{
    int result = o.id - this.id;
    if(result == 0)
    {
        //this.xx 在前面，升序
        //o.xx 在前面，降序
        return o.getHiredate().compareTo(this.getHiredate());
    }
    return o.id - this.id;
}
```

2.3.4 外部比较器

```
class StudentSort implements Comparator<Student2> {
    @Override
    public int compare(Student2 o1, Student2 o2) {
        int result = (int) (o1.getMoney() - o2.getMoney());
        if (result == 0) {
            result = o1.getDate().compareTo(o2.getDate());
            if (result == 0) {
                result = o1.getId() - o2.getId();
            }
        }
        return result;
    }
}
```

```
/* main函数 */
StudentSort sort = new StudentSort();
Set<Student2> set = new TreeSet<>(sort);
```

2.3.5 匿名外部比较器

```
Comparator sort = new Comparator<Student3>() {
    @Override
    public int compare(Student3 o1, Student3 o2) {
        int result = (int) (o1.getMoney() - o2.getMoney());

        if (result == 0) {
            result = o1.getDate().compareTo(o2.getDate());

            if (result == 0) {
                result = o1.getId() - o2.getId();
            }
        }

        return result;
    }
};

Set<Student3> set = new TreeSet<>(sort);
```