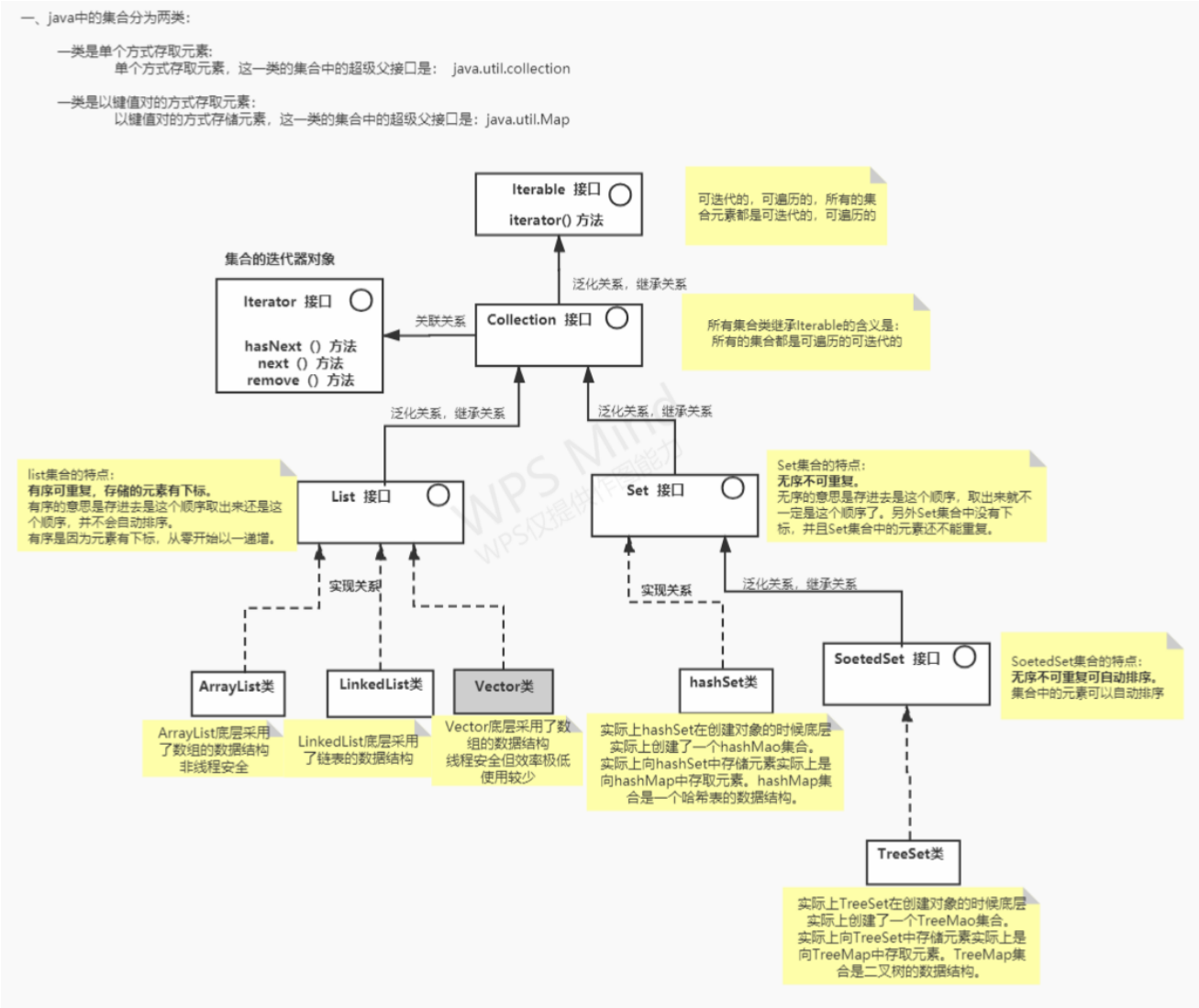
Collection接口



集合中只能存放引用数据类型。

所有实现Collecrion接口的类都是可迭代的

方法	返回值	功能
hasNext（）	boolead	查看集合中有元素没有
next（）		获取一个集合对象
remove（）	boolead	删除一个集合元素（使用这个方法删除不需要重新获取迭代器）

1List接口

List集合的特点：有序可重复，List集合的实现类都有下标

有序的意思是存进去是这个顺序取出来还是这个顺序，可重复的意思是可存进去相同元素，

1.1ArrayList类

1.1.1ArrayList基本概念与方法

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倍数，如果扩容出得出小数则取整数舍弃小数位

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(17)	ArrayList	设置初始化容量

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	获取下标为x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);
        }
```

1.1.2java中的泛型机制

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，所有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 list = new ArrayList();

缺省写法（推荐使用）

List list2 = 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);

        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);
        list3.add(2);
        list3.add(3);

        System.out.println(list3.get(0) + 1);
```

1.1.3ArrayList常用方法

```
/**
 * ArrayList常用方法
 *      boolean         add(E e)                    向列表的尾部添加指定的元素
 *      void            add(int index, E element)   在列表的指定位置插入指定元素
 *      void            clear()                     清空列表中元素
 *      boolean         contains(Object o)          判断列表中是否包含指定元素，如果包含，返回true
 *      E               get(int index)              获取指定位置的元素
 *      boolean         isEmpty()                   判断列表中是否存在元素，如果为空，返回true
 *      Iterator<E>     iterator()                  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
 *      E               remove(int index)           删除指定位置的元素，并返回删除的元素

 *      boolean         remove(Object obj)          删除列表中第一次出现的指定对象，存在true
```

```
*      E               set(int index , E   element)  替换列表中指定位置的元素，返回被替换的元素
*      int             size()                        获取列表长度
*      <T> T[]         toArray(T[] a)                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("ocean");
        list.add("Redred");
        list.add("hai");

        list.add(2,"yang");                     //插队后面的后移
        list.remove("hai");                     //删除指定元素
        list.remove(2);                         //删除指定元素下标
        list.isEmpty();                         //查看集合是否有元素
        System.out.println(list.size());        //获取集合大小
        list.set(1,"red");                      //替换下标位置的元素
        System.out.println(list);
        list.clear();                           //删除所有元素
```

1.1.4ArrayList循环遍历

fori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）

foreach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。

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("1");
        list.add("2");
        list.add("3");
        list.add("4");
        list.add("5");
        System.out.println("============fori============");
        for (int i = 0; i < list.size(); i++) {
            System.out.println(list.get(i));
        }
        System.out.println("============迭代器============");
        //集合增加或者删除要重新获取迭代器
        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext()){
            System.out.println(iterator.next());
        }
        System.out.println("============foreach============");
        for (String s : list){
            System.out.println(s);
        }
        System.out.println("============toString============");
        System.out.println(list.toString());
```

1.2LinkedList类

1.2.1LinkedList集合基础

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)；

```
/**
 * ArrayList常用方法
 *      boolean         add(E e)                        向列表的尾部添加指定的元素
 *      void            add(int index, E element)       在列表的指定位置插入指定元素
 *      void            clear()                         清空列表中元素
 *      boolean         contains(Object o)              判断列表中是否包含指定元素，如果包含，返回true
 *      E               get(int index)                  获取指定位置的元素
 *      boolean         isEmpty()                       判断列表中是否存在元素，如果为空，返回true
 *      Iterator<E>     iterator()                      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
 *      E               remove(int index)               删除指定位置的元素，并返回删除的元素
 *      boolean         remove(Object obj)              删除列表中第一次出现的指定对象，存在true
 *      E               set(int index , E   element)    替换列表中指定位置的元素，返回被替换的元素
 *      int             size()                          获取列表长度
 *      <T> T[]         toArray(T[] a)                  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 LinkedList<>();

        list.add("a");
        list.add("b");
        list.add("c");

        list.remove("a");
        list.remove(0);

        list.set(0,"wangyang");

        for(String str : list)
        {
            System.out.println(str);
        }
```

1.3Vector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，运行速度比数组还慢。

2.Set接口

无序不可重复，Set集合没有下标

无序的意思是存进去的东西和取出来的东西顺序不一样。不可重复就是不能存储相同元素

2.1hashSet类

2.1.1hashSet集合基础与常用方法

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倍数

方法名（实例方法）	返回值类型	作用
isEmpty()	boolean	判断集合中是否有袁术
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<String> set= new HashSet<>();

        System.out.println(set.isEmpty());
        System.out.println(set.size());

        set.add("zhanghaiyang");
        set.add("baiyuntao");
        set.add("weixunhao");
        set.add("wangyang");
        set.add("liyingbin");

        set.remove("liyingbin");

        boolean flag = set.contains("wangyang");
        System.out.println(flag);

        set.clear();

        System.out.println(set);
```

2.1.2hashSet遍历方法

因为Set集合没有下标只能通过迭代器与Foreach的方式遍历

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

        set.add("yangjiuqing");
        set.add("haoxiaodong");
        set.add("zhujianlin");
        set.add("lichao");
        set.add("linshuaiyang");
        set.add("yuanruilei");

        for(Iterator<String> itr = set.iterator();itr.hasNext();)
        {
            String str = itr.next();

            System.out.println(str);
        }
        System.out.println("-------------------------");

        for(String str : set)
        {
            System.out.println(str);
        }
```

2.1.3hashSet存元素

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

        String str = "1";
        String str2 = new String("2");
        System.out.println(str == str2);
        set.add(str);
        set.add(str2);
        set.add("2");
        set.add("3");
        set.add("4");

        System.out.println(set);
```

2.1.4hashSet判断如何相等

equals（）比较对象的内容，

hashcode（）是通过哈希算法把一个对象的内存地址（十六进制数）进行复杂的计算，返回一个数值，如果内存地址相同则计算出的数值也是相同的，如果两个内存地址不相同则计算出的数值也不可能相同。

只有这两个数值都一样的时候才算一样的元素

```
        Student s1 = new Student();
```

```
        Student s2 = new Student();
        s1.name = "张三";
        s2.name = "lisi";

        Set<Student> set = new HashSet<>();

        //set会自动去调用s1对象的equals方法，和当前集合中所有元素比较，
        //如果equals方法返回为真，则set判定两者重复，它就只会保存一个
        //同时调用hashCode方法
        set.add(s1);
        set.add(s2);

        System.out.println(s1.equals(s2));

        System.out.println(set);
```

2.2SoetedSet接口

特点：无序不可重复，可自动排序

2.3TreeSet类

2.3.1TreeSetke

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存储元素对应的类型必须实现Comparable接口特点：默认自然排序

```
        //从小到大排序
        Set<Integer> set = new TreeSet<>();
        set.add(1);
        set.add(3);
        set.add(5);
        set.add(7);
        set.add(9);
        set.add(2);
        set.add(4);
        set.add(6);
        set.add(8);
        set.add(10);
        System.out.println(set);
        //可根基Ascll码排序
        System.out.println("--------------------");
        Set<String> set2 = new TreeSet<>();
        set2.add("A");
        set2.add("B");
        set2.add("e");
        set2.add("f");
        set2.add("f");
        set2.add("a");
        set2.add("c");
        set2.add("K");
        set2.add("z");
        set2.add("Y");
```

```
        set2.add("d");
        set2.add("M");
        set2.add("Z");
        //乱序
        System.out.println(set2);
        System.out.println("--------------------");
        Set<String> set3 = new TreeSet<>();
        set3.add("伍");
        set3.add("壹");
        set3.add("叁");
        set3.add("肆");
        set3.add("贰");

        System.out.println(set3);
```

2.3.2TreeSet循环遍历

因为Set集合没有下标只能通过foreach和迭代器方式获取集合元素

```
        Set<String> set = new TreeSet<>();
        set.add("A");
        set.add("B");
        set.add("e");
        set.add("f");
        set.add("f");
        set.add("a");
        set.add("c");
        set.add("K");
        set.add("z");
        set.add("Y");
        set.add("d");
        set.add("M");
        set.add("Z");

        System.out.println("---增强for--foreach--");
        for(String s : set)
        {
            System.out.println(s);
        }

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

2.3.3TreeSet中存放自定数据类型

主要是两个比较器的用法

comparable(一参)内部比较器

comparator(俩参)外部比较器

内部比较器

```
//内部比较器
public class Emp implements Comparable<Emp> {
@Override
    public int compareTo(Emp o) {
        if (this.sal == o.sal){
            return this.eip - o.eip;
        }
        return (int)(o.sal- this.sal);
    }
}
//内部比较器  直接创建类型集合就能使用
Set<Emp> treeSet1 = new TreeSet<>();
```

外部比较器

```
//外部比较器
class Compare1 implements Comparator<Emp>{
    @Override
    public int compare(Emp o1, Emp o2) {
        if (o2.getSal()==o1.getSal()){
           return o1.getHiredate().compareTo(o2.getHiredate());
        }
        return (int) (o2.getSal()-o1.getSal());
    }
}
//外部比较器  使用的时候要用TreeSet集合的有参构造方法
Set<Emp> treeSet2 = new TreeSet<>(new Compare1());
```

匿名内部类的形式

```
//匿名内部类
Set<Emp> treeSet3 = new TreeSet<>(new Comparator<Emp>() {
    public int compare(Emp o1, Emp o2) {
        if (o2.getSal()==o1.getSal() &&
                o2.getHiredate().compareTo(o1.getHiredate())==0){
            return o1.getEip()-o2.getEip();
        }
        if (o2.getSal()==o1.getSal()){
            return o1.getHiredate().compareTo(o2.getHiredate());
        }
        return (int) (o2.getSal()-o1.getSal());
    }
});
```