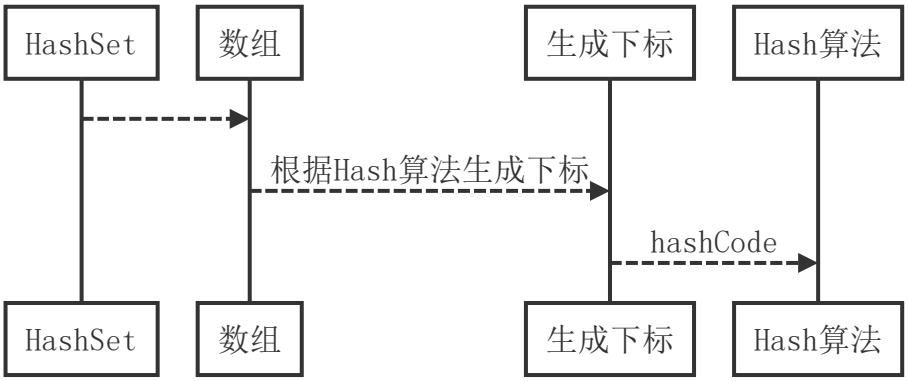
集合2课件

1.Set

1.1 简介

```
Set和List一样，可以看成可变长度的数组的接口
Set是一种不重复且无序的集合，
即对于满足e1.equals(e2)条件的e1与e2对象元素，不能同时存在于同一个Set集合里
因此，为Set集合里的元素的实现类实现一个有效的equals(Object)方法
```



1.2 实现类

```
Set的实现类中，我们推荐：
HashSet   底层通过Hash表实现
TreeSet   通过树来实现
```

1.3 HashSet

```
    HashSet类是对AbstractSet类的扩展。它创建了一个类集。该类集使用散列表进行存储，而散列表则通过使用称之为散列法的机制来存储信息。在散列中，一个关键字的信息内容被用来确定唯一的一个值，称为散列码。而散列码则被用来当作与关键字相连的数据的存储下标
```

1.3.1 构造方法

```
Set hashSet=new HashSet();  //默认初始化
Set hashSet=new HashSet(conllection)// 用一个集合去初始化
Set hashSet=new Hashset(size) ;//用初始化因子去初始化
```

1.3.2 常见方法

public boolean add(E o)	向集合添加指定元素
public void clear()	清空集合中所有元素
public boolean contains(Object o)	判断集合是否包含指定元素
public boolean isEmpty()	判断集合是否还有元素。如果集合不包含任何元素，则返回true
public Iterator iterator()	返回对此集合中元素进行迭代的迭代器
public boolean remove(Object o)	删除集合中的元素
public int size()	返回此集合中的元素的个数
public Object[] toArray	将集合中的元素放到数组中，并返回该数组

HashSet加入元素流程

```
 *    3.HashSet源码
 *       3.1 HashSet底层就是一个数组
 *       3.2 在操作的时候,对HashSet只是添加数据,没有传入下标,Hash算法帮我们生成一个下标
 *       3.3 Hash算法根据我们提供的hashCode()的值,生成当前元素的下标
 *       3.4 当两个对象hashcode()不同,它们会放入数组的两个位置,跟equals没有关系
 *       3.5 两个对象的hashcode相同,用equals判断,分下面两种情况
 *             3.5.1 如果equals也相同,那么说明他们是同一个值,后面的值是覆盖还是不添加?
 *             3.5.2 如果equals不相同,要根据JDK版本分两种情况
 *                   3.5.2.1 JDK8以前 那么把原来的值和当前的值放入链表,并且把链表放入数组的当前位置(hash碰撞)
 *                   3.5.2.2 JDK8以后,分为两种情况
```

```
 *                                3.5.2.2.1: 数组元素小于等于64 当前元素放入链表,
 *                                3.5.2.2.2  数组元素大于64,当前链表转为红黑树,把元素放如树中
 *        3.6总结: 数组--->数组+链表--->数组+链表+红黑树
 *     4.注意事项:
 *        4.1 当重写equals方法的时候,必须重写hashcode.
 *        4.2 当两个都重写,equals相等,hashcode一定要相等,
 *        4.3 当两个对象equals不相同,hashcode有可能相同,如果相同,会发生hash碰撞,降低性能
 *        4.4 假设一个类中所有对象的hashcode都相同,那么他们放入HashSet中,底层编程了纯粹的链表
 *        4.5 为了降低hash碰撞,尽量少使用手写hashCode,多采用系统生成的hashCode方法,
```

1.4.TreeSet

TreeSet为使用树来进行存储的Set接口提供了一个工具。对象按升序进行存储，这方便我们对其进行访问和检索。在存储了大量的需要进行快速检索的排序信息的情况下，TreeSet是一个很好的选择。

1.4.1初始化方法

```
Set set=new TreeSet()             // 使用默认构造器
Set set=new TreeSet(Collcection) //传入一个集合做初始化值
Set set=new TreeSet(Comparator)  //传入一个比较器初始化
```

1.4.2 特殊方法

方法	功能描述
public E first()	返回有序集合中第一个元素，即最小的那个元素
public E last()	返回有序集合中最后一个元素，即最大的那个元素
public SortedSet subSet(E fromElement,E toElement)	返回有序集合从fromElement（包括）到toElement（不包括）的元素

1.4.3 Tree排序原则

1.自然排序

```
默认在加入TreeSet里面的对象要求这个类要实现Comparable接口
并且实现这个接口中提供的compareTo排序方法，TreeSet就能自动调用这个方法为我们排序
```

```
**
 * 在实现接口的时候,如果确定了排序的对象,那么最好通过泛型传入要排序类,实现的方法中就不需要做强转
 * 需求: 先按照姓名排序,当姓名相同时,再按照年龄排序
 */
public class Cat implements Comparable<Cat> {
    private String name;
    private int age;

    @Override
```

```
    public int compareTo(Cat o) {
    //极简写法
        return (name.compareTo( o.name ) == 0) ? (Integer.compare( age, o.age )) : name.compareTo( o.name );
    }
}
```

2.定制排序

```
对象不用实现接口,耦合度低
在构建TreeSet的时候传入自定义的排序接口Comparator即可
```

```
    //写一个局部类,用来实现比较器接口
        class JinCatCompartor implements Comparator<JinCat> {
            @Override
            public int compare(JinCat o1, JinCat o2) {
                return (o1.getName().compareTo( o2.getName() ) == 0) ? (Integer.compare( o1.getAge(), o2.getAge() )) : o1.getName().compareTo( o2.getName() );
            }
        }

        //在TreeSet初始化的时候可以传入比较器对象,这时加入到TreeSet中的元素就不需要实现Comparable接口,降低了对象的耦合度
        Set set = new TreeSet(new JinCatCompartor());
        set.add( new JinCat("jinCat",8) );
        set.add( new JinCat("jinCat4",2) );
        set.add( new JinCat("jinCat2",2) );

        System.out.println( "set = " + set );
```

1.5集合遍历

可以转化为iterater来遍历,但一般使用增强型循环:

```
for (Object object : set) {
       System.out.println( "o = " + object );
}
```

2.Map

```
映射（map）是一个存储关键字和值的关联，或者说是“关键字/值”对的对象，即给定一个关键字，可以得到它的值。关键字和值都是对象，关键字必须是唯一的，但值是可以重复的.
```

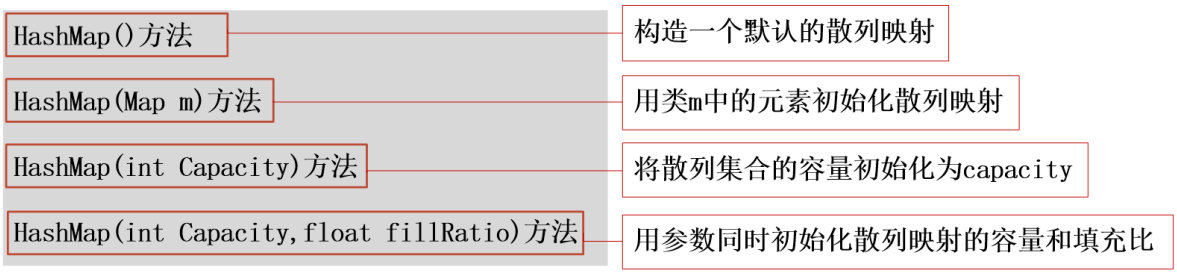
```
Map接口映射唯一关键字到值。关键字是以后用于检索值的对象。给定一个关键字和一个值，可以存储这个值到一个Map对象中。当这个值被存储以后，就可以使用它的关键字来检索它。Map.Entry接口使得可以操作映射的输入。而SortMap接口扩展了Map，它确保了各项关键字按升序排列。
```

2.1 Map实现类

类名	类的描述
AbstractMap	实现大多数的Map接口
HashMap	将AbstractMap扩展到使用散列表
TreeMap	将AbstractMap扩展到使用树

2.2 HashMap

2.2.1 HashMap初始化



2.2.2 HashMap主要方法

方法	方法说明
public void clear()	删除映射中所有映射关系
public boolean containsKey(Object key)	判断HashMap中是否包指定的键的映射关系，如果包含则返回true
public boolean containsValue(Object value)	判断HashMap中是否包指定的键值的映射关系
public V get(Object key)	返回参数key键在该映射中所映射的值
public boolean isEmpty()	判断HashMap中是否包含键-值映射关系，如果不包含则返回true
public V put(K key, V value)	在映射中放入指定值与指定键
public void putAll(Map m)	将指定映射的所有映射关系复制到此映射中
public int size()	返回映射中键-值映射关系的数目
public V remove(Object key)	删除映射中存在该键的映射关系

2.2.3 HashMap底层原理

```
1 HashMap实现Map并扩展AbstractMap。同HashSet一样，HashMap也不保证它的元素的顺序。
2 在向HashMap中添加元素时，不但要将元素添加，还要为每个元素设置一个Hash码。Hash码不只可以为数字，同样它也可以为字符串。
```

2.3 TreeMap

```
1 TreeMap类通过使用树实现Map接口。TreeMap提供了按排序顺序存储关键字/值对的有效手段，同时允许快速检索。不像散列映射，树映射保证它的元素按照关键字升序排序。
```

2.3.1 TreeMap的初始化方法

2.3.2 TreeMap的方法

```
1 TreeMap实现SortedMap并且扩展AbstractMap。本身并没有定义其他方法
```

方法	方法说明
clear()	从此TreeMap中删除所有映射关系
clone()	返回TreeMap实例的浅表复制
comparator()	返回用于对此映射进行排序的比较器，如果此映射使用它的键的自然顺序，则返回null
containsKey(Objectkey)	如果此映射包含对于指定的键的映射关系，则返回true
containsValue(Objectvalue)	如果此映射把一个或多个键映射到指定值，则返回true
entrySet()	返回此映射所包含的映射关系的set视图
firstKey()	返回有序映射中当前第一个键
get(Object key)	返回此映射中映射到指定键的值
headMap(Kto Key)	返回此映射的部分视图，其键严格小于toKey
keySet()	返回此映射中所包含的键的Set视图
lastKey()	返回有序映射中当前最后一个键

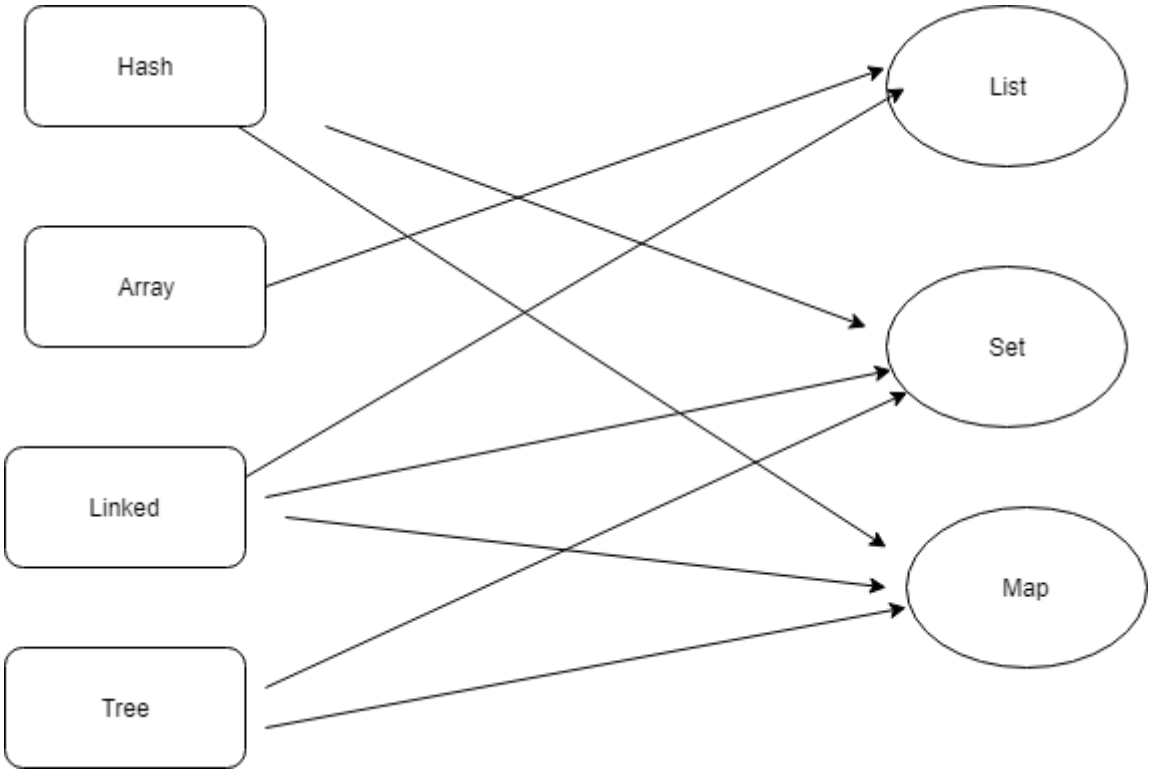
2.4 Hashtable

```
Hashtable 是已经过时的一个Map实现结构,项目中不建议使用,只做了解
HashMap:   线程不安全   效率高
Hashtable   线程安全  效率低
```

2.5 Map遍历

```
Map是因为存储的是键值对,无法直接遍历,一般先使用keyset拿到所有key的集合,然后再通过key去遍历,如下:
```

```
        HashMap hashMap = new HashMap();
        hashMap.put( "userName", "Eric" );
        hashMap.put( "userAge", "16" );
        Set keys = hashMap.keySet();// 得到所有的key，并保存在一个Set中
        for (Object key : keys) {
            System.out.println( "value " + hashMap.get( key ) );
        }
```



3.工具类

Collections

Arrays

Objects