TreeSet

LinkedHashSet 和 TreeSet 都是存储一组有序且唯一的数据，但是这里的两个有序是有区别的。

LinkedHashSet 的有序是指元素的存储顺序和遍历顺序是一致的。

6,3,4,5,1,2-->6,3,4,5,1,2

TreeSet 的有序是指集合内部会自动对所有的元素按照升序进行排列，无论存入的顺序是什么，遍历的时候一定按照生序输出。

```
package com.southwind.demo;

import java.util.Iterator;
import java.util.TreeSet;

public class Test {
    public static void main(String[] args) {
        TreeSet treeSet = new TreeSet();
//        treeSet.add(1);
//        treeSet.add(3);
//        treeSet.add(6);
//        treeSet.add(2);
//        treeSet.add(5);
//        treeSet.add(4);
//        treeSet.add(1);
        treeSet.add("b11");
        treeSet.add("e22");
        treeSet.add("a33");
        treeSet.add("c44");
        treeSet.add("d55");
```

```
        System.out.println("treeSet的长度
是"+treeSet.size());
        System.out.println("treeSet遍历");
        Iterator iterator = treeSet.iterator();
        while(iterator.hasNext()){
            System.out.println(iterator.next());
        }
    }
}
```

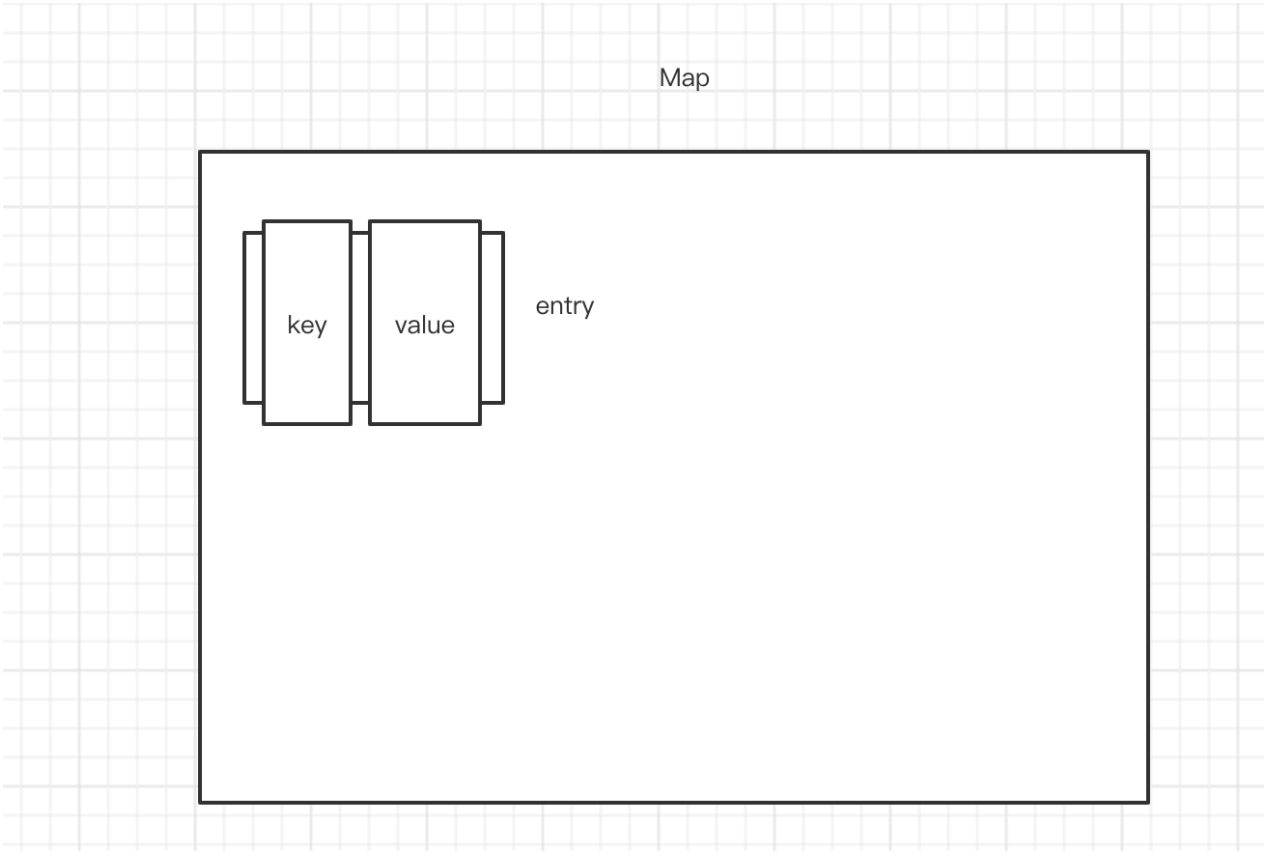
```
package com.southwind.demo2;

import java.util.Iterator;
import java.util.TreeSet;

public class Test {
    public static void main(String[] args) {
        TreeSet treeSet = new TreeSet();
        treeSet.add(new Data(1));
        treeSet.add(new Data(3));
        treeSet.add(new Data(6));
        treeSet.add(new Data(2));
        treeSet.add(new Data(5));
        treeSet.add(new Data(4));
        treeSet.add(new Data(1));
        System.out.println("treeSet的长
度"+treeSet.size());
        System.out.println("treeSet遍历");
        Iterator iterator = treeSet.iterator();
        while(iterator.hasNext()){
            System.out.println(iterator.next());
```

```
        }
    }
}

class Data implements Comparable{
    private int num;

    public Data(int num) {
        this.num = num;
    }

    /**
     * A.compareTo(B)
     * 返回值:
     * 1 表示A大于B
     * 0 表示A等于B
     * -1 表示A小于B
     * @param o
     * @return
     */
    @Override
    public int compareTo(Object o) {
        if(o instanceof Data){
            Data data = (Data) o;
            if(this.num < data.num){
                return 1;
            }else if(this.num == data.num){
                return 0;
            }else{
                return -1;
            }
        }
        return 0;
```

```
    }

    @Override
    public String toString() {
        return "Data{" +
                "num=" + num +
                '}';
    }
}
```

Map

key-value，数据字典



List、Set 接口都是 Collection 的子接口，Map 接口是与 Collection 完全独立的另外一个体系。

List & Set VS Map

List & Set & Collection 只能操作单个元素，Map 可以操作一对元素，因为 Map 存储结构是 key - value 映射。

Map 接口定义时使用了泛型，并且定义两个泛型 K 和 V，K 表示 key，规定键元素的数据类型，V 表示 value，规定值元素的数据类型。

方法	描述
int size()	获取集合长度
boolean isEmpty()	判断集合是否为空
boolean containsKey(Object key)	判断集合中是否存在某个 key
boolean containsValue(Object value)	判断集合中是否存在某个 value
V get(Object key)	取出集合中 key 对应的 value
V put(K key,V value)	向集合中存入一组 key-value 的元素
V remove(Object key)	删除集合中 key 对应的 value
void putAll(Map map)	向集合中添加另外一个 Map
void clear()	清除集合中所有的元素
Set keySet()	取出集合中所有的 key，返回一个 Set
Collection values()	取出集合中所有的 value，返回一个 Collection
Set<Map.Entry<K,V>> entrySet()	将 Map 以 Set 的形式输出
int hashCode()	获取集合的散列值
boolean equals(Object o)	比较两个集合是否相等

Map 接口的实现类

- HashMap：存储一组无序，key 不可以重复，value 可以重复的元素。
- Hashtable：存储一组无序，key 不可以重复，value 可以重复的元素。
- TreeMap：存储一组有序，key 不可以重复，value 可以重复的元素，可以按照 key 进行排序。

HashMap 的使用

```
package com.southwind.demo4;

import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class Test {
    public static void main(String[] args) {
        HashMap hashMap = new HashMap();
        hashMap.put("h","Hello");
        hashMap.put("w","World");
        hashMap.put("j","Java");
        hashMap.put("s","JavaSE");
        hashMap.put("m","JavaME");
        hashMap.put("e","JavaEE");
        System.out.println(hashMap);
        hashMap.remove("e");
        System.out.println("删除之后"+hashMap);
        hashMap.put("m","Model");
```

```
        System.out.println("添加之后"+hashMap);
        if (hashMap.containsKey("a")){
            System.out.println("集合中存在key=a");
        }else{
            System.out.println("集合中不存在key=a");
        }
        if(hashMap.containsValue("Java")){
            System.out.println("集合中存在
value=Java");
        }else {
            System.out.println("集合中不存在
value=Java");
        }
        Set keys = hashMap.keySet();
        System.out.println("集合中的key");
        Iterator iterator = keys.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
        Collection values = hashMap.values();
        for (Object value : values) {
            System.out.println(value);
        }
        System.out.println("************");
        iterator = keys.iterator();
        while(iterator.hasNext()){
            String key = (String) iterator.next();
            String value = (String)
hashMap.get(key);
            System.out.println(key+"-"+value);
        }
    }
}
```

Hashtable 用法与 HashMap 基本一样，它们的区别是，Hashtable 是线程安全的，但是性能较低。HashMap 是非线程安全的，但是性能较高。

HashMap，方法没有用 synchronized 修饰，所以是非线程安全的。

```
 */
public V put(K key, V value) {
    return putVal(hash(key), key, value, onlyIfAbsent: false, evict: true);
}
```

Hashtable，方法用 synchronized 修饰，所以是线程安全的。

```
public synchronized V put(K key, V value) {
    // Make sure the value is not null
    if (value == null) {
        throw new NullPointerException();
    }

    // Makes sure the key is not already in the hashtable.
    Entry<?,?> tab[] = table;
    int hash = key.hashCode();
    int index = (hash & 0x7FFFFFFF) % tab.length;
    /unchecked/
    Entry<K,V> entry = (Entry<K,V>)tab[index];
    for(; entry != null ; entry = entry.next) {
        if ((entry.hash == hash) && entry.key.equals(key)) {
            V old = entry.value;
            entry.value = value;
            return old;
        }
    }

    addEntry(hash, key, value, index);
    return null;
}
```

Hashtable 的使用

```
package com.southwind.demo5;

import java.util.Collection;
import java.util.Hashtable;
```

```
import java.util.Set;

public class Test {
    public static void main(String[] args) {
        Hashtable hashtable = new Hashtable();
        hashtable.put("h","Hello");
        hashtable.put("w","World");
        hashtable.put("j","Java");
        hashtable.put("s","JavaSE");
        hashtable.put("m","JavaME");
        hashtable.put("e","JavaEE");
        System.out.println(hashtable);
        hashtable.remove("e");
        System.out.println(hashtable);

 System.out.println(hashtable.containsKey("a"));

 System.out.println(hashtable.containsValue("Java"))
;
        Set keys = hashtable.keySet();
        System.out.println(keys);
        Collection values = hashtable.values();
        System.out.println(values);
    }
}
```

HashMap 和 Hashtable，保存的书画家都是无序的，Map 的另外一个实现类 TreeMap 主要功能是按照 key 对集合中的元素进行排序。

TreeMap 的使用

```
package com.southwind.demo6;
```

```
import java.util.Iterator;
import java.util.Set;
import java.util.TreeMap;

public class Test2 {
    public static void main(String[] args) {
        TreeMap treeMap = new TreeMap();
        treeMap.put(new User(3,"Java"),"Java");
        treeMap.put(new User(5,"JavaME"),"JavaME");
        treeMap.put(new User(1,"Hello"),"Hello");
        treeMap.put(new User(6,"JavaEE"),"JavaEE");
        treeMap.put(new User(2,"World"),"World");
        treeMap.put(new User(4,"JavaSE"),"JavaSE");
        System.out.println(treeMap);
        Set set = treeMap.keySet();
        Iterator iterator = set.iterator();
        while(iterator.hasNext()){
            Object key = iterator.next();
            System.out.println(key+"-
"+treeMap.get(key));
        }
    }
}

class User implements Comparable{
    private int id;
    private String name;

    public int getId() {
        return id;
    }
```

```
    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public User(int id, String name) {
        this.id = id;
        this.name = name;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }

    @Override
    public int compareTo(Object o) {
        if (o instanceof User){
            User user = (User)o;
            if(this.id > user.id){
                return 1;
            }else if(this.id == user.id){
                return 0;
```

```
            }else {
                return -1;
            }
        }
        return 0;
    }
}
```