Map

Map集合的特点：
`public interface Map<K,V>`

1.Map集合是一个双列集合，一个元素包含两个值（一个key,yige value） 2.Map集合中的元素，key和value的数据类型可以相同，也可以不相同
3.Map集合中的元素，key是不允许重复的，value是可以重复的
4.Map集合中的元素，key和value是一一对应

put()

```
public class MapDemo {
    public static void main(String[] args) {
        putMethod();
    }

    /**
     * V put(K key,V value):把指定的键与指定的值添加到Map集合中
     * 返回值：V
     *      存储键值对的时候，key不重复，返回值V是null
     *      存储键值的时候，key重复，会使用新的value替换重复的vlaue,返回被替换的value值
     */
    public static  void putMethod(){
        Map<String,String> map = new HashMap<String,String>();
        String v1 = map.put("hh", "cxy");
        System.out.println("v1: "+v1); //v1: null
        String v2 =  map.put("hh", "club");
        System.out.println("v2: "+v2); //v2: cxy
        System.out.println(map);//{hh=club}

        map.put("aa", "kk");
        map.put("bb", "kk");
        map.put("cc", "kk");
        System.out.println(map);//{hh=club, aa=kk, bb=kk, cc=kk}
    }
}
```

Map的两种遍历方法

第一种：Set<K> keySet()
把Map集合中的所有的key取出来存储到Set集合中

```
public class MapDemo {
   public static void main(String[] args) {
       ketSetMethod();
   }
   /**
```

```
     * Map集合中的第一种遍历方式：通过键找值的方式
     * Set<K> keySet()
     */

    public static void ketSetMethod(){
        Map<String,String> map = new HashMap<String,String>();
        map.put("aa", "AA");
        map.put("bb", "BB");
        map.put("cc", "CC");
        //使用Map集合中的keySet()，把Map集合中的所有key取出来，存储到一个Set集合中
        Set<String> set = map.keySet();
        //遍历Set集合，获取Map集合中的每一个key
        //使用迭代器遍历Set集合
        Iterator<String> it = set.iterator();
        while(it.hasNext()){
            String key = it.next();
            //通过Map集合的方法get(key),通过key找到value
            String value = map.get(key);
            System.out.println(value);//AA BB CC
        }
    //也可以通过增强for循环

    for (String key : map.keySet()) {
            String value = map.get(key);
            System.out.println(value);//AA BB CC
        }

    }
}
```

第二种方法：Set<Map.Entry<K,V>> entrySet()

```
public class MapDemo2 {
    public static void main(String[] args) {
        Map<String,String> map = new HashMap<String,String>();
        map.put("aa", "AA");
        map.put("bb", "BB");
        map.put("cc", "CC");
        //使用Map集合中的entrySet()方法，返回此映射关系的set视图
        Set<Map.Entry<String, String>> entries = map.entrySet();
        //使用set集合获取每一个entry对象
        Iterator<Map.Entry<String, String>> it = entries.iterator();
        while(it.hasNext()){
            Map.Entry<String, String> entry = it.next();
            //使用Entry对象中的方法getKey()和getValue()获取键与值
            String key = entry.getKey();
            String value = entry.getValue();
            //输出结果 aa AA bb BB cc CC
            System.out.println(key);
            System.out.println(value);
        }
    }
}
```

Map存储自定义类型键值

1.创建一个自定义类

```
public class Person {
    private String name;
    private Integer age;


    public Person(String name, Integer age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

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

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }

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

//重写equals()方法
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Person person = (Person) o;
        return Objects.equals(name, person.name) &&
                Objects.equals(age, person.age);
    }
//重写hashCode()方法
    @Override
    public int hashCode() {

        return Objects.hash(name, age);
    }
}
```

2.以自定义类作为Map的key值

```
private static void method1() {
        Map<String,Person> map = new HashMap<String,Person>();
        map.put("aa", new Person("花花",18));
        map.put("bb", new Person("康康",20));
        map.put("cc", new Person("宝宝",23));
        map.put("aa", new Person("李靖",23));

        for (String key : map.keySet()) {
            Person value = map.get(key);

            /**
             * person类必须重写toString()方法
             * 输出结果：
             *  Person{name='李靖', age=23}
             *  Person{name='康康', age=20}
             *  Person{name='宝宝', age=23}
             *
             *  李靖直接覆盖掉花花
             */
            System.out.println(value);
        }
    }
```

3.以自定义类作为Map的value值

```
private static void method1() {
        Map<String,Person> map = new HashMap<String,Person>();
        map.put("aa", new Person("花花",18));
        map.put("bb", new Person("康康",20));
        map.put("cc", new Person("宝宝",23));
        map.put("aa", new Person("李靖",23));

        for (String key : map.keySet()) {
            Person value = map.get(key);

            /**
             * person类必须重写toString()方法
             * 输出结果：
             *  Person{name='李靖', age=23}
             *  Person{name='康康', age=20}
             *  Person{name='宝宝', age=23}
             *
             *  李靖直接覆盖掉花花
             */
            System.out.println(value);
        }
    }
```

HashMap

HashMap集合的特点：

`public class HashMap<K,V>extends AbstractMap<K,V>implements Map<K,V>, Cloneable, Serializable`

1.HashMap集合是一个哈希表：查询速度非常快
1).jdk1.8之前：数组+单向链表
2).jdk1.8之后：数组+双向链表+红黑树(链表的长度超过8)：提高查询的速度
2.HashMap集合是一个无序的集合，存储元素和取出元素的顺序有可能不一致 3).hashMap集合是一个线程不安全的集合，是多线程的集合，速度快 4).可以存储null值，null键

LinkedHashMap

LinkedHashMap的特点：

`public class LinkedHashMap<K,V>extends HashMap<K,V>implements Map<K,V>`

1.LinkedHashMap集合底层是哈希表+链表(保证迭代的顺序)
2.linkedHashMap集合是一个有序的集合，存储元素和取出元素的顺序是一致的

HashTable(一般不用它，已经被hashMap替换掉了)

HashTable的特点： 1).HashTable底层是一个哈希表
2).HashTable是一个线程安全的集合，是单线程 3).不可以存储null值，null键

Map面试经典案例：计算一个字符串中每个字符串出现的次数


```
/**
 * Map经典面试题案例:计算一个字符串中每个字符出现的此时
 */
public class MapDemo4 {
    public static void main(String[] args) {
        System.out.println("请输入一个字符串:");
        Scanner sc = new Scanner(System.in);
        String str = sc.next();
```

```
        System.out.println(str);
        Map<Character, Integer> map = calculateCharCount(str);
        System.out.println(map);
    }

    private static Map<Character,Integer> calculateCharCount(String str) {
        Map<Character,Integer> map = new HashMap<>();
        //第一种方法：利用String类的方法 char charAt(int index) 返回指定索引处的 char 值。
      /*  for (int i = 0; i < str.length(); i++) {
            char c = str.charAt(i);
            if(!map.containsKey(c)){
                map.put(c, 1);
            }else{
                Integer count = map.get(c);
                count++;
                map.put(c, count);
            }
        }*/

      //第二种方法：利用String类的方法  char[] toCharArray() 将此字符串转换为一个新的字符数
组。
        char[] chars = str.toCharArray();
        for (int i = 0; i < chars.length; i++) {
            char c = chars[i];
            if(!map.containsKey(c)){
                map.put(c, 1);
            }else{
                Integer count = map.get(c);
                count++;
                map.put(c, count);
            }
        }
        return map;
    }

}
```