

day04 【Map】

主要内容

- Map集合

教学目标

- ☐ 能够说出Map集合特点
- ☐ 使用Map集合添加方法保存数据
- ☐ 使用“键找值”的方式遍历Map集合
- ☐ 使用“键值对”的方式遍历Map集合
- ☐ 能够使用HashMap存储自定义键值对的数据
- ☐ 能够使用可变参数
- ☐ 能够使用HashMap编写斗地主洗牌发牌案例

第一章 Map集合

1.1 概述

现实生活中，我们常会看到这样的一种集合：IP地址与主机名，身份证号与个人，系统用户名与系统用户对象等，这种一一对应的关系，就叫做映射。Java提供了专门的集合类用来存放这种对象关系的对象，即 `java.util.Map` 接口。

我们通过查看 `Map` 接口描述，发现 `Map` 接口下的集合与 `Collection` 接口下的集合，它们存储数据的形式不同，如下图。

Collection 接口 定义了 单列集合规范
每次 存储 一个元素 单个元素

单身集合

Collection<E>

Map 接口
定义了 双列集合的规范

每次 存储 一对儿元素

Map<K,V>

K 代表键的类型

夫妻对儿集合

V 代表值的类型

Key 键

Value 值

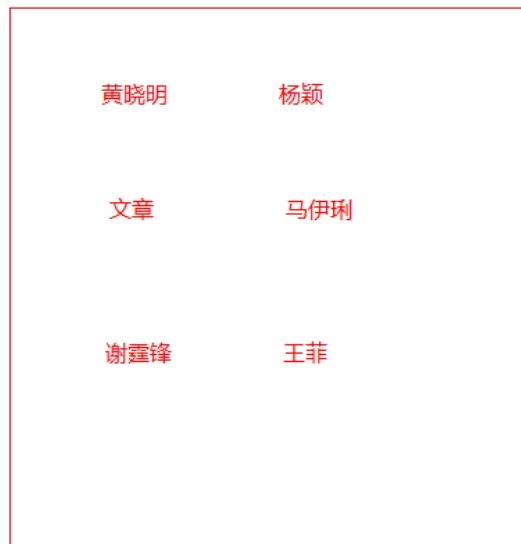


通过 键 可以找 对应的值

1: 键唯一 (值可以重复)

2: 键和价值——映射
一个键对应一个值

3: 靠键维护他们关系



- Collection 中的集合，元素是孤立存在的（理解为单身），向集合中存储元素采用一个个元素的方式存储。
- Map 中的集合，元素是成对存在的(理解为夫妻)。每个元素由键与值两部分组成，通过键可以找对所对应的值。
- Collection 中的集合称为单列集合， Map 中的集合称为双列集合。
- 需要注意的是， Map 中的集合不能包含重复的键，值可以重复；每个键只能对应一个值。

1.2 Map的常用子类

通过查看Map接口描述，看到Map有多个子类，这里我们主要讲解常用的HashMap集合、LinkedHashMap集合。

- **HashMap**：存储数据采用的哈希表结构，元素的存取顺序不能保证一致。由于要保证键的唯一、不重复，需要重写键的hashCode()方法、equals()方法。
- **LinkedHashMap**：HashMap下有个子类LinkedHashMap，存储数据采用的哈希表结构+链表结构。通过链表结构可以保证元素的存取顺序一致；通过哈希表结构可以保证的键的唯一、不重复，需要重写键的hashCode()方法、equals()方法。

tips: Map接口中的集合都有两个泛型变量,在使用时，要为两个泛型变量赋予数据类型。两个泛型变量的数据类型可以相同，也可以不同。

1.3 Map的常用方法

Map接口中定义了很多方法，常用的如下：

- `public V put(K key, V value)`：把指定的键与指定的值添加到Map集合中。
- `public V remove(Object key)`：把指定的键 所对应的键值对元素 在Map集合中删除，返回被删除元素的值。
- `public V get(Object key)` 根据指定的键，在Map集合中获取对应的值。
- `public Set<K> keySet()`：获取Map集合中所有的键，存储到Set集合中。
- `public Set<Map.Entry<K,V>> entrySet()`：获取到Map集合中所有的键值对对象的集合(Set集合)。



- `public boolean containKey(Object key)` :判断该集合中是否有此键。

Map接口的方法演示

```
public class MapDemo {
    public static void main(String[] args) {
        //创建 map对象
        HashMap<String, String> map = new HashMap<String, String>();

        //添加元素到集合
        map.put("黄晓明", "杨颖");
        map.put("文章", "马伊琍");
        map.put("邓超", "孙俪");
        System.out.println(map);

        //String remove(String key)
        System.out.println(map.remove("邓超"));
        System.out.println(map);

        // 想要查看 黄晓明的媳妇 是谁
        System.out.println(map.get("黄晓明"));
        System.out.println(map.get("邓超"));
    }
}
```

tips:

使用put方法时，若指定的键(key)在集合中没有，则没有这个键对应的值，返回null，并把指定的键值添加到集合中；

若指定的键(key)在集合中存在，则返回值为集合中键对应的值（该值为替换前的值），并把指定键所对应的值，替换成指定的新值。

1.4 Map的遍历

方式1:键找值方式

通过元素中的键，获取键所对应的值

分析步骤：

1. 获取Map中所有的键，由于键是唯一的，所以返回一个Set集合存储所有的键。方法提示: `keySet()`
2. 遍历键的Set集合，得到每一个键。
3. 根据键，获取键所对应的值。方法提示: `get(K key)`

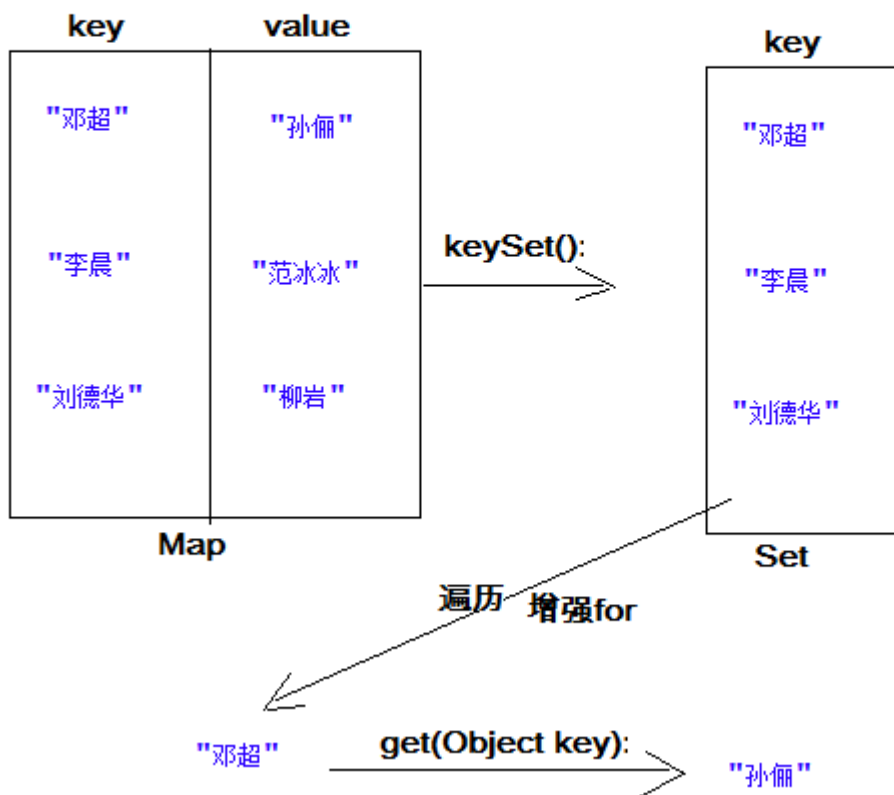
遍历图解：

Map集合遍历方式1：键找值

Map集合方法：

keySet(): 得到Map集合中所有的键

get(Object key): 通过指定的键，从map集合中找对应的值



方式2:键值对方式

即通过集合中每个键值对(Entry)对象，获取键值对(Entry)对象中的键与值。

Entry键值对对象:

我们已经知道，Map中存放的是两种对象，一种称为key(键)，一种称为value(值)，它们在Map中是一一对应关系，这一对对象又称做Map中的一个Entry(项)。Entry将键值对的对应关系封装成了对象。即键值对对象，这样我们在遍历Map集合时，就可以从每一个键值对(Entry)对象中获取对应的键与对应的值。

在Map集合中也提供了获取所有Entry对象的方法：

- `public Set<Map.Entry<K,V>> entrySet()`：获取到Map集合中所有的键值对对象的集合(Set集合)。

获取了Entry对象，表示获取了一对键和值，那么同样Entry中，分别提供了获取键和获取值的方法：

- `public K getKey()`：获取Entry对象中的键。
- `public V getValue()`：获取Entry对象中的值。

操作步骤与图解：

1. 获取Map集合中，所有的键值对(Entry)对象，以Set集合形式返回。方法提示：`entrySet()`。
2. 遍历包含键值对(Entry)对象的Set集合，得到每一个键值对(Entry)对象。

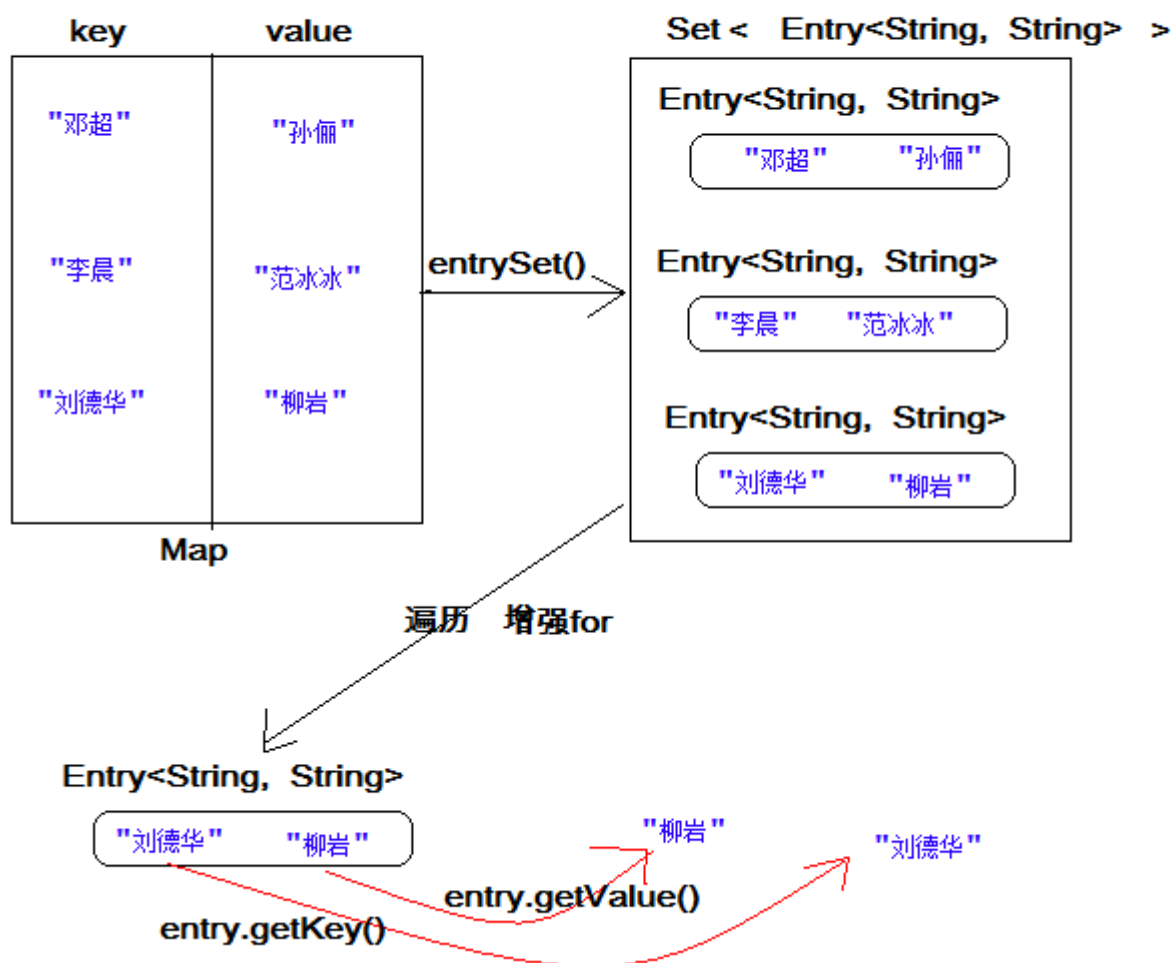
3. 通过键值对(Entry)对象，获取Entry对象中的键与值。 方法提示: `getKey()` `getValue()`

遍历图解:

Map集合遍历方式2：通过键值对，找键，找值的方式

Map集合方法：

`entrySet()`：得到一个包含多个键值对元素的Set集合



tips: Map集合不能直接使用迭代器或者foreach进行遍历。但是转成Set之后就可以使用了。

1.5 HashMap存储自定义类型

练习：每位学生（姓名，年龄）都有自己的家庭住址。那么，既然有对应关系，则将学生对象和家庭住址存储到map集合中。学生作为键，家庭住址作为值。

注意，学生姓名相同并且年龄相同视为同一名学生。

编写学生类：

```
public class Student {  
    private String name;  
    private int age;  
  
    //构造方法  
    //get/set
```



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

@Override
public int hashCode() {
    return Objects.hash(name, age);
}
}
```

编写测试类：

```
public class HashMapTest {
    public static void main(String[] args) {
        //1,创建HashMap集合对象。
        Map<Student,String> map = new HashMap<Student,String>();
        //2,添加元素。
        map.put(new Student("lisi",28), "上海");
        map.put(new Student("wangwu",22), "北京");
        map.put(new Student("wangwu",22), "南京");

        //3,取出元素。键找值方式
        Set<Student> keySet = map.keySet();
        for(Student key: keySet){
            String value = map.get(key);
            System.out.println(key.toString()+"....."+value);
        }
    }
}
```

- 当给HashMap中存放自定义对象时，如果自定义对象作为key存在，这时要保证对象唯一，必须复写对象的hashCode和equals方法(如果忘记，请回顾HashSet存放自定义对象)。
- 如果要保证map中存放的key和取出的顺序一致，可以使用 `java.util.LinkedHashMap` 集合来存放。

1.6 LinkedHashMap介绍

我们知道HashMap保证成对元素唯一，并且查询速度很快，可是成对元素存放进去是没有顺序的，那么我们要保证有序，还要速度快怎么办呢？

在HashMap下面有一个子类LinkedHashMap，它是链表和哈希表组合的一个数据存储结构。

```
public class LinkedHashMapDemo {
    public static void main(String[] args) {
        LinkedHashMap<String, String> map = new LinkedHashMap<String, String>();
        map.put("邓超", "孙俪");
        map.put("李晨", "范冰冰");
        map.put("刘德华", "朱丽倩");
        Set<Entry<String, String>> entrySet = map.entrySet();
        for (Entry<String, String> entry : entrySet) {
            System.out.println(entry.getKey() + " " + entry.getValue());
        }
    }
}
```

结果:

邓超 孙俪
李晨 范冰冰
刘德华 朱丽倩



1.7 Map集合练习



需求:

计算一个字符串中每个字符出现次数。

分析:

1. 获取一个字符串对象
2. 创建一个Map集合，键代表字符，值代表次数。
3. 遍历字符串得到每个字符。
4. 判断Map中是否有该键。
5. 如果没有，第一次出现，存储次数为1；如果有，则说明已经出现过，获取到对应的值进行++，再次存储。
6. 打印最终结果

方法介绍

`public boolean containKey(Object key)`:判断该集合中是否有此键。

代码:

```
public class MapTest {
    public static void main(String[] args) {
        //友情提示
        System.out.println("请录入一个字符串:");
        String line = new Scanner(System.in).nextLine();
        // 定义 每个字符出现次数的方法
        findChar(line);
    }
    private static void findChar(String line) {
        //1:创建一个集合 存储 字符 以及其出现的次数
        HashMap<Character, Integer> map = new HashMap<Character, Integer>();

        //2:遍历字符串
```



```
for (int i = 0; i < line.length(); i++) {  
    char c = line.charAt(i);  
    //判断 该字符 是否在键集中  
    if (!map.containsKey(c)) { //说明这个字符没有出现过  
        //那就是第一次  
        map.put(c, 1);  
    } else {  
        //先获取之前的次数  
        Integer count = map.get(c);  
        //count++;  
        //再次存入 更新  
        map.put(c, ++count);  
    }  
}  
System.out.println(map);  
}
```

第二章 补充知识点

2.1 可变参数

在JDK1.5之后，如果我们定义一个方法需要接受多个参数，并且多个参数类型一致，我们可以对其简化。

格式：

```
修饰符 返回值类型 方法名(参数类型... 形参名){ }
```

代码演示：

```
public class ChangeArgs {  
    public static void main(String[] args) {  
        int sum = getSum(6, 7, 2, 12, 2121);  
        System.out.println(sum);  
    }  
    public static int getSum(int... arr) {  
        int sum = 0;  
        for (int a : arr) {  
            sum += a;  
        }  
        return sum;  
    }  
}
```

注意：

1. 一个方法只能有一个可变参数
2. 如果方法中有多个参数，可变参数要放到最后。

应用场景: Collections

在Collections中也提供了添加一些元素方法：

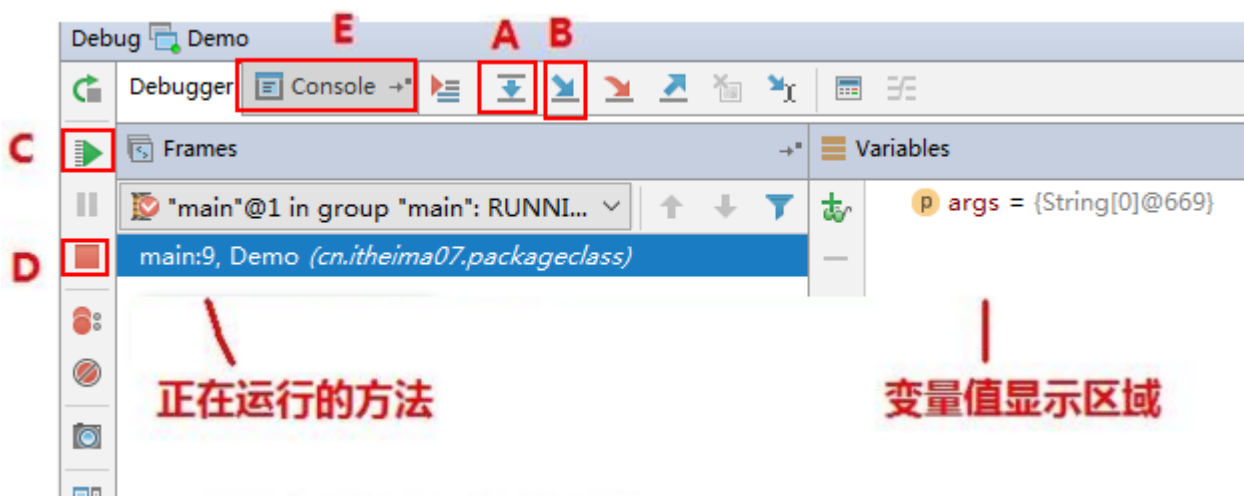
```
public static <T> boolean addAll(Collection<T> c, T... elements) :往集合中添加一些元素。
```

代码演示:

```
public class CollectionsDemo {  
    public static void main(String[] args) {  
        ArrayList<Integer> list = new ArrayList<Integer>();  
        //原来写法  
        //list.add(12);  
        //list.add(14);  
        //list.add(15);  
        //list.add(1000);  
        //采用工具类 完成 往集合中添加元素  
        Collections.addAll(list, 5, 222, 1, 2);  
        System.out.println(list);  
    }  
}
```

2.2 Debug追踪

使用IDEA的断点调试功能，查看程序的运行过程 Debug调试窗口介绍



A: 代码向下执行一行 快捷键 F8

B: 进入要调用的方法 快捷键 F7

C: 运行完所有程序 快捷键 F9

D: 停止Debug调试模式 快捷键 Ctrl+F2

E: 切换到控制台查看运行结果

第三章 模拟斗地主洗牌发牌



3.1 案例介绍

按照斗地主的规则，完成洗牌发牌的动作。

令狐冲: [♠2, ♠A, ♥A, ♠A, ♠K, ♥Q, ♦J, ♠J, ♥J, ♦9, ♠7, ♦5, ♥4, ♠4, ♠3, ♥3, ♠3]
石破天: [小王, ♦2, ♠2, ♥2, ♠A, ♦K, ♠Q, ♦10, ♥10, ♠10, ♠8, ♠6, ♥6, ♠5, ♠5, ♦4, ♠4]
鸠摩智: [大王, ♥K, ♦Q, ♠Q, ♠10, ♥9, ♠9, ♦8, ♠8, ♥8, ♦7, ♥7, ♠7, ♦6, ♠6, ♥5, ♦3]
底牌: [♠K, ♠J, ♠9]

具体规则:

1. 组装54张扑克牌将
2. 54张牌顺序打乱
3. 三个玩家参与游戏，三人交替摸牌，每人17张牌，最后三张留作底牌。
4. 查看三人各自手中的牌（按照牌的大小排序）、底牌

规则：手中扑克牌从大到小的摆放顺序：大王,小王,2,A,K,Q,J,10,9,8,7,6,5,4,3

3.2 案例需求分析

1. 准备牌:

完成数字与纸牌的映射关系:

使用双列Map(HashMap)集合，完成一个数字与字符串纸牌的对应关系(相当于一个字典)。

2. 洗牌:

通过数字完成洗牌发牌

3. 发牌:

将每个人以及底牌设计为ArrayList,将最后3张牌直接存放于底牌，剩余牌通过对3取模依次发牌。

存放的过程中要求数字大小与斗地主规则的大小对应。

将代表不同纸牌的数字分配给不同的玩家与底牌。

4. 看牌:

通过Map集合找到对应字符展示。

通过查询纸牌与数字的对应关系，由数字转成纸牌字符串再进行展示。

- 准备牌:
完成数字与纸牌的映射关系:
使用双列 Map(HashMap)集合，完成一个数字与字符串纸牌的对应关系(相当于一个字典)。
LinkedHashMap<Integer, String> **值为扑克牌** **键为牌编号**
- 洗牌:
ArrayList<Integer> 记录54个牌的编号
通过数字完成洗牌发牌 **Collections.shuffle(List list)**
- 发牌:
将每个人以及底牌设计为 ArrayList<String>,将最后 3 张牌直接存放于底牌，剩余牌通过对 3 取模依次发牌。 **发牌：发的是牌的编号**
存放的过程中要求数字大小与斗地主规则的大小对应。
将代表不同纸牌的数字分配给不同的玩家与底牌。
- 看牌:
通过 Map 集合找到对应字符展示。**通过牌的编号，去Map集合中，查询对应编号的扑克牌**
通过查询纸牌与数字的对应关系，由数字转成纸牌字符串再进行展示。
把查询到的扑克牌 存储到 ArrayList< String >

```
{0=大王, 1=小王,
2=♥2, 3=♠2, 4=♦2, 5=♣2,
6=♥A, 7=♠A, 8=♦A, 9=♣A,
10=♥K, 11=♠K, 12=♦K, 13=♣K,
14=♥Q, 15=♠Q, 16=♦Q, 17=♣Q,
18=♥J, 19=♠J, 20=♦J, 21=♣J,
22=♥10, 23=♠10, 24=♦10, 25=♣10,
26=♥9, 27=♠9, 28=♦9, 29=♣9,
30=♥8, 31=♠8, 32=♦8, 33=♣8,
34=♥7, 35=♠7, 36=♦7, 37=♣7,
38=♥6, 39=♠6, 40=♦6, 41=♣6,
42=♥5, 43=♠5, 44=♦5, 45=♣5,
46=♥4, 47=♠4, 48=♦4, 49=♣4,
50=♥3, 51=♠3, 52=♦3, 53=♣3}
```



3.3 实现代码步骤

```
package com.itheima04;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;

/*
 * 组合牌
 * 定义一个Map集合用来存储牌号 和 牌
 * 定义一个List集合用来存储牌号
 * 花色:♥-♠-♦-♣
 * 数字:2-A-K-Q-J-10-9-8-7-6-5-4-3
 * 洗牌
 * Collections.shuffle(牌号集合)
 * 发牌
 * 三个玩家三个集合
 * 发牌号
 * 排序
 * 看牌
 */
public class Pooker {

    public static void main(String[] args) {
        // 定义一个Map集合用来存储牌号 和 牌
        HashMap<Integer, String> pookerMap = new HashMap<Integer, String>();
        //定义一个List集合用来存储牌号
        ArrayList<Integer> pookerList = new ArrayList<Integer>();

        String[] colors = "♥-♠-♦-♣".split("-");
        String[] nums = "2-A-K-Q-J-10-9-8-7-6-5-4-3".split("-");

        int index = 2;
        for(String num : nums){
            for(String color : colors){
                String thisPooker = color+num;
                System.out.println(thisPooker);
                //将扑克牌放入Map集合
                pookerMap.put(index, thisPooker);
                //将牌号放入到pookerList集合中
                pookerList.add(index);
                index++;
            }
        }

        //将大王小王添加到集合
        pookerMap.put(0, "大王");
        pookerMap.put(1, "小王");

        pookerList.add(0);
    }
}
```



```
pokerList.add(1);

//      System.out.println(pokerMap);
//      System.out.println(pokerList);

//洗牌
Collections.shuffle(pokerList);

//发牌
ArrayList<Integer> player1 = new ArrayList<Integer>();
ArrayList<Integer> player2 = new ArrayList<Integer>();
ArrayList<Integer> player3 = new ArrayList<Integer>();
ArrayList<Integer> diPai = new ArrayList<Integer>();

//遍历牌号的集合 判断索引发牌号
for(int i = 0 ;i < pokerList.size() ;i++){
    Integer pokerNum = pokerList.get(i);

    if(i>=51)
        diPai.add(pokerNum);
    }else if(i % 3 == 0){
        player1.add(pokerNum);
    }else if(i % 3 == 1){
        player2.add(pokerNum);
    }else if(i % 3 == 2){
        player3.add(pokerNum);
    }
}

//      排序

Collections.sort(player1);
Collections.sort(player2);
Collections.sort(player3);
Collections.sort(diPai);
//      System.out.println(player1);
//      System.out.println(player2);
//      System.out.println(player3);
//      System.out.println(diPai);

show("柳岩",player1,pokerMap);
show("唐嫣",player2,pokerMap);
show("金莲",player3,pokerMap);
show("底牌",diPai,pokerMap);

}
//定义方法 看牌
public static void show(String name,ArrayList<Integer> player,HashMap<Integer, String>
pokerMap ){
    System.out.print(name+":");
    for(Integer pokerNum : player){

        String thisPoker = pokerMap.get(pokerNum);
```

```
        System.out.print(thisPooker+" ");  
    }  
    System.out.println();  
}  
}
```