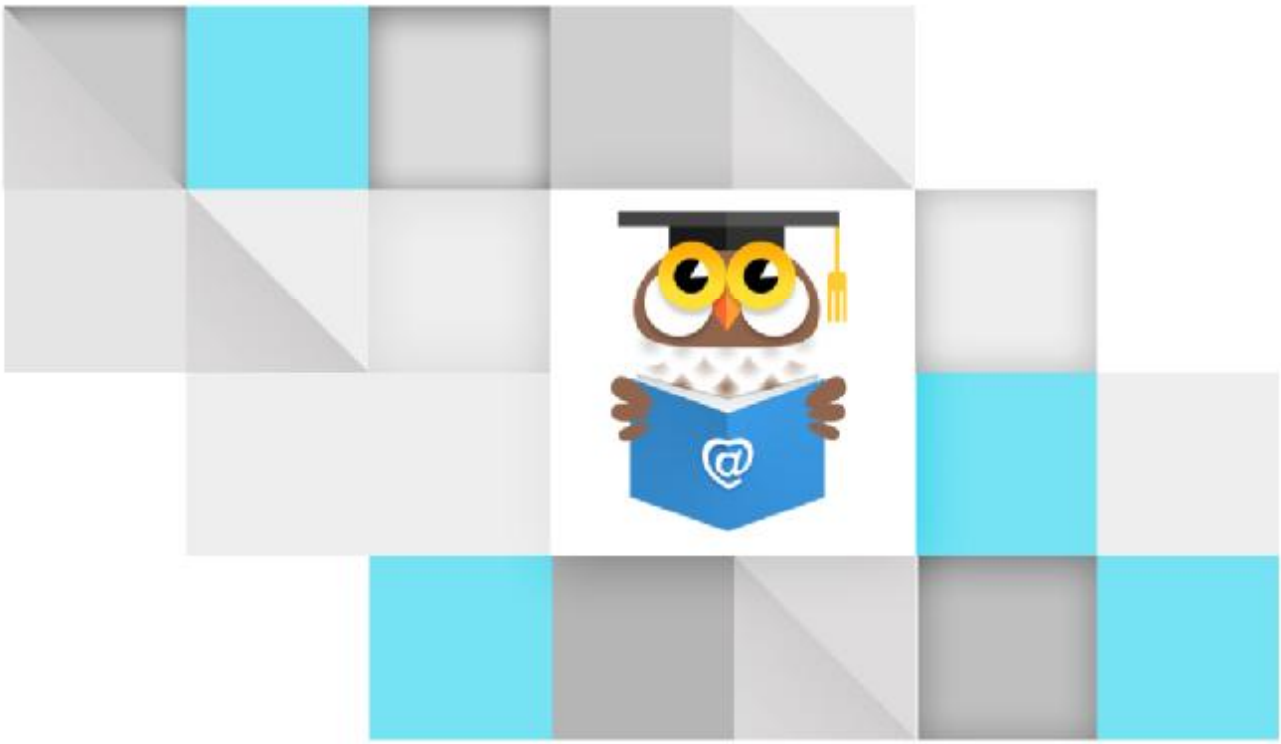
消灭泡泡糖（Java）

实训指导手册

实训场景 012 – 移动垂直方向的泡泡糖（三）



Campus Solution Group

目 录

一、任务编号：PRJ-BU2-JAVA-012

1、实训技能

- Java API 运用技能

2、涉及知识点

- Map 中的方法
- Map 的应用
- 泛型类和泛型接口
- 使用迭代器

3、实现效果

图 3-1

4、场景说明

1、业务说明：

本场景主要用于实现游戏界面上（多列）泡泡糖的消除与垂直移动“填补空隙”的功能，移动的顺序必须满足从左向右，从上向下。

本场景在**PRJ-BU2-JAVA-010**场景的基础上实现了（多列）待移动泡泡糖个数与移动步长自动计算的效果，并确保满足移动顺序。

2、实现思路：

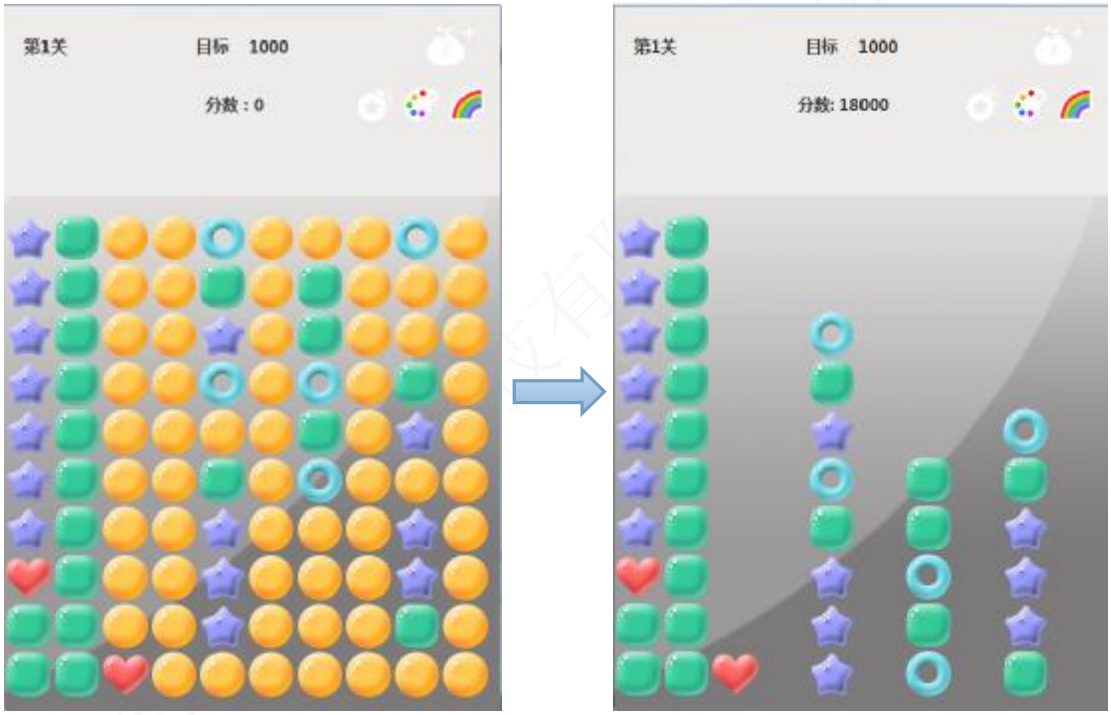


图 **4-1**

通过图**4-1**可知，为确保界面正确实现泡泡糖垂直移动的功能，需要提供 **“垂直方向待移动泡泡糖”**。如果把多列泡泡糖理解为**N**个单列泡泡糖，我们就可以循环利用**PRJ-BU2-JAVA-010**场景中获取单列**“垂直方向待移动泡泡糖”**的方法，获取多列**“垂直方向待移动泡泡糖”**。

2-1. 首先，为保证移动顺序，我们需要将“**待消除的泡泡糖”**按（列）进行排序。

2-2. 然后，将完整的“**待消除的泡泡糖”集合**按（列）进行分组产生多列泡泡对象。

2-3. 最后，依次封装每一列**“垂直方向待移动泡泡糖”**。

3、核心组件介绍：

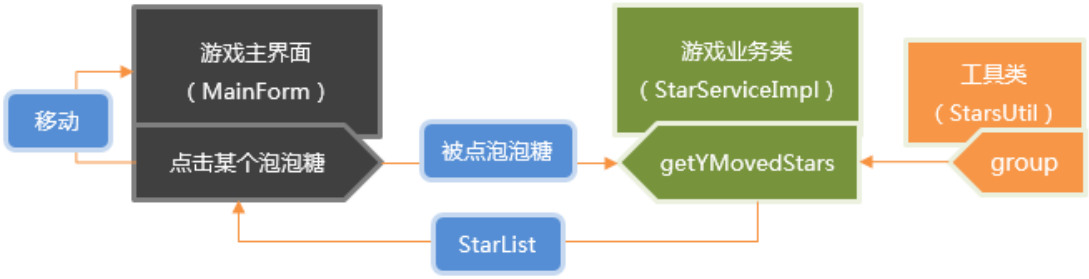


图 **4-2**

3-1. MainForm - 游戏界面类（本场景无需实现）：

负责游戏数据显示、响应用户在界面上的各类操作。

3-2. StarServiceImpl - 游戏业务类

负责游戏相关逻辑计算，例如：泡泡糖移动、消除、分数计算等操作。

3-3. StarList - 用于保存待移动泡泡糖（**MovedStar**）集合。

3-4. StarServiceImpl中的getYMovedStars：

该方法主要负责计算**“垂直方向待移动泡泡糖”**的数量和移动步长，并保存在**StarList**中。回顾场景**PRJ-BU2-JAVA-010**发现，我们已经实现了单列泡泡糖垂直方向移动，在本场景中我们将会实现多列泡泡糖垂直方向的移动，移动顺序从左向右，从上向下。

步骤一、将**“待消除泡泡糖”**按（列）进行排序，以确保界面按从左向右，从上向下的顺序移动泡泡糖。

步骤二、将完整的“**待消除的泡泡糖”集合**按（列）进行分组，这样原本挤压在一个集合中的所有“**待消除的泡泡糖”**将会按（列）组成多个“待消除的泡泡糖”集合，每个集合中的**“待消除的泡泡糖”**仅行号不同，列号均相同。

步骤三、依次获取每一列的**“垂直方向待移动泡泡糖”**（获取单列**“垂直方向待移动**

泡泡糖”功能已在场景**010**中实现）。

3-5. StarsUtil - 工具类

提供一些简单的操作函数，如排序，克隆，类型转换等。

4、了解更多：

请参考《消灭泡泡糖 **-** 需求说明文档》

5、前置条件：

5-1. 前置场景：**PRJ-BU2-JAVA-010** – 移动垂直方向的泡泡糖（一）

5-2. 必备知识与技能：

5-1.1. Java开发工具 （**Eclipse**）。

5-1.2. Java面向对象编程技能（**while**循环块，**if**条件块，类的成员方法）。

5、快速开始

1、 开发环境：

1-1. Oracle JDK8.x 以上版本

1-2. Eclipse Luna（**4.4.x**）以上版本

1-3. 工程包：**PRJ_BU2_JAVA_012**

2、 进入开发环境：

详见**SPOC**平台上《**PRJ-BU2-JAVA-012** 前置任务：进入开发环境》

图 **5-1**

6、任务 1 – 更新集合的排序算法

1、任务描述：

场景**010**在**StarsUtil**类中实现的**sort**函数，是针对单列**“待消除泡泡糖”**的排序，排序要求按照（行值）升序排序；本场景业务从原本单列变成多列，新的排序的规则如下：

1-1. 排序时，要求按（列值）升序排列。

1-2. 排序时，如相邻泡泡糖的（列值）相同，则按（行值）升序排序。

以上排序方法将确保后续任务实现的“待移动泡泡糖”能按从左向右 ,从上向下的顺序移动（业务流程详见下图）：

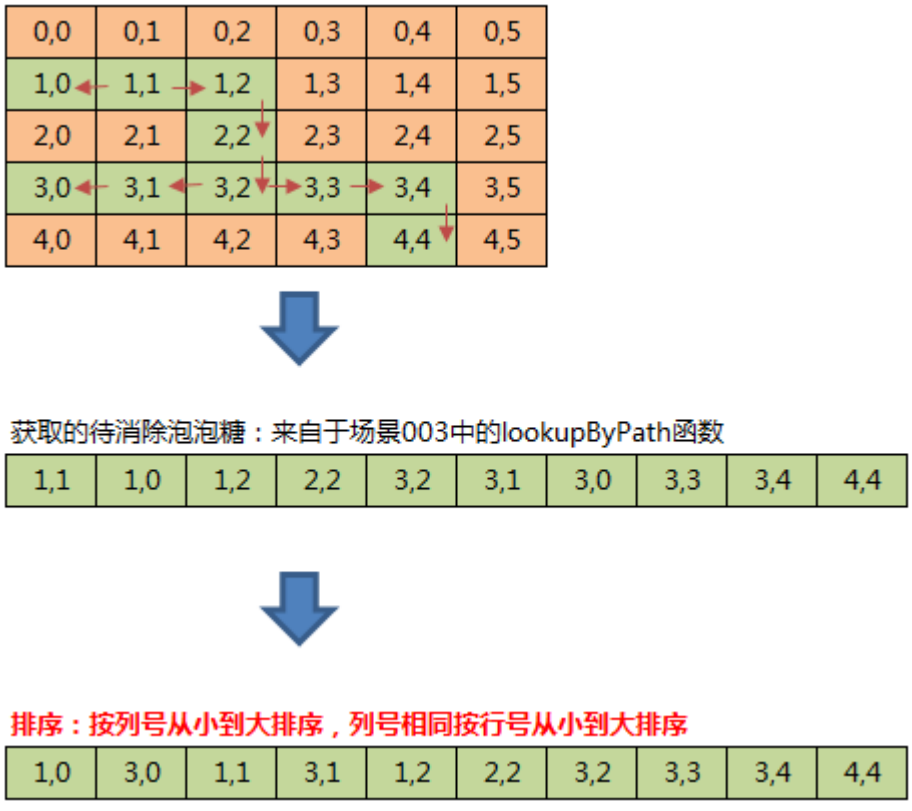


图 **6-1**

2、推荐步骤：

2-1. 场景定位

2-1.1. 定位到：**cn.campsg.practical.bubble.util.StarsUtil**类

2-1.2. 找到**StarsUtil**类中的**sort**函数

2-2. 更新排序的对比条件

2-2.1. 将冒泡排序中原本通过（行值）对比进行排序的代码，修改为通过（列值）对比进行排序。

2-3. 在按（列值）进行对比判断的代码后，添加（列值）相等的对比代码：

2-3.1. 如果两个泡泡糖的列值相等，则继续以下判断：

2-3.2. 如果前一个泡泡糖的（行值）大于后一个泡泡糖的（行值），则交换两个泡泡糖的数据。

2-4. 添加循环中断条件

2-4.1. 当前一个泡泡糖（列值）大于后一个泡泡糖（列值）时，由于已经不可能再发生两个泡泡糖列值相等的情况，因此直接中断后续判断，重新进入循环。

+提示：

中断当前循环，进入下一次循环可以使用**continue**关键字。

3、验证与测试：

3-1. 创建程序入口函数**-main**

3-2. 编写以下测试语句：

3-2.1. 创建**StarList**集合实例

3-2.2. 依次添加五个不同位置的泡泡糖对象：

1）【2,3】- BLUE

2）【1,5】- GREEN

3）【0,9】- PURPLE

4）【0,3】- RED

5）【0,8】- YELLOW

3-2.3. 为泡泡糖集合进行排序

3-3. 打印测试结果：

3-3.1. 在**sort**函数调用前后，分别打印出泡泡糖集合

3-3.2. 输出结果与下图一致：

```
排序前, starList:
(2,3-BLUE)      ,(1,5-GREEN)    ,(0,9-PURPLE)   ,(0,3-RED)      ,(0,8-YELLOW)
排序后, starList:
(0,3-RED)       ,(2,3-BLUE)     ,(1,5-GREEN)    ,(0,8-YELLOW)   ,(0,9-PURPLE)
```

图 **6-2**

7、任务 2 – 待消除泡泡糖的排序与分组

1、任务描述：

为确保界面能够按照从左向右，从上到下的顺序移动所有“待移动的泡泡糖”，我们需要对已经排序完毕的“待移动泡泡糖”进行分组，分组后原本挤压在一个集合中的所有“**待消除的泡泡糖”**将会被按（列）组成多个“待消除的泡泡糖”集合，每个集合中的**“待消除的泡泡糖”**仅行号不同，列号均相同（详见下图）。

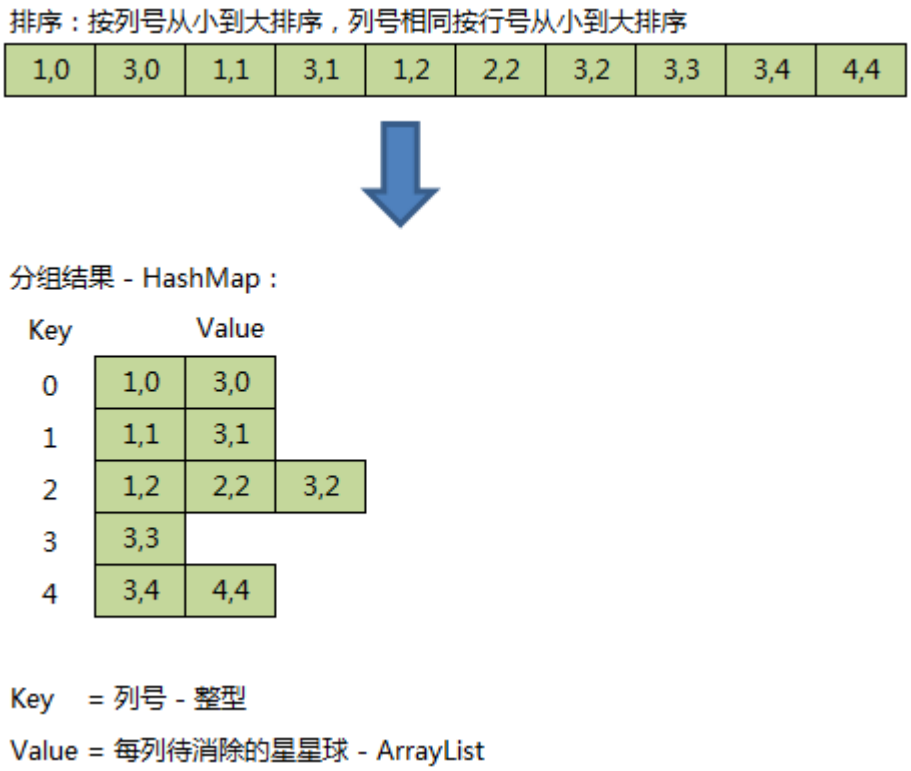


图 **7-1**

按图**7-1**分解“待消除泡泡糖”的目的是：确保组织**“待移动的泡泡糖”**集合时，也将会按从左向右，从上到下的顺序进行排列。

2、推荐步骤：

2-1. 场景定位

2-1.1. 定位到：**cn.campsg.practical.bubble.util.StarsUtil**类

2-2. 创建公共静态函数 **- group**

2-2.1. 函数参数：待消除的泡泡糖集合

2-2.2. 返回类型：**Map**类型，泛型对应**Key**为整型，**Value**为泡泡糖集合类型

+ 提示：

泛型用来规范集合可存放的数据类型，泛型可以是任意数据类型，使用方式如下：

例如**1**：**List<String> lst = new ArrayList<String>();**

例如2：**Map<Integer, StarList> map = new HashMap< Integer, StarList>();**

2-3. 创建满足返回类型的**HashMap**对象实例

2-4. 对待消除泡泡糖集合进行排序

2-5. 创建循环遍历“待消除泡泡糖”集合。

2-5.1. 通过循环变量从“待消除泡泡糖”集合中获取一个泡泡糖对象。

2-5.2. 判断泡泡糖的【列值】是否已存在于**HashMap**中（判断**Key**值）

1） 如果存在，则将当前【列值】对应的**value**（类型为**StarList**）从**HashMap**中取出，并将泡泡糖加入到该**StarList**之中。

2） 如果不存在，则创建新的**StarList**集合，将泡泡糖加入新的**StarList**集合中，以【列值】为**Key**，新**StarList**集合为**Value**，存入**HashMap**中。

2-6. 遍历结束后，返回**HashMap**对象。

3、验证与测试：

3-1. 找到任务**1**中的**main**函数 。

3-2. 删除原本的打印语句与**sort**函数。

3-3. 在删除代码处，调用**group**函数对集合进行分组，并接收**group**的返回。

3-4. 打印测试结果：

3-4.1. 打印出**group**函数返回的对象。

3-4.2. 运行**main**函数，输出结果与下图一致：

```
分组后:{3=
(0,3-RED)          ,(2,3-BLUE)       , 5=
(1,5-GREEN)        , 8=
(0,8-YELLOW)       , 9=
(0,9-PURPLE)       }
```

图 **7-2**

8、任务 3 – 获取垂直方向待移动泡泡糖

1、任务描述：

场景**PRJ-BU2-JAVA-010**实现了获取单列**“垂直方向待移动泡泡糖”**的集合；本场景主要是在此基础之上，实现获取多列**“垂直方向待移动泡泡糖”**的功能；步骤如下：

1-1. 首先，我们需要将“**待消除的泡泡糖”**按列进行**排序**与**分组。**

1-2. 然后，按**Key**（列值）遍历分组后的**Map**集合数据，依次获得每一列**“待消除的泡泡糖”**集合对象。

1-3. 最后，遍历当前列所有泡泡糖，将需要移动的泡泡糖封装成**MovedStar**，并将**MovedStar**保存于**“垂直方向待移动泡泡糖”**集合中即可。

其中**1-3**步，封装单列**“垂直方向待移动泡泡糖”**在场景**PRJ-BU2-JAVA-010**中已实现。

2、推荐步骤：

2-1. 复制场景**PRJ_BU2_JAVA_010**已完成的代码

2-1.1. 定位**PRJ_BU2_JAVA_010**工程中**StarServiceImpl**类中的**getYMovedStars**函数

2-1.2. 将函数中的代码复制到**PRJ_BU2_JAVA_012**对应的**getYMovedStars**函数中。

2-2. 获取待消除泡泡糖集合的分组对象。

2-2.1. 在判断多列泡泡糖移动时，不能简单的调用排序函数（仅适合单例），因此请将**getYMovedStars**函数中**StarsUtil.sort**函数删除。

2-2.2. 在原调用**sort**的函数处，利用任务**2**实现的**group**函数对“待消除泡泡糖”集合进行分组，并获取返回的分组对象。

2-3. 获取**group**返回的【**Map**】对象中**Key**的迭代器【**Iterator**】

2-4. 创建循环来遍历迭代器，整个循环应包括场景**PRJ-BU2-JAVA-010**所编写的所有与获取单列“待移动泡泡糖”相关的代码。

+ 提示：

1） 获取**Map**中所有的**Key**：**map**集合**.keySet()**

2） 获取**Map**的**Key**迭代器：**map**集合**.keySet().iterator()**

3） 推荐使用**while**遍历迭代器：

循环条件：**iterator.hasNext**；获取当前**key**值：**iterator.next**

2-5. 更新列值变量

2-5.1. 将代表当前列的变量初始值，修改为迭代器中动态获取的**key**值。

+ 业务说明：

1） 场景**PRJ_BU2_JAVA_010**将当前列值设定为**0**，只适合单列泡泡糖的情况。

2） 本场景将当前列设置为循环变量，可确保每列泡泡糖都获得判断。

2-6. 更新单列起始遍历位置的变量

2-6.1. 通过**2-5.1**获取的**key**值，获取**Map**中对应的**value**（**StarList**），此集合为当前列的待消除的泡泡糖集合。

2-6.2. 将判断“待移动泡泡糖”的起始位置变量修改为：循环获取的每一列“待清除泡泡糖”集合的最后一个元素的行值。

+ 业务说明：

1） 场景**PRJ_BU2_JAVA_010**的“待清除泡泡糖”集合始终为**1**列，因此判断“待移动泡泡糖”的起始位置为“待清除泡泡糖”集合最后一位成员的行值。

2）本场景“待清除泡泡糖”包含多列，之前的步骤通过循环依次获取了多个单列“待清除泡泡糖”集合，因此判断“待移动泡泡糖”的起始位置为每列“待清除泡泡糖”集合的最后一个元素的行值。

3、验证与测试：

3-1. 运行项目工程，选择启动函数：**cn.campsg.practical.bubble.MainClass**

3-2. 点击界面上任意泡泡糖，实现消除同色泡泡糖、移动垂直方向泡泡糖的功能。

3-3. 输出结果与下图一致：

图 **8-1**

9、场景总结

Q1. 在您的项目中如何使用 HashMap 对象？结合项目说说 HashMap 对象的优势。

1. Java 集合框架隶属于 **java.util** 包，其中 **HashMap** 非常常用。

2. HashMap 存放数据时不考虑数据的位置与顺序，以 **Key** 作为集合成员的“识别码”。

3. HashMap 通过 **put** 方法存放数据，**get** 方法按 **key** 获取数据，**key** 重复时 **value** 会被覆盖。

4. 当前游戏中对"待清除泡泡糖"进行分组，是获取"待移动泡泡糖"的核心步骤。分组需要按列号作为分组条件，将相同列号的"泡泡糖"归类在一起，由于列号不会重复，每

列都有可能对应一个 "泡泡糖"集合，因此 **HashMap** 就是一个很好的集合选择（**key**：列号，**value**： "泡泡糖"集合）。

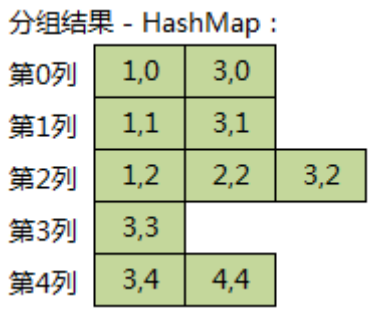


图 **9-1**

Q2. 如何对 HashMap 执行循环操作？。

1. HashMap 的循环操作与 **ArrayList** 有很大的区别：

1-1. ArrayList 由于按索引号存储数据，所以可以通过 **for** 循环按索引号依次获取数据。

1-2. HashMap 存放数据时不考虑顺序，自然也不存在索引的概念。

1-3. HashMap 可以通过 **Iterator** 迭代器实现循环。

1-4. HashMap 有三种获取迭代器的方法：

1） **map.values().iterator()**; 获取 **HashMap** 的 **value** 迭代器。

2） **map.keySet(). iterator()**; 获取 **HashMap** 的 **key** 迭代器。

3） **map.entrySet(). iterator()**; 获取 **HashMap** 的 **key-value** 的联合迭代器

Q3. 谈谈集合泛型的作用：

1. 由于 Java 集合中的 ArrayList 和 HashMap 都允许存放任意数据类型的数据，因此为集合配套泛型可以规范集合数据类型，保证集合中的数据类型一致（实际项目不可能允许集合中的数据类型不一致）。

2. 泛型有语法约束性，强制限定数据类型的完整性。

扩展 1. 泛型不仅仅可以在集合中使用，您自己编写的类也可以配套泛型，例如：

```
public class ExcelManager<S> {

    public void saveDataToSheet(S datasources) {}

}

ExcelManager<ArrayList<String>> manager =

new ExcelManager<ArrayList<String>>();

ExcelManager<DataBean> manager = new ExcelManager<DataBean>();
```

如需要了解更多泛型知识，可以查看本场景配套的泛型知识点。