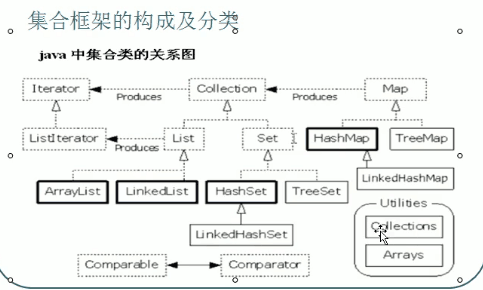
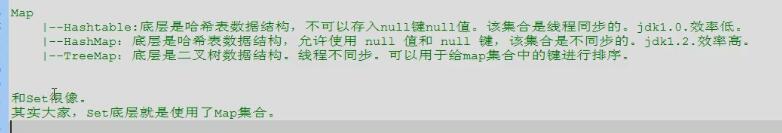
**P196 集合（Map概述）**

很像python里面的字典



**P197 集合（Map子类对象特点）**



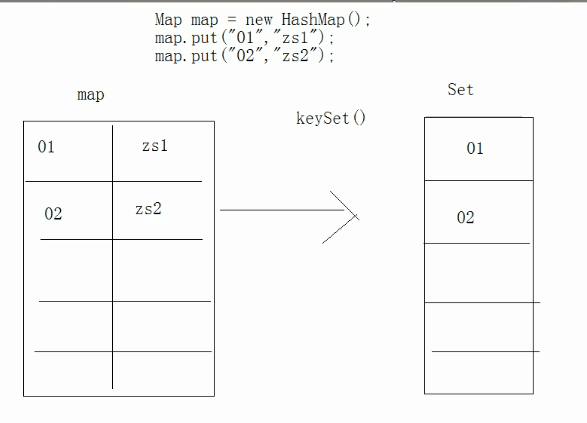
Map中的键必须是唯一的，所以必须有hashCode和equals方法，以判断唯一性。

**P198 集合（Map共性方法）**

//

**P199 集合（Map-keySet）**

--keySet方法在内存中的实现



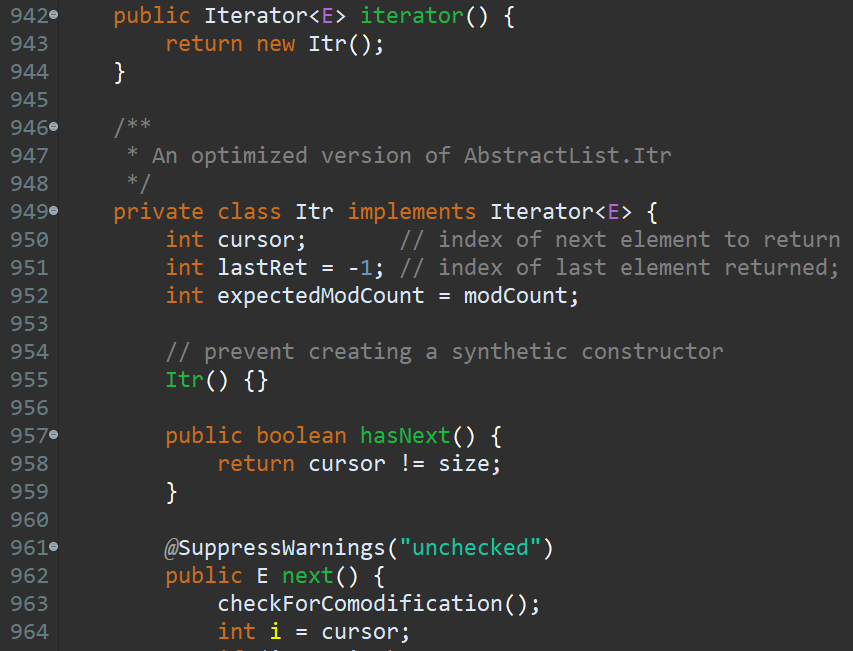
我们在对内存上开辟了map的空间，存入了一系列键值对。键和值的存放关系是这样的。然后我们通过keySet方法，取出了其中的key部份，数据结构是Set。

Map集合的keySet方法取出原理（其实一种数据结构，最重要的无非是弄清楚如何实现添加、删除、修改、查找。取出就是一种查找）：将map集合转成set集合，再通过迭代器取出。

**P200 集合（Map-entrySet）**

跟iterator相似，它也是封装了一系列方法的内部类，通过entrySet方法暴露了出来

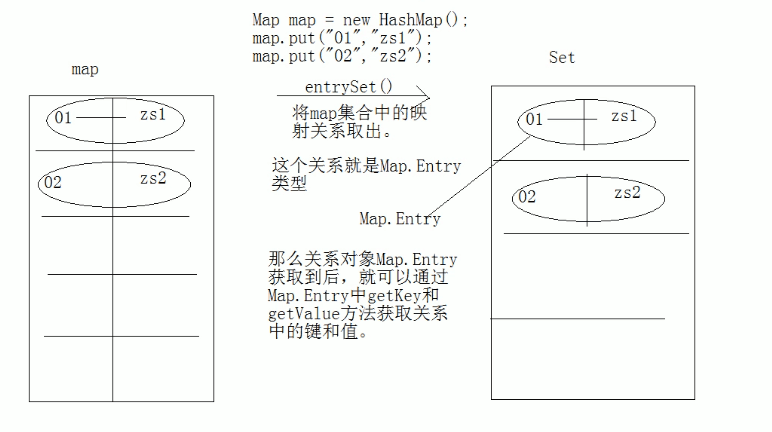
来看下源码



可以看到，再ArrayList中，将hasNext和next等方法进行了封装，形成了一个内部类。然后定义了iterator方法，该方法的功能就是将这个内部类传递出去。这也意味着，其实没有这个内部类，ArrayList也能实现hasNext等一系列方法，看似多此一举的用内部类进行封装，应该就是加强了内部方法之间的耦合，减弱了内部与外部方法之间的耦合。

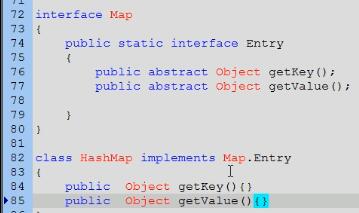
--entrySet方法在内存中的实现

每个圈都是Map.Entry类型的对象。



用夫妻类比：“01”那列是夫类型的变量，“zs1”那列是妻类型的变量，夫妻之前的映射就是结婚证书，是Map.Entry类型的数据。entrySet方法就像是获得了所有的结婚证书，此时通过结婚证书获得夫：翻到第一面，也就是调用getKey；获得妻：翻到第二面，也就是调用getValue

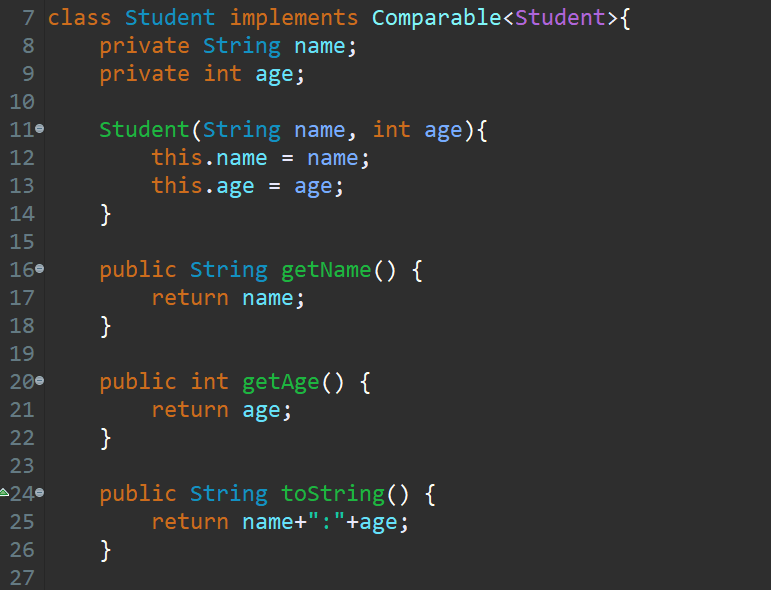
问题：Map.Entry到底是什么东西？

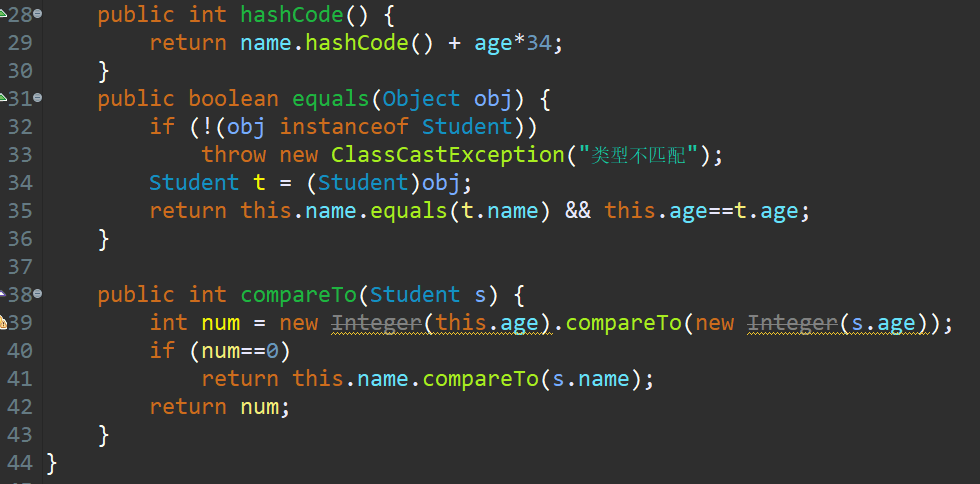


定义在Map接口中的内部接口。为什么定义成内部接口：1.Entry代表映射关系，现有键值对，才有映射关系。2.定义在内部可以直接使用外部的东西。

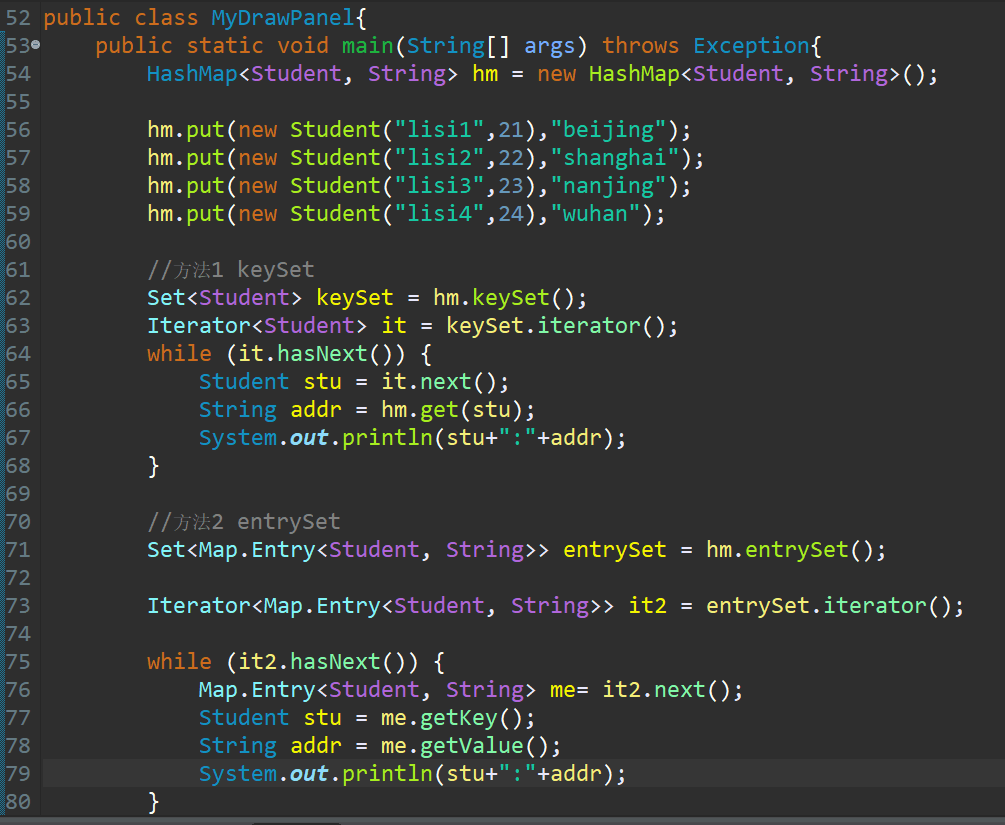
但是在实际代码中，Map.Entry是由HashMap的一个内部类来实现的。可以看作类类型。

**P201 集合（Map练习）**





以往一个简单的自定义类，只要写到27行就完成了基本功能。但是考虑到需求：我或者用户会创建很对该类的对象，对象一多，有可能把这些对象统一到HashSet，那么，我们就需要重写hashCode和equals方法；也有可能统一存放到TreeSet，就需要Comparable接口并实现compareTo方法。

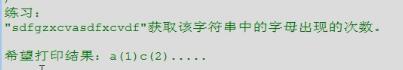


**P202集合（TreeMap练习）**

//

**P203集合（TreeMap练习-字母出现的次数）**

需求

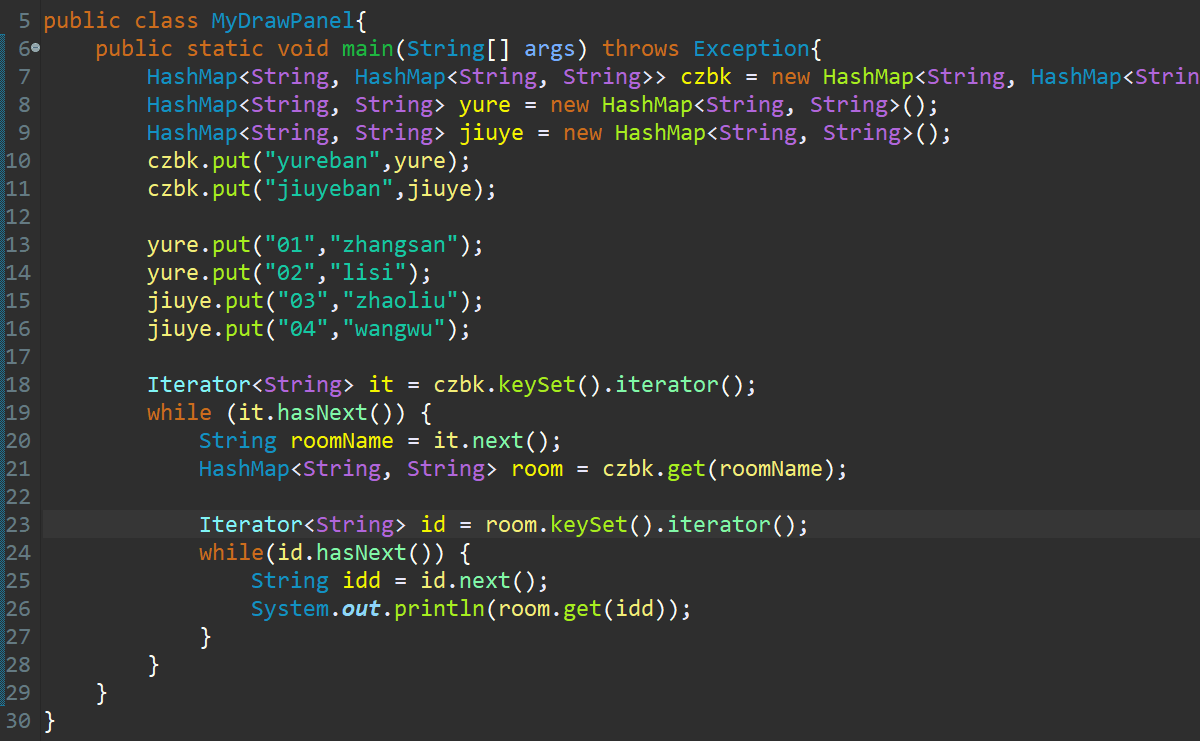


我们发现字母和次数是类似于键值对。所以使用Map。TreeMap和HashMap都行。思考这个TreeMap的泛型，应该是char和int，但是泛型中必须是引用数据类型，传入基本数据类型会报错，所以应该是Character和Integer。



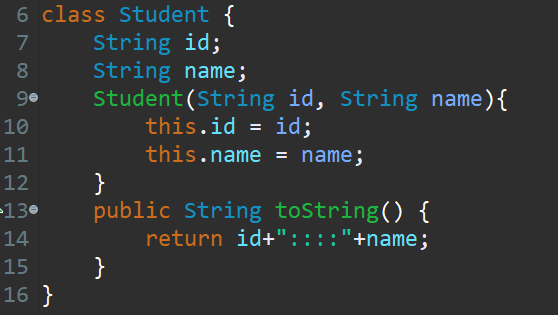
**P204集合（Map扩展）**

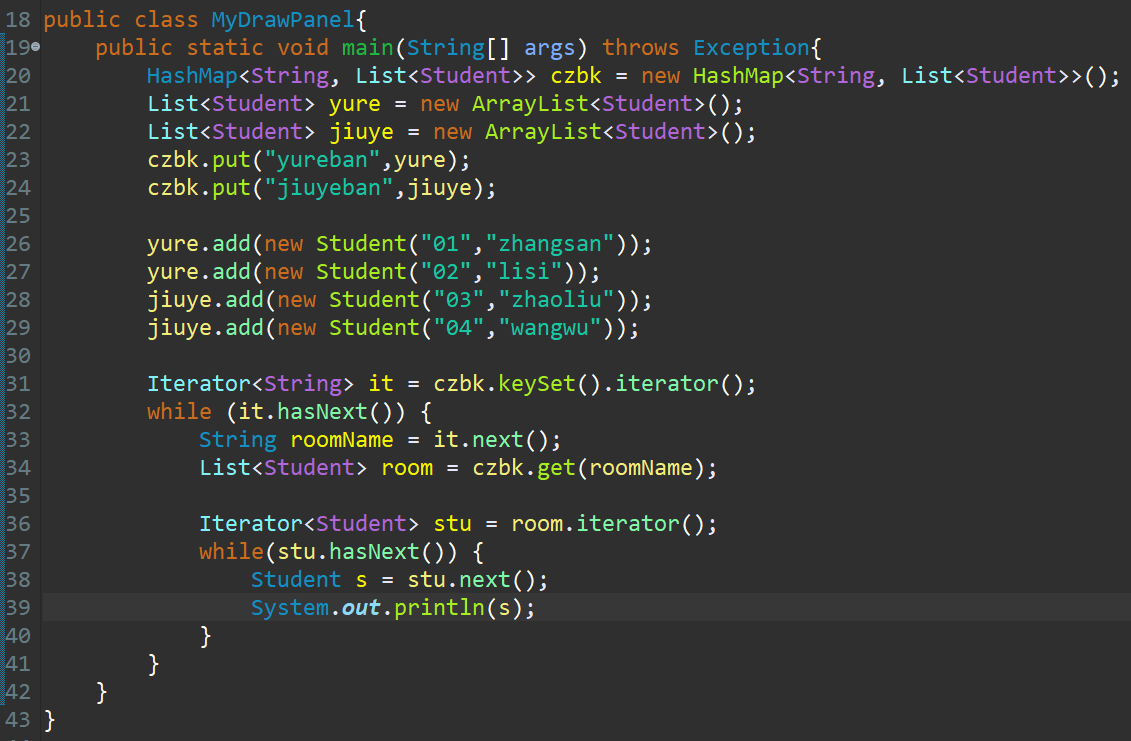
有多个映射关系怎么办？比如一个学生有班级和学号。



但是很多时候，学号和姓名是被封装成一个Student对象。这时外面的大集合的引用数据类型改成HashMap<String, HashMap<String,String>> 🡪

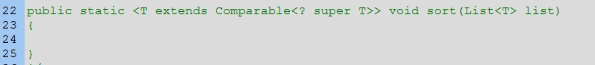
HashMap<String, List<Student>>





**P205集合（Collections-sort）**

Collections的用途：有一堆元素，我不需要保证唯一性，但是我又要对它们进行排序。那就使用Collections中的sort方法。



理解22行的泛型：

T extends Comparable保证传入的必须是Comparable1的子类，那么就具有比较性，这样如果传入的元素没有比较性，错误就会产生在编译时期而不是运行时期。

**P206集合（Collections-max）**

//

**P207集合（Collections-binarySearch）**

实际上就是基于二分法的查找

**P208集合（Collections-替换反转）**

//

**P209集合（Collections-reverseOrder）**

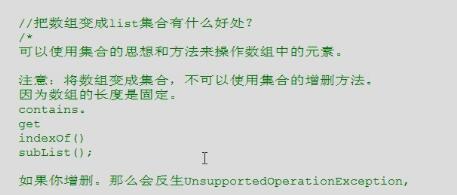
//

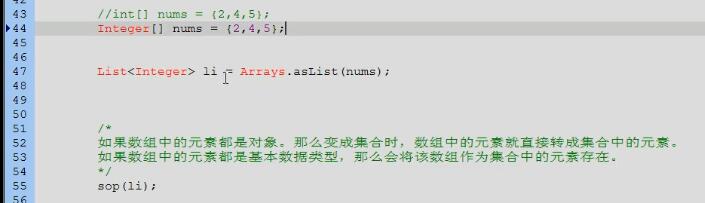
**P210集合（Collections-SynList）**

//

**P211集合（Arrays）**

Arrays.asList()方法，传入一个数组，返回一个集合。





比较43行和44行的差异

43：集合将nums整个作为一个元素。47行泛型：List<int[]>

44：集合将2，4，5作为3个元素。47行泛型：List<Integer>

**P212集合（集合转数组）**

为什么要将集合变成数组？

为了限定对元素的操作，不需要增删了。

**P213集合（增强for循环）**

Map里面没有iterator()方法，只有collection里面有。前面之所以有对Map类型的遍历操作，是用entrySet或keySet转成collection了。

**P214集合（可变参数）**

需求：定义一个方法，接受2-10个参数，返回它们的输出。

方法1：

用重载的方式定义9个相同的方法，分别接受2个，3个，。。。10个。

方法2：

让该方法接受一个数组。这样只要定义一个方法就可。

-public static void show (int[] arr){}

但是，如果我想传入两个参数2，3

就必须先把它们封装成数组：

-int[] arr = new int[2,3];

-show(arr);

方法3：

-public static void show (int… arr){}

直接传入即可：show(2,3);

调用看起来是方法1，实际上是方法2的简化。函数接受的仍然是个数组。

优点是不用每一次偶读手动的建立数组对象。只要将要操作的元素作为参数传递即可。隐式将这些参数封装成了数组。

注意事项：可变参数要定义在参数列表的最后面：

错误：public void show(int… arr, String str){}

正确：public void show(String str, int… arr){}

**P215集合（静态导入）**

//

**P216其他对象（System）**

//

**P217其他对象（Runtime）**

查阅Runtime的API我们发现一个有趣的现象：Runtime没有构造函数，方法又都是非静态的。所以我们推测出：一定有一个静态方法，其返回值是Runtime类型。这让我们回想起单例设计模式。

**P218其他对象（Date）**

//

**P219其他对象（Calender）**

//

**P220其他对象（Math-Random）**

//