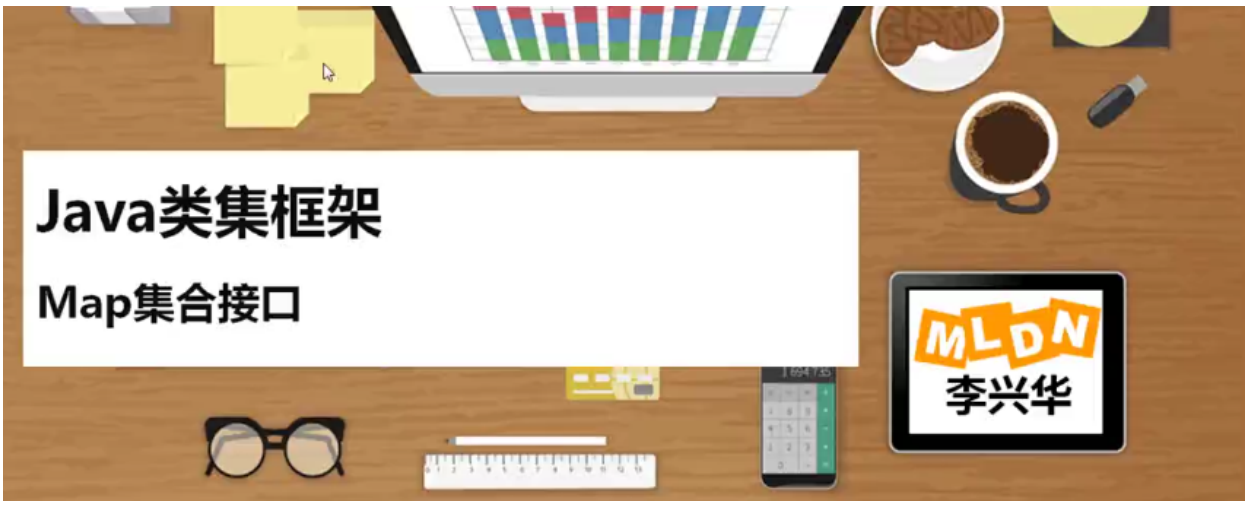


博客： https://www.cnblogs.com/HOsystem/p/14116443.html

2、具体内容

在之前已经学习了Collection接口以及其对应的子接口，可以发现在Collection接口之中所保存的数据全部都只是单个对象，在数据结构里面除了可以进行单个对象的保存之外，实际上也可以进行二元偶对象的保存(key=value)的形式来存储，而存储二元偶对象的核心意义在于，需要通过key获取对应的value。

在开发里面：Collection集合保存数据的目的是为了输出，Map集合保存数据的目的是为了进行key的查找。

■Map接口简介

Map接口是进行二元偶对象保存的最大父接口，该接口定义如下：

```
public interface Map<K,V>
```

该接口为一个独立的父接口，并且在进行接口对象实例化的时候需要设置Key与value的类型，也就是说在整体操作的时候需要保存两个内容，在Map接口里面定义有许多的操作方法，但是需要记住以下的核心操作方法：

No	方法名称	类型	描述
01	public V put(K key,V value)	普通	向集合之中保存数据
02	public V get(Object key)	普通	根据Key查询数据
03	public Set<Map.Entry<K,V>> entrySet()	普通	将Map集合转为Set集合
04	public boolean containsKey(Object key)	普通	查询指定的key是否存在
05	public Set<K> keySet()	普通	将Map集合中的key转为Set集合
06	public V remove(Object key)	普通	根据key删除掉指定的数据

从JDK1.9之后Map接口里面也扩充了一些静态方法供用户使用。

范例：观察Map集合的特点

```
package cn.mldn.demo;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = Map.of("one", 1, "two", 2, "one", 101, null, 0);
		System.out.println(map);
	}
}
```

在Map集合之中数据的保存就是按照“key=value”的形式存储的，并且使用of()方法的时候里面的数据不不允许重复的，如果重复则会出现“IllegalArgumentException”异常，如果设置的内容为null，则会出现“NullPointerException”。

对于现在见到的of()方法严格意义上来讲并不是Map集合的标准用法，因为正常的开发之中需要通过Map集合的子类泪来进行接口对象的实例化，而常用的子类：HashMap、Hashtable、TreeMap、LinkedHashMap。

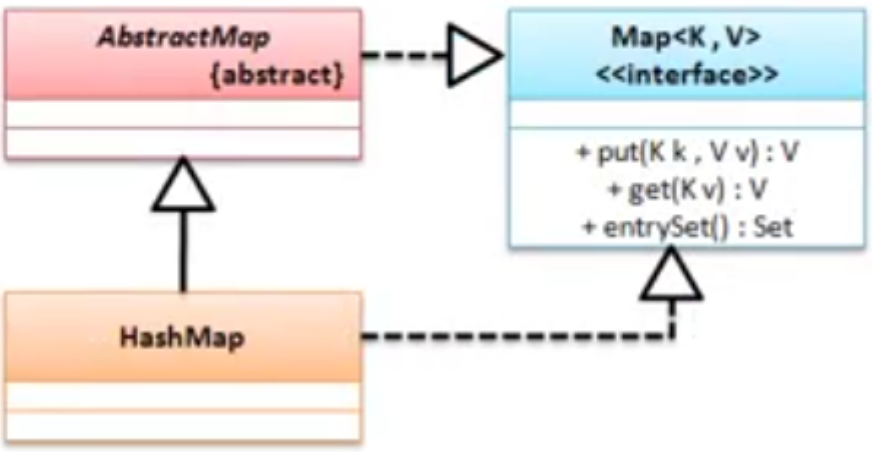
■HashMap子类

HashMap是Map接口之中最为常见的一个子类，该类的主要特点是无序存储，通过Java文档首先来观察一下HashMap子类的定义形式：

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

该类的定义继承形式符合之前的集合定义形式，依然提供有抽象类并且依然需要重复实现Map接口。

HashMap子类



范例：观察map集合的使用

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new HashMap<String,Integer>() ;
		map.put("one", 1) ;
		map.put("two", 2) ;
		map.put("one", 101) ;	// key重复
		map.put(null, 0) ;		// key为null
		map.put("zero", null) ; // value为null
		System.out.println(map.get("one"));	// key存在
		System.out.println(map.get(null));  // key存在
		System.out.println(map.get("ten"));	// key不存在
	}
}
```

以上的操作形式为Map集合使用的最标准的处理形式，通过代码可以发现，通过HashMap实例化的Map接口可以针对于key或value保存null的数据，同时也可以发现即便保存数据的key重复，那么也不会出现错误，而是出现内容的替换。

但是对于Map接口中提供的put()方法本身是提供有返回值的，那么这个返回值指的是在重复key的情况下返回旧的value。

范例：观察put()方法

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new HashMap<String,Integer>() ;
		System.out.println(map.put("one", 1));  // key不重复，返回null
		System.out.println(map.put("one", 101));	// key重复，返回旧数据
	}
}
```

在设置了相同的key的内容的时候put()方法会返回原始的数据内容。

清楚了HashMap的基本功能之后下面就需要来研究一下HashMap之中给出的源代码。HashMap之中肯定需要存储大量的数据，那么对于数据的存储

public HashMap() {
this.loadFactor = DEFAULT_LOAD_FACTOR;
}
	当使用无参构造的时候会出现有一个loadFactor属性，并且该属性默认的内容为“0.75”(static final float DEFAULT_LOAD_FACTOR = 0.75f;)

public V put(K key, V value) {
return putVal(hash(key), key, value, false, true);
}
	在使用Put()方法进行数据保存的时候会调用一个putVal()方法,同时会将key进行hash处理(生成一个hash码),而对于putVal()方法里面会发现依然会提供有一个Node节点类进行数据的保存,而在使用putVal()方法操作的过程之中会调用有一个resize()方法可以进行容量的扩充

面试题：在进行HashMap的put()操作的时候，如何实现容量扩充的?

·在HashMap类里面提供有一个“DEFAULT_INITIAL_CAPACITY”常量，作为初始化的容量配置，而这个常量的默认大小为16个元素，也就是说默认可以保存的最大内容是16；

·当保存的内容的容量超过了一个阈值(DEFAULT_LOAD_FACTOR = 0.75f)，相当于“容量 * 阈值 = 12” 保存12个元素的时候就会进行容量的扩充；

·在进行扩充的时候HashMap采用的是成倍的扩充模式，即：每一次都扩充2倍的容量；

面试题：请解释HashMap的工作原理(JDK1.8之后开始的)

·在HashMap之中进行数据存储的依然是利用了Node类完成的，那么这种情况下就证明可以使用的数据结构只有两种：链表（时间复杂度“O(N)”）、二叉树（时间复杂度“O(logn)”）；

`从JDK1.8开始，HashMap的实现出现了改变，因为其要适应于大数据时代的海量数据问题，所以对于其存储发生了变化，并且在HashMap类的内部提供有一个重要的常量：static final int TREEIFY_THRESHOLD= 8;在使用HashMap进行数据保存的时候，如果保存的数据个数没有超过阈值8（TREEIFY_THRESHOLD），那么就会按照链表的形式进行存储，而如果超过了这个阈值，则会将链表转为红黑树以实现树的平衡，并且利用左旋与右旋保证数据的查询性能。

```
/**
 * The bin count threshold for using a tree rather than list for a
 * bin.  Bins are converted to trees when adding an element to a
 * bin with at least this many nodes. The value must be greater
 * than 2 and should be at least 8 to mesh with assumptions in
 * tree removal about conversion back to plain bins upon
 * shrinkage.
 */
static final int TREEIFY_THRESHOLD = 8;
```

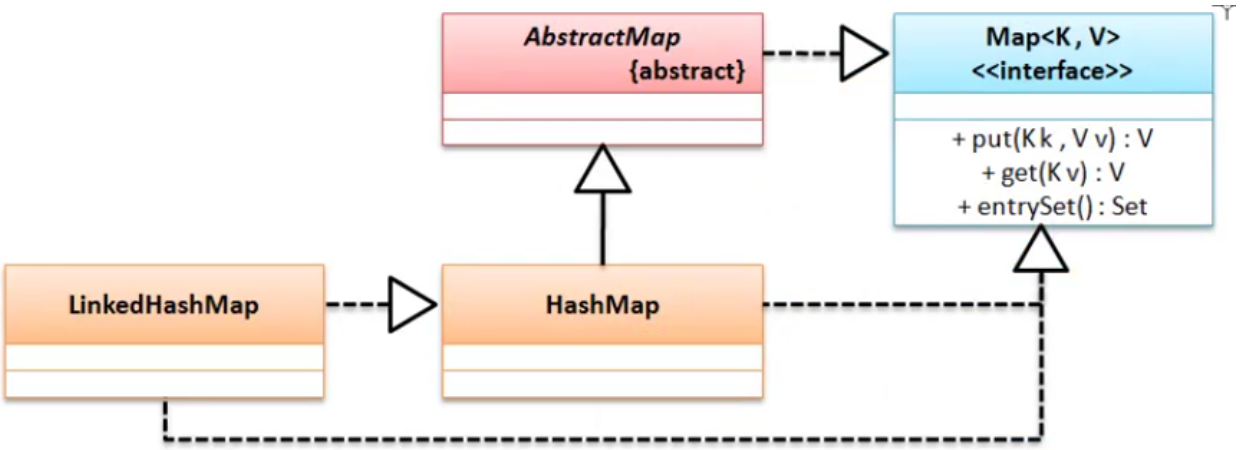
```
static final int DEFAULT_INITIAL_CAPACITY = 1 << 4; // aka 16
```

■LinkedHashMap

HashMap虽然是Map集合最为常用的一个子类，但是其本身所保存的数据都是无须的(有序与否对Map没有影响)，如果现在希望Map集合之中保存的数据的顺序为升序顺序，则就可以更换子类为LinkedHashMap(基于链表实现的)，观察LinkedHashMap类的定义形式：

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

既然是链表保存，所以一般在使用LinkedHashMap类的时候往往数据量都不要特别大，因为会造成时间复杂度攀升，通过继承结构可以发现LinkedHashMap是HashMap子类，继承关系如下：



范例：使用LinkedHashMap

```
package cn.mldn.demo;
import java.util.LinkedHashMap;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new LinkedHashMap<String,Integer>() ;
		map.put("one", 1) ;
		map.put("two", 2) ;
		map.put("one", 101) ;	// key重复
		map.put(null, 0) ;		// key为null
		map.put("zero", null) ; // value为null
		System.out.println(map);		// key不存在
	}
}
```

通过此时的程序执行可以发现当使用LinkedHashMap进行存储之后所有数据的保存顺序为添加顺序。

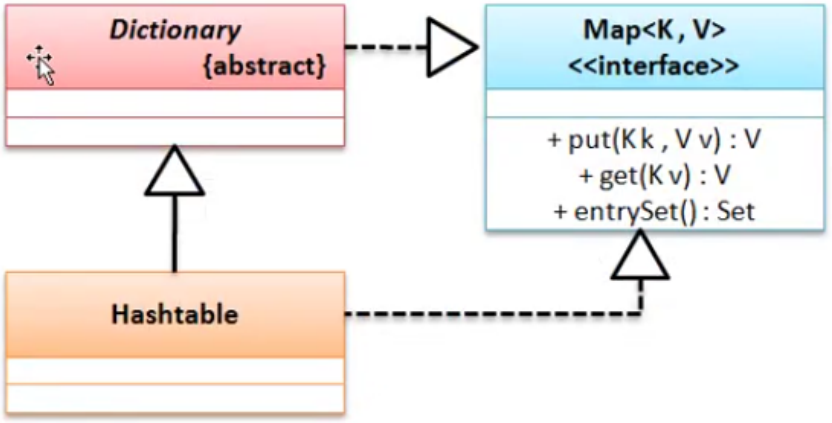
■Hashtable子类

Hashtable类是从JDK1.0的时候提供的，与Vector、Enumeration属于最早的一批动态数组的实现类，后来为了将其继续保存下来所以让其多实现了一个Map接口，Hashtable类的定义如下：

```
public class Hashtable<K,V>
 extends Dictionary<K,V> implements Map<K,V>, Cloneable, Serializable
```

Hashtable类的继承结构如下：

Hashtable子类



范例：观察Hashtable子类的使用

```
package cn.mldn.demo;
import java.util.Hashtable;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new Hashtable<String,Integer>() ;
		map.put("one", 1) ;
		map.put("two", 2) ;
		map.put("one", 101) ;	// key重复
		System.out.println(map);	// key不存在
	}
}
```

通过观察可以发现在Hashtable里面进行数据存储的时候设置的key或value都不允许为null，否则会出现NullPointerException异常。

面试题：请解释HashMap与Hashtable的区别?

·HashMap中的方法属于异步操作(非线程安全),HashMap允许保存有null数据；

·Hashtable中的方法属于同步方法(线程安全),Hashtable不允许保存null，否则会出现NullPointerException；

■Map.Entry接口

虽然已经清楚了整个的Map集合的基本操作形式，但是依然需要有一个核心的问题要解决，Map集合里面是如何进行数据存储的？对于List而言(LinkedList子类)依靠的是链表的形式实现的数据存储，那么在进行数据存储的时候一定要将数据保存在一个Node节点之中，虽然在HashMap里面也可以见到Node类型定义，通过源代码定义可以发现，HashMap类中的node内部类本身实现了Map.Entry()接口。

```
static class Node<K, V> implements Map.Entry<K, V>{}
```

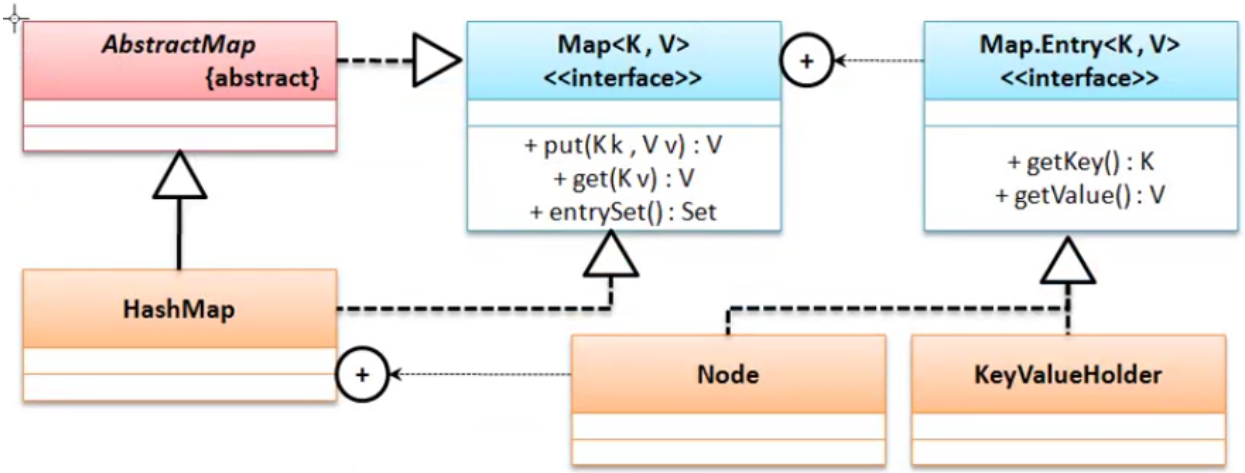
所以可以得出结论:所有的key和value的数据都被封装在Map.Entry接口之中,而此接口定义如下:

```
public static interface Map.Entry<K,V>
```

并且在这个内部接口里面提供有两个重要的操作方法:

·获取key:public K getKey();

·获取value:public V getValue();



在JDK1.9以前的开发版本之中,使用者基本上都不会去考虑创建Map.Entry的对象,实际上在正常的开发过程之中使用者也不需要关心Map.Entry对象创建,可是从JDK1.9之后,map接口里面追加有一个新的方法:

·创建Map.Entry对象:public static <K,V> Map.Entry<K,V> entry(K k,V v);

范例:创建Map.Entry对象

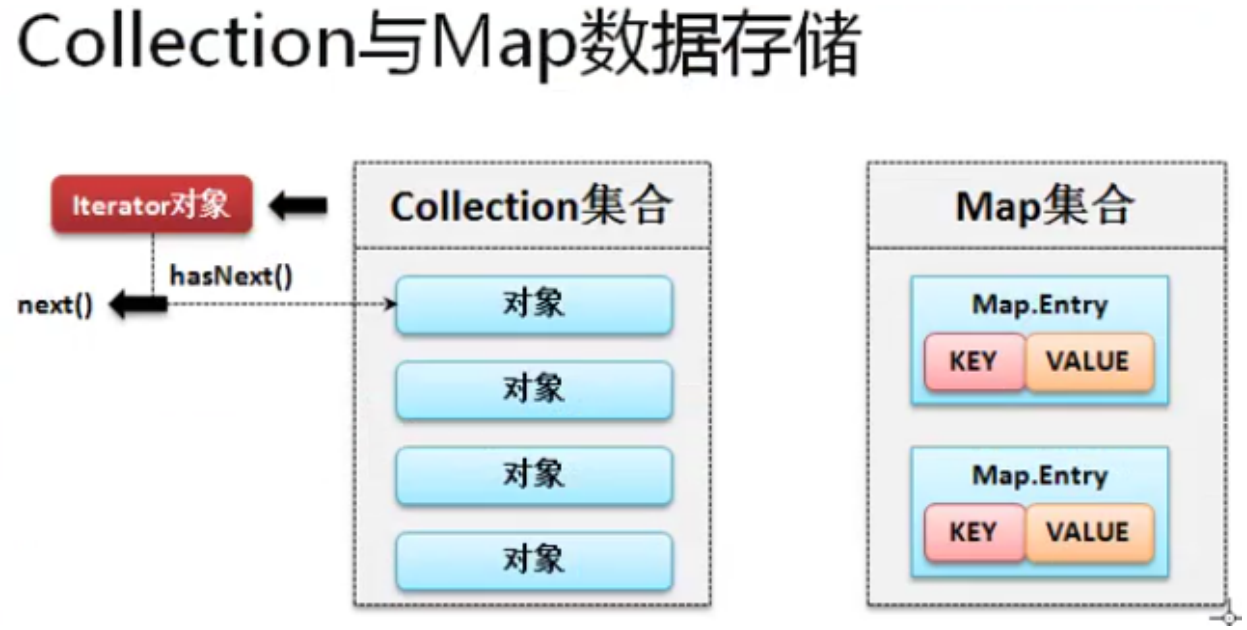
```
package cn.mldn.demo;
import java.util.Map;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map.Entry<String, Integer> entry = Map.entry("one", 1) ;
		System.out.println("获取Key:" + entry.getKey());
		System.out.println("获取Value:" + entry.getValue());
		System.out.println(entry.getClass().getName()); // 观察使用的子类
	}
}
```

通过分析可以发现在整个的Map集合里面,Map.Entry的主要作用就是作为一个Key和Value的包装类型使用,而大部分情况下载进行数据存储的时候都会将Key和Value包装为一个Map.Entry对象进行使用。

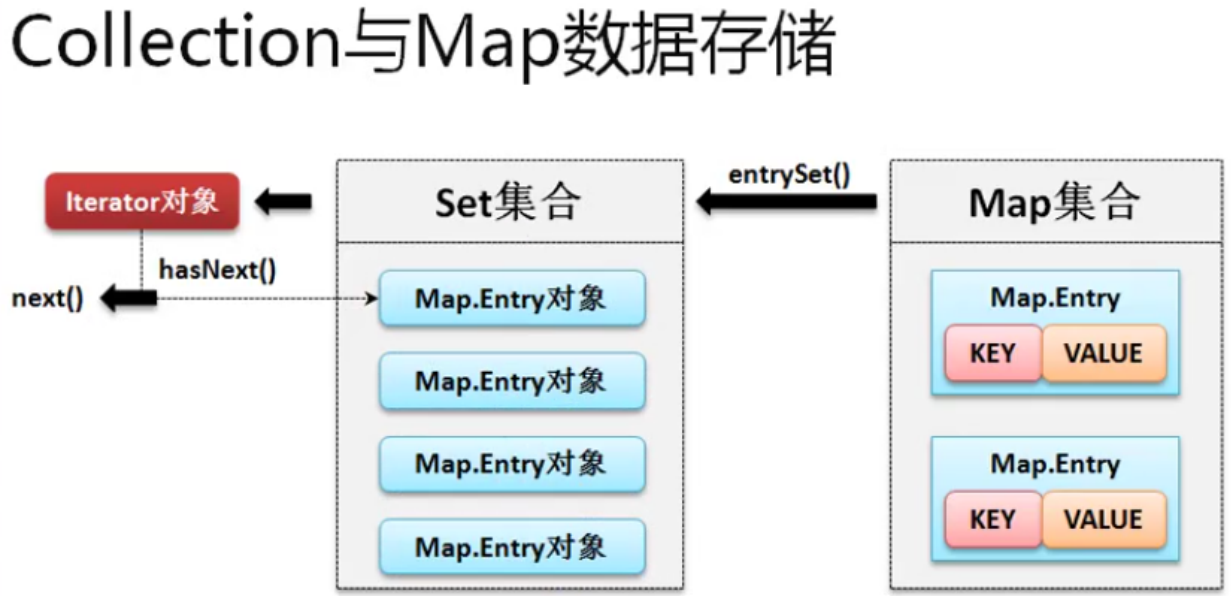
■使用Iterator输出Map集合

对于集合的输出而言,最标准的做法就是利用Iterator接口来完成,但是需要明确一点的是在Map集合里面并没有一个方法可以直接返回Iterator接口对象,所以这种情况下就必

须分析不直接提供Iterator接口实例化的方法的原因，下面对Collection与Map集合的存储结构进行一个比较。



发现在Map集合里面保存的实际上是一组Map.Entry接口对象(里面包装的是Key与Value,所以整个来讲Map依然实现的是单值的保存,这样在Map集合里面提供有一个方法“public Set<Map.Entry<K,
V>> entrySet()”,将全部的Map集合转为Set集合。



经过分析可以发现如果要想使用Iterator实现Map集合的输出则必须按照如下步骤处理：

·利用Map接口中提供的entrySet()方法将Map集合转为Set集合；

·利用Set接口中的iterator()方法将Set集合转为Iterator接口实例；

·利用Iterator进行迭代输出获取每一组的Map.Entry对象,随后通过getKey()与getValue()获取数据

范例：利用Iterator输出Map集合

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new HashMap<String, Integer>();
		map.put("one", 1);
		map.put("two", 2);
		Set<Map.Entry<String,Integer>> set = map.entrySet() ;	// 将Map集合变为Set集
合
		Iterator<Map.Entry<String,Integer>> iter = set.iterator() ;
		while (iter.hasNext()) {
			Map.Entry<String, Integer> me = iter.next() ;
			System.out.println(me.getKey() + " = " + me.getValue());
		}
	}
}
```

虽然Map集合本身支持有迭代输出的支持，但是如果从实际的开发来讲，Map集合最主要的用法在于实现数据的Key查找操作，另外需要提醒的是，如果现在不使用Iterator而是用foreach语法输出则也需要将Map集合转为Set集合。

范例：使用foreach输出Map集合

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
		Map<String, Integer> map = new HashMap<String, Integer>();
		map.put("one", 1);
		map.put("two", 2);
		Set<Map.Entry<String,Integer>> set = map.entrySet() ;	// 将Map集合变为Set集
合
		for (Map.Entry<String, Integer> entry : set) {
			System.out.println(entry.getKey() + " = " + entry.getValue());
		}
	}
}
```

由于Map迭代输出的情况相对较少，所以对于此类的语法应该深入理解一下，并且一定要灵活掌握。

■关于KEY的定义

在使用Map集合的时候可以发现对于Key和Value的类型实际上都可以由使用者任意决定，那么也就意味着现在依然可以使用自定义的类来进行Key类型的设置。对于自定义Key

类型所在的类中一定要覆写hashCode()与equals()方法，否则无法查找到。

public V put(K key, V value) {
return putVal(hash(key), key, value, false, true);
}
	在进行数据保存的时候发现会自动使用传入的key的数据生成一个hash码，也就是说存储的时候是有这个hash数值

public V get(Object key) {
Node<K,V> e;
return (e = getNode(hash(key), key)) == null ? null : e.value;
}
	在根据key获取数据的时候依然要将传入的key通过hash()方法来获取其对应的hash码,那么也就证明,查询的过程之中首先要利用hashCode()来进行数据查询,但使用getNode()方法查询的时候还需要使用到equals()方法

范例：使用自定义类作为Key类型

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Map;
class Person {
	private String name;
	private int age;
	public Person(String name, int age) {
		this.name = name;
		this.age = age;
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + age;
		result = prime * result + ((name == null) ? 0 : name.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Person other = (Person) obj;
		if (age != other.age)
			return false;
		if (name == null) {
			if (other.name != null)
				return false;
		} else if (!name.equals(other.name))
			return false;
		return true;
	}
}
public class JavaAPIDemo {
	public static void main(String[] args) throws Exception {
```

```
        Map<Person, String> map = new HashMap<Person, String>();
        map.put(new Person("小强", 78), "林弱"); // 使用自定义类作为Key
        System.out.println(map.get(new Person("小强", 78))); // 通过key找到value
    }
}
```

```
package cn.mldn.demo;
import java.util.HashMap;
import java.util.Map;
class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + age;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Person other = (Person) obj;
        if (age != other.age)
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }
}
public class JavaAPIDemo {
    public static void main(String[] args) throws Exception {
        Map<Person, String> map = new HashMap<Person, String>();
        map.put(new Person("小强", 78), "林弱"); // 使用自定义类作为Key
        System.out.println(map.get(new Person("小强", 78))); // 通过key找到value
    }
}
```

虽然允许你使用自定义的类作为Key的类型，但是也需要注意一点，在实际的开发之中对于Map集合的Key常用的类型就是：String、Long、Integer，尽量使用系统类。

面试题：如果在进行HashMap进行数据操作的时候出现了Hash冲突(Hash码相同)，HashMap是如何解决的?

当出现了Hash冲突之后为了保证程序的正常执行，会在冲突的位置上将所有的Hash冲突的内容转为链表保存。