# 12月18日

## 一个简单需求：HashMap实现相同key存入数据后不被覆盖

<https://blog.csdn.net/u010648555/article/details/89390250>

存入相同的key后，旧值就被新值替换

/\*\*

\* HashMap 的put方法

\*\*/

public V put(K key, V value) {

return putVal(hash(key), key, value, false, true);

}

/\*\*

\* HashMap 的containsKey方法

\*\*/

public boolean containsKey(Object key) {

return getNode(hash(key), key) != null;

}

/\*\*

\* 将存入的key进行hash操作，也就是使用key.hashCode()！

\*\*/

static final int hash(Object key) {

int h;

return (key == null) ? 0 : (h = key.hashCode()) ^ (h >>> 16);

}

判断put和判断key是否是同一个key的时候，使用大概如下判断逻辑：

if (e.hash == hash && ((k = e.key) == key || (key != null && key.equals(k))))

* 1

先判断[Hash](https://so.csdn.net/so/search?q=Hash&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u010648555/article/details/_blank)是否一致，然后在判断传入key和当前集合中是否有相同的key。如果key相同，则新值替换旧值。其中在判断中使用了

* ==
* equals

==和 equals 的区别有时候面试会问到，如何你知道这两个的区别不仅看源码能够很好的理解，并且遇到面试也不怕了。

tips：简述==和 equals 的区别>

1）对于==，如果作用于基本数据类型的变量，则直接比较其存储的 “值”是否相等；如果作用于引用类型的变量，则比较的是所指向的对象的地址！

2）对于equals方法，注意：equals方法不能作用于基本数据类型的变量。如果没有对equals方法进行重写，则比较的是引用类型的变量所指向的对象的地址；诸如[String](https://marketing.csdn.net/p/3127db09a98e0723b83b2914d9256174?pId=2782&utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u010648555/article/details/_blank)等类对equals方法进行了重写的话，比较的是所指向的对象的内容。

有了上面的分析基础，那针对上面String类型的key的话，那实现起来就比较简单了！因为String中已经实现了HashCode和 equals代码如下：

* 自定义HashMap

public class MyHashMap<K> extends HashMap<K,String> {

/\*\*

\* 使用HashMap中containsKey判断key是否已经存在

\* @param key

\* @param value

\* @return

\*/

@Override

public String put(K key, String value) {

String newV = value;

if (containsKey(key)) {

String oldV = get(key);

newV = oldV + "---" + newV;

}

return super.put(key, newV);

}}

public static void main(String[] args) {

MyHashMap<String> map = new MyHashMap<String>();

map.put("aflyun", "Java编程技术乐园");

map.put("aflyun", "生活在长沙的延安人");

map.put("aflyun", "期待你加入乐园");

System.out.println(map.toString());

}

--打印：---

{aflyun=Java编程技术乐园---生活在长沙的延安人---期待你加入乐园}

此时同样的key内容是进行叠加的，不是进行替换！那如何是自定义的类，要当作key，那要怎么做呢？

其实也就是重写了hashCode和equals就可以了。

public class PrettyGirl {

/\*\*

\* 姑娘唯一认证ID

\*/

private String id;

/\*\*

\* 姑娘姓字名谁

\*/

private String name;

@Override

public boolean equals(Object o) {

if (this == o) {return true;}

if (o == null || getClass() != o.getClass()) {return false;}

PrettyGirl that = (PrettyGirl) o;

return Objects.equals(id, that.id) &&

Objects.equals(name, that.name);

}

@Override

public int hashCode() {

return Objects.hash(id, name);

}

}

public static void main(String[] args) {

PrettyGirl prettyGirl = new PrettyGirl();

Map<PrettyGirl,String> map = new HashMap<>();

map.put(prettyGirl, "Java编程技术乐园");

map.put(prettyGirl, "生活在长沙的延安人");

map.put(prettyGirl, "期待和你加入乐园");

System.out.println("map :" + map.toString());

MyHashMap<PrettyGirl> myMap = new MyHashMap<PrettyGirl>();

myMap.put(prettyGirl, "Java编程技术乐园");

myMap.put(prettyGirl, "生活在长沙的延安人");

myMap.put(prettyGirl, "期待和你加入乐园");

System.out.println("myMap :" + myMap.toString());

}

--打印：---

map :{com.happy.PrettyGirl@3c1=期待和你加入乐园}

myMap :{com.happy.PrettyGirl@3c1=Java编程技术乐园---生活在长沙的延安人---期待和你加入乐园}

总结：要实现开头的需求

1、如果是类似String这种，已经重写了hashCode和equals的。则只需要创建一个自己的HashMap类，重写put即可。

2、如果是自定义的类，那就必须重写了hashCode和equals的，然后在使用自定义的HashMap类了。

具体的代码判断逻辑：

判断key是否存在的时候是先比较key的hashCode，再比较相等或equals的，所以重写hashCode()和equals()方法即可实现添加重复元素。重写这两个方法之后就可以覆盖重复的键值对，如果需要对value进行叠加，调用put()方法之前用containsKey()方法判断是否有重复的键值，如果有，则用get()方法获取原有的value，再加上新加入的value即可。

本文涉及的相关知识：

1、HashMap相关源码

2、== 、equals和 hashCode

3、Hash[算法](https://edu.csdn.net/course/detail/40020?utm_source=glcblog&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u010648555/article/details/_blank)

相关面试题：

1、为什么要重写了equals方法必须要重写hashcode方法？

2、使用HashMap在什么情况会出现内存泄漏？