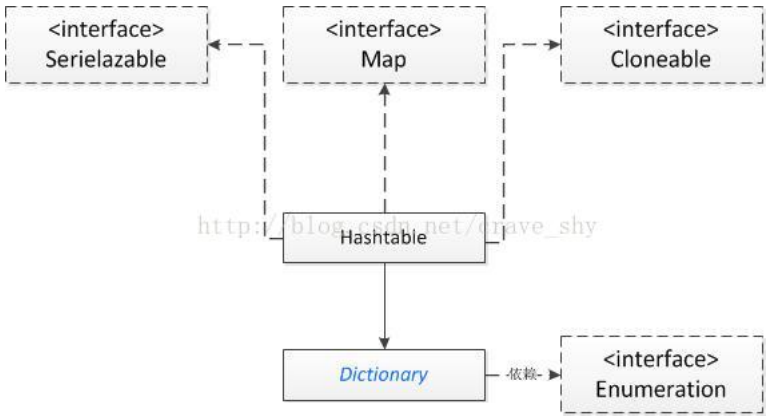
java_集合体系之Hashtable详解、源码及示例——10

原创　2013年12月26日 15:29:50　1840　1　9

java_集合体系之Hashtable详解、源码及示例——10

一：Hashtable结构图



简单说明：

1、上图中虚线且无依赖字样、说明是直接实现的接口
2、虚线但是有依赖字样、说明此类依赖与接口、但不是直接实现接口
3、实线是继承关系、类继承类、接口继承接口
4、实现Serielazable接口、允许使用ObjectInputStream/ObjectOutputStream读取/写入
5、实现Map接口、以键值对形式存储数据
6、实现Cloneable接口、允许克隆Hashtable
7、继承Dictionary、说明Hashtable是键值对形式类、并且键、值都不允许为null。
8、Dictionary依赖Enumeration、Hashtable可以使用Enumeration、Iterator迭代其中元素。

二：Hashtable类简介：

1、 基于哈希表的Map结构的实现
2、线程安全
3、内部映射无序
4、不允许值为null的key和value

三：Hashtable API

1、构造方法


Oscar Chen (http://blog....
+关注
(http://blog.csdn.net/chenghuaying)
原创 182　粉丝 14　喜欢 3

› CentOS 集群机器之间ssh免密 (/crave_shy/article/details/72964997)
› JMV-内存管理-运行时数据区域 (/crave_shy/article/details/56675052)
› JMV-Blog目录 (/crave_shy/article/details/56675032)
› JVM-为什么要学JVM (/crave_shy/article/details/56673439)
更多文章 (http://blog.csdn.net/chenghuaying)

```
// 默认构造函数。
public Hashtable()

// 指定“容量大小”的构造函数
public Hashtable(int initialCapacity)

// 指定“容量大小”和“加载因子”的构造函数
public Hashtable(int initialCapacity, float loadFactor)

// 包含“子Map”的构造函数
public Hashtable(Map<? extends K, ? extends V> t)
```

2、一般方法

```
synchronized void               clear()
synchronized Object             clone()
             boolean            contains(Object value)
synchronized boolean            containsKey(Object key)
synchronized boolean            containsValue(Object value)
synchronized Enumeration<V>     elements()
synchronized Set<Entry<K, V>>   entrySet()
synchronized boolean            equals(Object object)
synchronized V                  get(Object key)
synchronized int                hashCode()
synchronized boolean            isEmpty()
synchronized Set<K>             keySet()
synchronized Enumeration<K>     keys()
synchronized V                  put(K key, V value)
synchronized void               putAll(Map<? extends K, ? extends V> map)
synchronized V                  remove(Object key)
synchronized int                size()
synchronized String             toString()
synchronized Collection<V>      values()
```

四：Hashtable 源码分析

说明：

1、对哈希表要有简单的认识、

2、Hashtable是通过“拉链法”解决哈希冲突的

3、理解Hashtable源码中的关键部分、Entry实体类的行为、属性。Entry的存储方式、Hashtable的扩容方式、Hashtable内部关于获取新的hash code的算法。

4、与遍历相关：可以使用Enumeration、也可以使用Iterator。

5、与容量有关的内容Hashtable的实例有两个参数影响其性能：初始容量和加载因子。容量是哈希表中桶的数量，初始容量只是哈希表在创建时的容量。加载因子是哈希表在其容量自动增加之前可以达到多满的一种尺度。当哈希表中的条目数超出了加载因子与当前容量的乘积时，则要对该哈希表进行rehash 操作（即重建内部数据结构），从而哈希表将具有大约两倍的桶数。

6、默认加载因子 (0.75) 在时间和空间成本上寻求一种折衷。加载因子过高虽然减少了空间开销，但同时也增加了查询成本（在大多数Hashtable 类的操作中，包括get 和put 操作，都反映了这一点）。初始容量主要控制空间消耗与执行 rehash 操作所需要的时间损耗之间的平衡。如果初始容量大于Hashtable 所包含的最大条目数除以加载因子，则永远不会发生 rehash 操作。但是，将初始容量设置太高可能会浪费空间。

7、如果迭代性能很重要，则不要将初始容量设置得太高（或将加载因子设置得太低）。

8、如果很多映射关系要存储在 Hashtable 实例中，则相对于按需执行自动的 rehash 操作以增大表的容量来说，使用足够大的初始容量创建它将使得映射关系能更有效地存储。

总结 ：

1、数据结构：Hashtable是以哈希表的形式存储数据的、并且是通过“拉链法”解决冲突、Hashtable中存储的Entry继承Map.Entry<K,V>即实现了getKey() getValue() setValue() equals() hashCode()方法、关于Hashtable存储元素的结构

```
    /** Hashtable中表示节点的实体类、本质是一个单向链表*/
private static class Entry<K,V> implements Map.Entry<K,V> {
            int hash;
            K key;
            V value;
            Entry<K,V> next;

            protected Entry(int hash, K key, V value, Entry<K,V> next) {
                this.hash = hash;
                this.key = key;
                this.value = value;
                this.next = next;
            }

            protected Object clone() {
                return new Entry<K,V>(hash, key, value,
                                      (next==null ? null : (Entry<K,V>) next.clone()));
            }

            // Map.Entry Ops

            public K getKey() {
                return key;
            }

            public V getValue() {
                return value;
            }

            public V setValue(V value) {
                if (value == null)
                    throw new NullPointerException();

                V oldValue = this.value;
                this.value = value;
                return oldValue;
            }

            public boolean equals(Object o) {
                if (!(o instanceof Map.Entry))
                    return false;
                Map.Entry e = (Map.Entry)o;

                return (key==null ? e.getKey()==null : key.equals(e.getKey())) &&
                   (value==null ? e.getValue()==null : value.equals(e.getValue()));
            }

            public int hashCode() {
                return hash ^ (value==null ? 0 : value.hashCode());
            }

            public String toString() {
                return key.toString()+"="+value.toString();
            }
}
```

2、通过Key的哈希值定位索引的算法：

```
    // 计算索引值， % tab.length 的目的是防止数据越界
    int hash = key.hashCode();
    int index = (hash & 0x7FFFFFFF) % tab.length;
```

3、遍历：

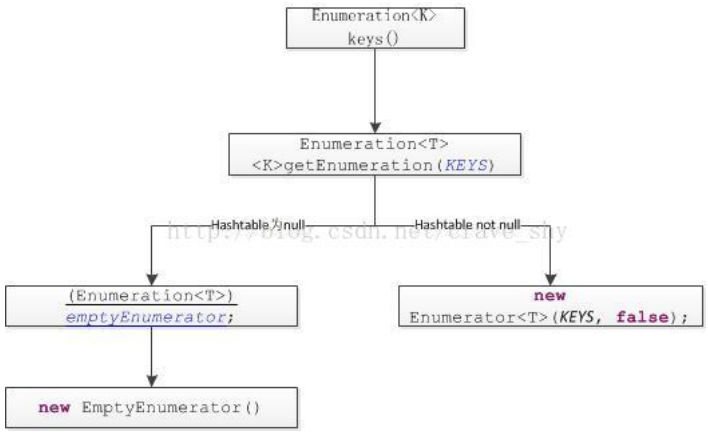
a）Enumerator实现了Enumeration和Iterator接口、说明Enumerator同时具有使用Enumeration迭代和使用Iterator迭代功能、

```
        private class Enumerator<T> implements Enumeration<T>, Iterator<T> {
        //指向当前Hashtable的table
                Entry[] table = Hashtable.this.table;
                int index = table.length;
                Entry<K,V> entry = null;
                Entry<K,V> lastReturned = null;
                int type;

                /**Enumerator是迭代器还是Enumeration的标志、true——Iterator、false——Enumeration*/
                boolean iterator;

                /** 将Enumerator当作Iterator使用时需要用到的标志fail-fast机制*/
                protected int expectedModCount = modCount;

                Enumerator(int type, boolean iterator) {
                    this.type = type;
                    this.iterator = iterator;
                }

                // 从table末尾向前查找，直到找到不为null的Entry。
                public boolean hasMoreElements() {
                    Entry<K,V> e = entry;
                    int i = index;
                    Entry[] t = table;
                    /* Use locals for faster loop iteration */
                    while (e == null && i > 0) {
                        e = t[--i];
                    }
                    entry = e;
                    index = i;
                    return e != null;
                }

                //获取下一个元素、
                public T nextElement() {
                    Entry<K,V> et = entry;
                    int i = index;
                    Entry[] t = table;
                    /* Use locals for faster loop iteration */
                    while (et == null && i > 0) {
                        et = t[--i];
                    }
                    entry = et;
                    index = i;
                    if (et != null) {
                                Entry<K,V> e = lastReturned = entry;
                                entry = e.next;
                                return type == KEYS ? (T)e.key : (type == VALUES ? (T)e.value : (T)
e);
                    }
                    throw new NoSuchElementException("Hashtable Enumerator");
                }

                // Iterator方式判断是否有下一个元素
                public boolean hasNext() {
                    return hasMoreElements();
                }

                //Iterator方式获取下一个元素、多一步fail-fast验证
                public T next() {
                    if (modCount != expectedModCount)
                        throw new ConcurrentModificationException();
                    return nextElement();
                }

                /**
                 * 仅用于Iterator方式中的删除当前元素、通过计算最后一个返回的元素的hash值
                 * 定位到table中对应的元素、删除。
                 */
                public void remove() {
                    if (!iterator)
                        throw new UnsupportedOperationException();
                    if (lastReturned == null)
                        throw new IllegalStateException("Hashtable Enumerator");
                    if (modCount != expectedModCount)
                        throw new ConcurrentModificationException();

                    synchronized(Hashtable.this) {
                                Entry[] tab = Hashtable.this.table;
                                int index = (lastReturned.hash & 0x7FFFFFFF) % tab.length;
```

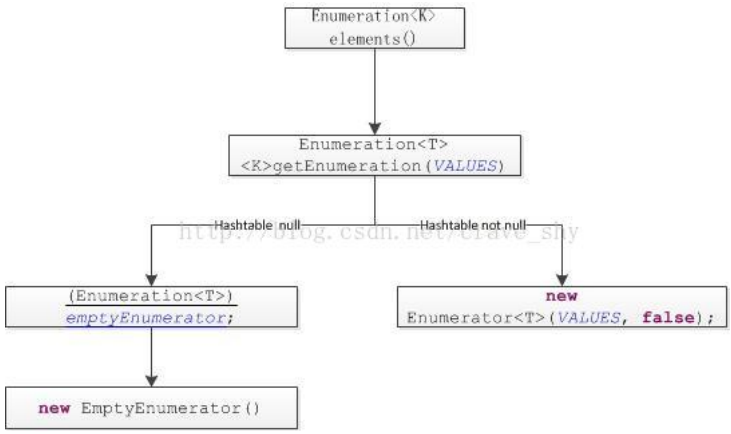
```
                                for (Entry<K,V> e = tab[index], prev = null; e != null; prev = e, e
= e.next) {
                                    if (e == lastReturned) {
                                                modCount++;
                                                expectedModCount++;
                                                if (prev == null)
                                                    tab[index] = e.next;
                                                else
                                                    prev.next = e.next;
                                                count--;
                                                lastReturned = null;
                                                return;
                                    }
                                }
                                throw new ConcurrentModificationException();
                        }
                    }
    }
```

b）使用Enumeration迭代时、获取Enumeration过程

（1）通过keys枚举：

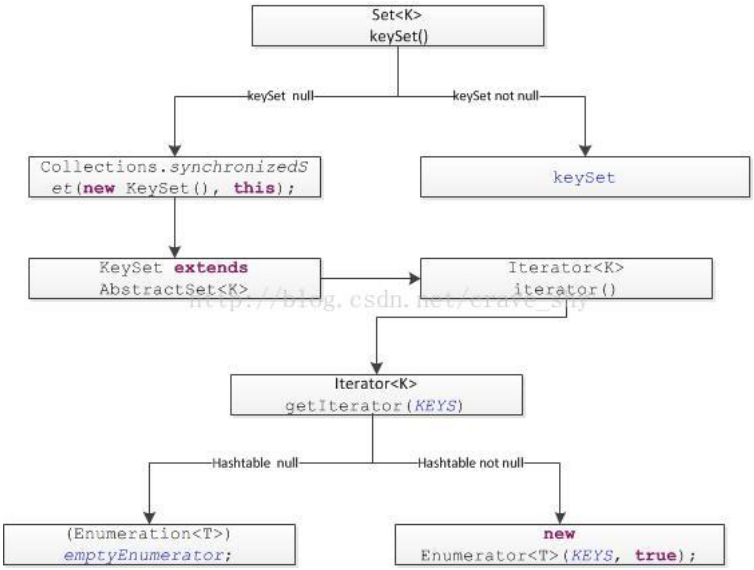


（2）通过elements枚举

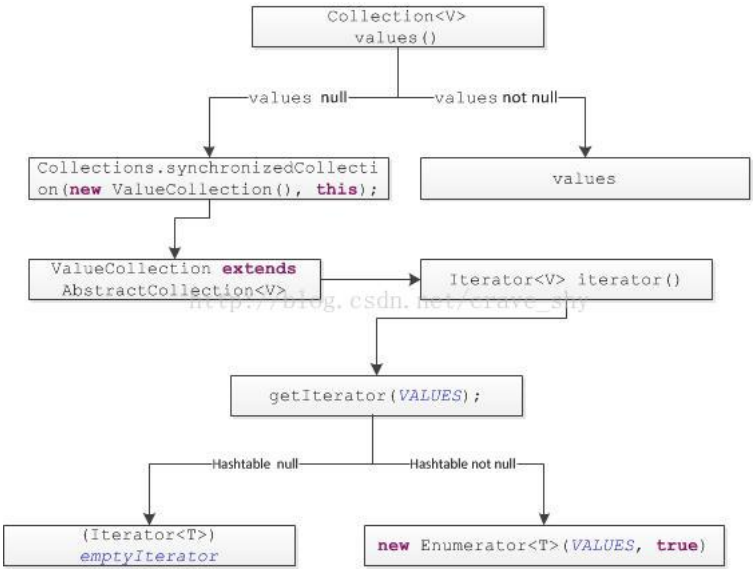


c）使用Iterator迭代

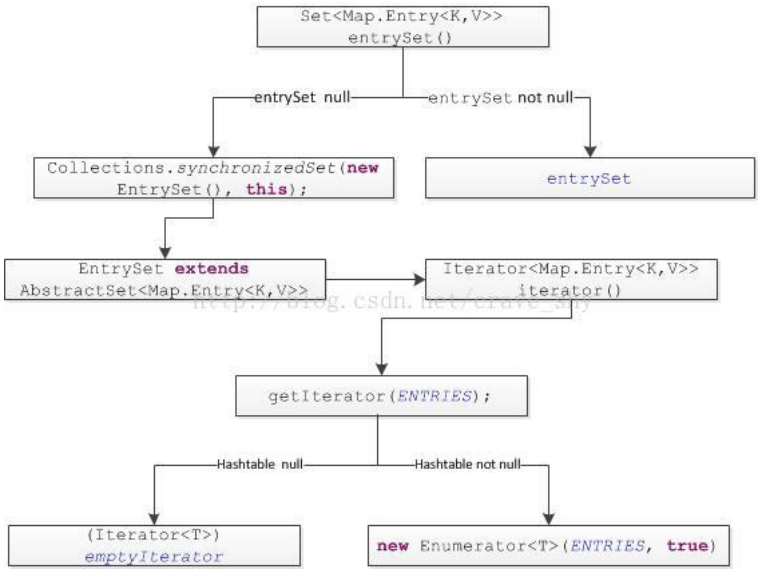
（1）通过keySet()获取Set<K>的Iterator



（2）通过values()获取Collection<K>的Iterator



（3）通过entrySet()获取Set<Map.Entry<K,V>>的Iterator



4、rehash()：当使用Hashtable对外提供的put()方法时、put()方法内部会检测容量是否大于等于阀值、是的话调用rehash()、重构。

```
/** 调整Hashtable的长度，将长度变成原来的(2倍+1)
 *  1、使用临时变量记录原来table中值
 *  2、创建一个新的容量为原来table容量*2 + 1 的table
 *  3、将原来table中值重新赋给新table
 */
protected void rehash() {
            int oldCapacity = table.length;
            Entry[] oldMap = table;

            int newCapacity = oldCapacity * 2 + 1;
            Entry[] newMap = new Entry[newCapacity];

            modCount++;
            threshold = (int)(newCapacity * loadFactor);
            table = newMap;

            for (int i = oldCapacity ; i-- > 0 ;) {
                for (Entry<K,V> old = oldMap[i] ; old != null ; ) {
                            Entry<K,V> e = old;
                            old = old.next;

                            int index = (e.hash & 0x7FFFFFFF) % newCapacity;
                            e.next = newMap[index];
                            newMap[index] = e;
                }
            }
}
```

```
    /** 将键值对存入Hashtable、不允许value为null*/
    public synchronized V put(K key, V value) {
                // Make sure the value is not null
                if (value == null) {
                    throw new NullPointerException();
                }

                // 如果存在相同key、则使用传入value替代旧的的value
                Entry tab[] = table;
                int hash = key.hashCode();
                int index = (hash & 0x7FFFFFFF) % tab.length;
                for (Entry<K,V> e = tab[index] ; e != null ; e = e.next) {
                    if ((e.hash == hash) && e.key.equals(key)) {
                                V old = e.value;
                                e.value = value;
                                return old;
                    }
                }

                // 若“Hashtable中不存在键为key的键值对”

                //修改Hashtable结构变动次数
                modCount++;
                //如果Hashtable中键值对总数大于等于阀值、则rehash()、即将容量扩增2倍+1
                if (count >= threshold) {
                    // Rehash the table if the threshold is exceeded
                    rehash();
            tab = table;
            index = (hash & 0x7FFFFFFF) % tab.length;
                }

                // 将“Hashtable中index”位置的Entry(链表)保存到e中
                Entry<K,V> e = tab[index];
                //创建新的Entry节点，并将新的Entry插入Hashtable的index位置，并设置e为新的Entry的下
一个元素。
                tab[index] = new Entry<K,V>(hash, key, value, e);
                //容量+1
                count++;
                return null;
    }
```

五：Hashtable 示例

1、遍历方式：

a）通过keys获取枚举类型对象遍历：

```
    Enumeration<String> e = hashtable.keys();
```

b）通过elements获取枚举类型对象对象遍历：

```
    Enumeration<String> e = hashtable.elements();
```

c）通过keySet获取Set类型对象的Iterator遍历：

```
    Set<String> keySet = hashtable.keySet();
    Iterator<String> it = keySet.iterator();
```

d）通过values获取Collection类型对象的Iterator遍历：

```
    Collection<String> values = hashtable.values();
    Iterator<String> it = values.iterator();
```

e）通过entrySet获取Set类型对象的Iterator遍历：

```
    Set<Entry<String, String>> entrySet = hashtable.entrySet();
    Iterator<Entry<String, String>> it = entrySet.iterator();
```

2、迭代示例：

```
package com.chy.collection.example;

import java.util.Collection;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;
import java.util.Map.Entry;

public class EragodicHashtable {
	//初始化Hashtable
	private static Hashtable<String, String> hashtable = new Hashtable<String, String>();
	static{
		for (int i = 0; i < 10; i++) {
			hashtable.put(""+i, ""+i);
		}
	}

	/**
	 * 测试使用keys获取Enumeration遍历
	 */
	private static void testKeys(){
		Enumeration<String> e = hashtable.keys();
		while(e.hasMoreElements()){
			System.out.println("hash table k :" + e.nextElement());
		}
		System.out.println("=============================================");
	}

	/**
	 * 测试使用elements获取Enumeration遍历
	 */
	private static void testElements(){
		Enumeration<String> e = hashtable.elements();
		while(e.hasMoreElements()){
			System.out.println("hash table vlaue ： " + e.nextElement());
		}
		System.out.println("=============================================");
	}

	/**
	 * 测试使用keySet获取Set<K>的Iterator遍历
	 */
	private static void testKeySet(){
		Set<String> keySet = hashtable.keySet();
		Iterator<String> it = keySet.iterator();
		while(it.hasNext()){
			System.out.println("hash table k : " + it.next());
		}
		System.out.println("=============================================");
	}

	/**
	 * 测试使用values获取Collection<V>的Iterator遍历
	 */
	private static void testValues(){
		Collection<String> values = hashtable.values();
		Iterator<String> it = values.iterator();
		while(it.hasNext()){
			System.out.println("hash table value : " + it.next());
		}
		System.out.println("=============================================");
	}

	/**
	 * 测试使用entrySet<Map.Entry<K, V>>的Iterator遍历
	 */
	private static void testEntrySet(){
		Set<Entry<String, String>> entrySet = hashtable.entrySet();
		Iterator<Entry<String, String>> it = entrySet.iterator();
		while(it.hasNext()){
			System.out.println("hash table entry : " + it.next());
		}
		System.out.println("=============================================");
	}

	public static void main(String[] args) {
		testKeys();
		testElements();
		testKeySet();
```

```
                testValues();
                testEntrySet();

        }
}
```

3、API示例：

```
package com.chy.collection.example;

import java.util.HashMap;
import java.util.Hashtable;

@SuppressWarnings("all")
public class HashtableTest {
        /**
         * 测试构造方法、下面四个方法效果相同、
         */
        private static void testConstructor(){
                //use default construct
                Hashtable<String, String> ht1 = new Hashtable<String, String>();
                //use specified initCapacity
                Hashtable<String, String> ht2 = new Hashtable<String, String>(11);
                //use specified initCapacity and loadFactor
                Hashtable<String, String> ht3 = new Hashtable<String, String>(11, 0.75f);
                //use specified Map
                Hashtable<String, String> ht4 = new Hashtable<String, String>(ht1);
        }


        /**
         * 测试API方法
         */
        public static void main(String[] args) {
                //初始化、键-值都为字符串"1"的hashMap
                Hashtable<String, String> ht = new Hashtable<String, String>();
                for (int i = 0; i < 10; i++) {
                        ht.put(""+i, ""+i);
                }
                /*
                 * 向Hashtable中添加键为null的键值对
                 * 只会在Hashtable的index为0处、保存一个键为null的键值对、键为null、值为最后一次添
加的键值对的值。
                 */
                ht.put(null, null);
                ht.put(null, "n");
                System.out.println(ht.size());
                System.out.println(ht);


                //是否包含键“1”
                System.out.println("Hashtable contans key ? " + ht.containsKey("1"));
                //是否包含值“1”
                System.out.println("Hashtable contans value ? " + ht.containsValue("1"));
                //获取键为“1”的值
                System.out.println("the value of key=1 " + ht.get("1"));
                //将键为“1”的值修改成"11"
                ht.put("1", "11");

                //将Hashtable复制到Hashtable1中
                Hashtable<String, String> Hashtable1 = (Hashtable<String, String>)ht.clone();
                //将Hashtable1所有键值对复制到Hashtable中
                ht.putAll(Hashtable1);
                System.out.println(ht);//不会有二十个元素、因为他不会再添加重复元素
                //如果Hashtable非空、则清空
                if(!ht.isEmpty()){
                        ht.clear();
                }
                System.out.println(ht.size());
        }
}
```

更多内容：java_集合体系之总体目录——00
(http://blog.csdn.net/crave_shy/article/details/174167

标签：Hashtable (http://so.csdn.net/so/search/s.do?q=Hashtable&t=blog) / HashMap (http://so.csdn.net/so/search/s.do?q=HashMap&t=blog) / Enumeration (http://so.csdn.net/so/search/s.do?q=Enumeration&t=blog) / Map框架图 (http://so.csdn.net/so/search/s.do?q=Map框架图&t=blog) / dictionary (http://so.csdn.net/so/search/s.do?q=dictionary&t=blog) /

1条评论

qq_36596145 (http://my.csdn.net/qq_36596145)

(http://my.csdn.net/qq_36596145)

发表评论

u011473031 (/u011473031) 2017-05-22 15:56 1楼

最后HashtableTest 中,Hashtable 不允许null的key吧?会抛出空指针异常

(/u011473031)

回复

更多评论

相关文章推荐

【五子棋AI循序渐进】发布一个完整的有一定棋力的版本（含源码） **(/cjianwyr/article/details/54911659)**

本博文来自于：http://www.cnblogs.com/zcsor/archive/2012/12/25/2832820.html 经过这半年左右的学习和探索，现在对五子棋A...

cjianwyr 2017-02-07 16:32 615

机器学习算法与Python实践之（四）支持向量机（SVM）实现 **(/zouxy09/article/details/17292011)**

机器学习算法与Python实践之（四）支持向量机（SVM）实现

zouxy09@qq.comhttp://blog.csdn.net/zouxy09 机器学习算法与Python实践这个系列...

zouxy09 2013-12-13 00:12 85522

eclipse中安装插件的四种方式 **(/after_you/article/details/54646174)**

Eclipse插件的安装方法大体有以下三种：[9] 第一种：直接复制法 假设Eclipse的安装目录在C:\eclipse，解压下载的eclipse 插件或者安装eclipse 插件到指定...

after_you 2017-01-21 15:45 192

简历 **(/abcd614110294/article/details/47358533)**

致读者: 首先谢谢大家能够下载这个文档并打开阅读。如有说得不妥的地方希望大家谅解提醒。 在黑马学习了4个月，老方和老师们都对我们云3班很是关照，我们都有目共睹的。 在此谢谢各位老师和同学,同...

abcd614110294 2015-08-08 15:07 666

目录 喜欢 收藏 分享

solr拼音检索 (/fengyong7723131/article/details/49154857)

拼音检索的大致思路是这样的： ①将需要使用拼音检索的字段汇集到一个拼音分词字段里（我的拼音分词字段使用pinyin4j+NGram做的）； ...

fengyong7723131 2015-10-15 16:57 2056

花生壳端口映射 (/nolatin/article/details/8883122)

动态域名解析“花生壳”软件使用教程由于大多数宽带用户的IP都是动态变化的（没有公网IP的朋友就别尝试了），而在域名管理中设定指向的IP必须是固定的（可以修改，但修改后需要数小时才能生效），“花生壳”（...

nolatin 2013-05-04 11:09 3764

rgw 各个pool作用 (/dengxiafubi/article/details/76206272)

struct RGWZoneParams { ...

dengxiafubi 2017-07-27 16:16 229

Linux系统编程（31）—— socket编程之TCP详解 (/itcastcpp/article/details/39047225)

TCP有源端口号和目的端口号，通讯的双方由IP地址和端口号标识。32位序号、32位确认序号、窗口大小稍后详细解释。4位首部长度和IP协议头类似，表示TCP协议头的长度，以4字节为单位，因此TCP协议头...

yincheng01 2014-09-04 07:39 1367

RTMPdump（libRTMP） 源代码分析 9： 接收消息（Message）（接收视音频数据） (/wishfly/article/details/73810744)

RTMPdump（libRTMP）源代码分析 9： 接收消息（Message）（接收视音频数据）
http://blog.csdn.net/leixiaohua1020/article/de...

wishfly 2017-06-27 17:46 458

http协议之面试题 (/ljh_learn_from_base/article/details/76777358)

Q1：什么是HTTP 、Socket、 TCP、 UDP？ HTTP ：全称是超文本传输协议，是一个应用层的协议。用于客户端和服务端之间进行通讯。 TCP/UDP ：都是传输层协议。TCP是可...

ljh_learn_from_base 2017-08-06 16:39 157

Java_io体系之PipedWriter、PipedReader简介、走进源码及示例——14 (http://810364804.iteye.com/blog/1992802)

Java_io体系之PipedWriter、PipedReader简介、走进源码及示例——14 ——管道字符输出流、必须建立在管道输入流之上、所以先介绍管道字符输出流。可以先看示例或者总结、总结写的有点Q、不喜可无视、有误的地方指出则不胜感激。 一：PipedWriter 1、类功能简介： 管道字符输出流、用于将当前线程的指定字符写入到与此线程对应的管道字符输入流

810364804 2013-12-08 18:50 57

java_集合体系之:LinkedList详解、源码及示例——04 (/crave_shy/article/details/17440835)

摘要： 本文通过对LinkedList内部存储数据的结构、LinkedList的结构图、示例、源码、多方面深入分析LinkedList的特性和使用方法。

chenghuaying 2013-12-20 15:11 6349

常见异常解析 **(http://fuyou0104.iteye.com/blog/1174579)**

ConcurrentHashMap与CopyOnWriteArrayList比较。 博客分类： Java ConcurrentHashMap ConcurrentHashMap引入了Segment，每个Segment又是一个hashtable，相当于是两级Hash表，然后锁是在Segment一级进行的，提高了并发性。缺点是对整个集合进行操作的方法如 size() 或 isEmpty()的实现很困难，基本无法得到精准的数据。Segment的read不加锁，只有在读到null的情况(一般不会有null的，只有在其他线程操作Map的时候，所以就用锁来等他操作完）下调用

fuyou0104 2011-09-18 16:29 4162

java_集合体系之HashMap详解、源码及示例——09 (/crave_shy/article/details/17552679)

摘要： 本文通过HashMap的结构图分析HashMap所具有的特性、通过源码深入了解HashMap实现原理、使用方法、通过实例加深对HashMap的应用的理解。篇幅较长、慎入！

chenghuaying 2013-12-25 14:54 2542

java_集合体系之ArrayList详解、源码及示例——03 (http://810364804.iteye.com/blog/1992789)

java_集合体系之ArrayList详解、源码及示例——03 一：ArrayList结构图 <img src="http://img.blog.csdn.net/20131220102938781?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvY3JhdmVfc2h5/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/

810364804 2013-12-20 10:56 163

java_集合体系之总体目录——00 (/crave_shy/article/details/17416791)

摘要： java集合系列目录、不断更新中、、、、水平有限、总有不足、误解的地方、请多包涵、也希望多提意见、多多讨论 ^_^

chenghuaying 2013-12-19 15:41 3210

Java_io体系之BufferedInputStream、BufferedOutputStream简介、走进源码及示例——10 (http://810364804.iteye.com/blog/1992805)

Java_io体系之BufferedInputStream、BufferedOutputStream简介、走进源码及示例——10 一：BufferedInputStream 1、类功能简介： 缓冲字节输入流、作为FilterInputStream的一个子类、他所提供的功能是为传入的底层字节输入流提供缓冲功能、他会通过底层字节输入流（in）中的字节读取到自己的buff

810364804 2013-11-28 20:37 53

java_集合体系之Vector详解、源码及示例——05 (/crave_shy/article/details/17504279)

摘要： 本文通过对Vector的结构图中涉及到的类、接口来说明Vector的特性、通过源码来深入了解Vector各种功能的实现原理、通过示例加深对Vector的理解。

chenghuaying 2013-12-23 14:40 2129

Java_io体系之OutputStreamWriter、InputStreamReader简介、走进源码及示例——17 (http://810364804.iteye.com/blog/1992797)

Java_io体系之OutputStreamWriter、InputStreamReader简介、走进源码及示例——17 一：

OutputStreamWriter 1、类功能简介： 输入字符转换流、是输入字节流转向输入字符流的桥梁、用于将输入字节流转换成输入

810364804 2013-12-10 09:51 60

Java线程池 (/nanmuling/article/details/37881089)

Java线程池 线程池编程 java.util.concurrent多线程框架---线程池编程（一） 一般的服务器都需要线程池，比如Web、FTP等服务器，不过它们一般都自己实现了线程池，比如以...

nanmuling 2014-07-16 16:44 2850