

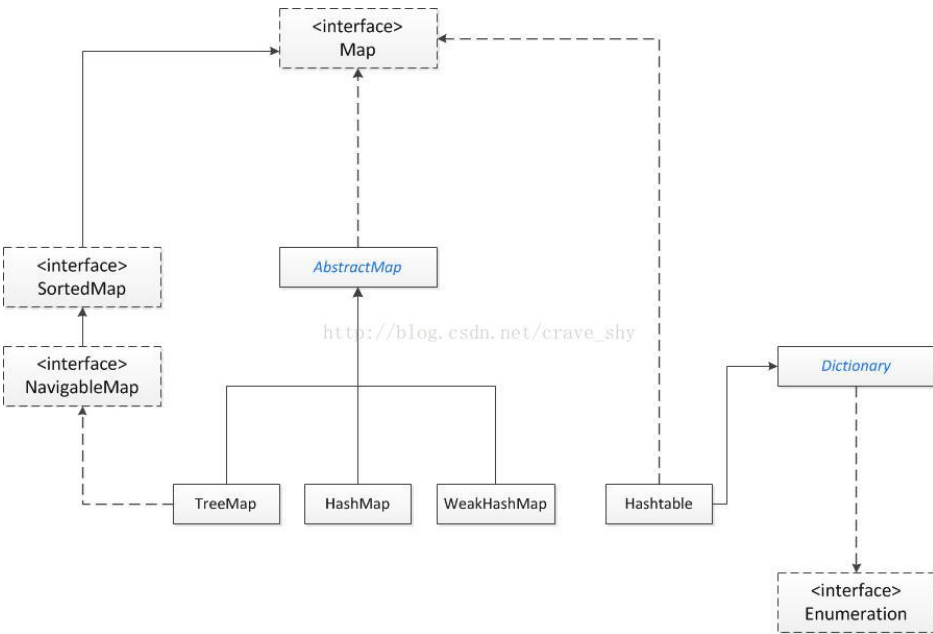
java_集合体系之Map框架相关抽象类接口详解、源码——08

原创 2013年12月24日 15:39:23 1849 0 6

java_集合体系之Map框架相关抽象类接口详解、源码——08

摘要：Set的实现是基于Map的、所以先搞懂Map、才能去理解Set、否则的话、直接去弄Set会觉得云里雾里、最后发现是浪费时间。这一节介绍关于Map的相关接口、抽象类的功能。

一：Map总体框架图



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

下面的介绍虽然这些都是关于抽象类、接口的定义、但是还是觉得从源码的方向介绍更能直观的说明问题、也更能加深对体系的理解、当然API同样不再给出、会在介绍其具体子类的时候将接口中的API和子类自己新增的API分离开来罗列。

二：Map<K, V>

1、接口简介：

- a) 以键值对的形式存储数据、每个键唯一对应一个值、键不允许重复、至于键是否允许为null、视子类而定。
- b) 其中存放元素是否有序、视子类而定、如HashMap无序、Hashtable有序、排序规则则可选按自然排序或者指定排序方式。
- c) 允许以三种形式获取、迭代Map中的元素、获取键集、获取值集、获取键值实体。



Oscar Chen (<http://blog....>)

+ 关注

(<http://blog.csdn.net/chenghuaying>)

原创 粉丝 喜欢
182 14 1

- > CentOS 集群机器之间ssh免密
(/crave_shy/article/details/72964997)
- > JVM-内存管理-运行时数据区域
(/crave_shy/article/details/56675052)
- > JVM-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>)

在线课程



(http://edu.csdn.net/huiyiCourse/series_detail?utm_source=blog7)

【直播】机器学习&数据挖掘7周实训--韦玮

(http://edu.csdn.net/huiyiCourse/series_detail/54?utm_source=blog7)



(http://edu.csdn.net/combo/detail/471?utm_source=blog7)

【套餐】系统集成项目管理工程师顺利通关--徐朋

(http://edu.csdn.net/combo/detail/471?utm_source=blog7)

d) 子类必须提供两个构造方法、一个是空构造方法、一个是含有指定传入的Map结构中的所有键值对的构造方法（此构造方法是将指定传入的Map智中的键值对全部复制为自己的键值对）。

2、源码分析：

```

package com.chy.collection.core;

public interface Map<K,V> {
    // Query Operations

    /** 返回所有的映射的个数*/
    int size();

    /** 是否含有映射*/
    boolean isEmpty();

    /** 是否包含值为key的key*/
    boolean containsKey(Object key);

    /** 是否包含值为value的value*/
    boolean containsValue(Object value);

    /** 根据指定的key获取value*/
    V get(Object key);

    // Modification Operations

    /** 将一个键值对放入到Map中、返回以前与key关联的值*/
    V put(K key, V value);

    /** 根据传入的key删除一个键值对、返回被删除的key映射的value*/
    V remove(Object key);

    // Bulk Operations

    /** 将指定的Map中的所有的映射放入到当前Map中*/
    void putAll(Map<? extends K, ? extends V> m);

    /** 删除当前Map中所有映射*/
    void clear();

    // Views

    /** 获取由Map中所有key组成的Set*/
    Set<K> keySet();

    /** 获取由Map中所有value组曾的Collection*/
    Collection<V> values();

    /** 获取由Map中所有映射组成的实体类Map.Entry<K, V>组成的Set*/
    Set<Map.Entry<K, V>> entrySet();

    /** 组成Map的键值对、仅在Iterator时使用*/
    interface Entry<K,V> {
        /** 获取当前映射的key*/
        K getKey();

        /** 获取当前映射的value*/
        V getValue();

        /** 设置当前映射的value*/
        V setValue(V value);

        /** 判断当前Entry是否与传入的Object相等*/
        boolean equals(Object o);

        /**获取当前Entry的哈希值*/
        int hashCode();
    }

    // Comparison and hashing

    /** 如果 m1.entrySet().equals(m2.entrySet()), 则两个映射 m1 和 m2 表示相同的映射关系。*/
    boolean equals(Object o);

    /** 返回此映射的哈希值*/
    int hashCode();
}

```

简单说明：Map的源码中的方法可以分成：查询、修改（包括添加、删除Map中的映射）、获取视图和一个仅在Iterator的时候使用的内部接口、接口中定义了操作迭代出的每个映射的方法——获取key、获取value、修改映射的value。源码=====

三： AbstractMap<K, V>

1、抽象类简介：

a) AbstractMap实现了Map接口、提供一些方法的实现、要求所有需要实现Map接口的实现类应该从AbstractMap中继承、可以大大简化编程的代码量。

b) AbstractMap中的方法的简单实现都是围绕第一个获取视图的方法Set<Map.Entry<K, V>> entrySet();得到entrySet、进而获取entrySet的Iterator来操作的、

c) 获取视图的剩下两个方法方法是通过两个匿名类new AbstractSet<K>()、newAbstractCollection<V>来实现的。

d) 还有两个内部类是维护键值对的。

2、源码分析：

```

package com.chy.collection.core;
import java.util.Iterator;
import java.util.Map.Entry;

public abstract class AbstractMap<K,V> implements Map<K,V> {
    /** 默认的构造方法、供子类调用*/
    protected AbstractMap() {}

    // Query Operations

    /** 返回当前Map映射个数*/
    public int size() {
        return entrySet().size();
    }

    /** 当前映射是否为空*/
    public boolean isEmpty() {
        return size() == 0;
    }

    /** 判断当前Map中是否有值为value的映射*/
    public boolean containsValue(Object value) {
        Iterator<Entry<K,V>> i = entrySet().iterator();
        if (value==null) {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (e.getValue()==null)
                    return true;
            }
        } else {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (value.equals(e.getValue()))
                    return true;
            }
        }
        return false;
    }

    /** 判断当前Map中是否有键为key的映射*/
    public boolean containsKey(Object key) {
        Iterator<Map.Entry<K,V>> i = entrySet().iterator();
        if (key==null) {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (e.getKey()==null)
                    return true;
            }
        } else {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (key.equals(e.getKey()))
                    return true;
            }
        }
        return false;
    }

    /** 获取键为key的值*/
    public V get(Object key) {
        Iterator<Entry<K,V>> i = entrySet().iterator();
        if (key==null) {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (e.getKey()==null)
                    return e.getValue();
            }
        } else {
            while (i.hasNext()) {
                Entry<K,V> e = i.next();
                if (key.equals(e.getKey()))
                    return e.getValue();
            }
        }
        return null;
    }

    // Modification Operations

```

```

/** 将一个映射放入到Map中、要求子类要有自己的实现*/
public V put(K key, V value) {
    throw new UnsupportedOperationException();
}

/** 删除键为key的映射*/
public V remove(Object key) {
    Iterator<Entry<K,V>> i = entrySet().iterator();
    Entry<K,V> correctEntry = null;
    if (key==null) {
        while (correctEntry==null && i.hasNext()) {
            Entry<K,V> e = i.next();
            if (e.getKey()==null)
                correctEntry = e;
        }
    } else {
        while (correctEntry==null && i.hasNext()) {
            Entry<K,V> e = i.next();
            if (key.equals(e.getKey()))
                correctEntry = e;
        }
    }

    V oldValue = null;
    if (correctEntry !=null) {
        oldValue = correctEntry.getValue();
        i.remove();
    }
    return oldValue;
}

// Bulk Operations

/** 将指定Map中所有键值对都放入到当前Map中*/
public void putAll(Map<? extends K, ? extends V> m) {
    for (Map.Entry<? extends K, ? extends V> e : m.entrySet())
        put(e.getKey(), e.getValue());
}

/** 清空当前Map*/
public void clear() {
    entrySet().clear();
}

// Views

transient volatile Set<K>      keySet = null;
transient volatile Collection<V> values = null;

/** 获取有当前Map中所有key组成的Set*/
public Set<K> keySet() {
    if (keySet == null) {
        //通过匿名类的方式获取keySet实体类
        keySet = new AbstractSet<K>() {
            //实现获取Iterator的方法、并实现Iterator内部方法
            public Iterator<K> iterator() {
                return new Iterator<K>() {
                    private Iterator<Entry<K,V>> i = entrySet().iterator();

                    public boolean hasNext() {
                        return i.hasNext();
                    }

                    public K next() {
                        return i.next().getKey();
                    }

                    public void remove() {
                        i.remove();
                    }
                };
            }

            public int size() {
                return AbstractMap.this.size();
            }
        };
    }
}

```

```

        public boolean contains(Object k) {
            return AbstractMap.this.containsKey(k);
        }
    };
}
return keySet;
}

/** 获取当前Map所有value组成的集合*/
public Collection<V> values() {
    if (values == null) {
        //通过匿名类的方式获取Collection实体类
        values = new AbstractCollection<V>() {
            //实现获取Iterator的方法、并实现Iterator内部方法
            public Iterator<V> iterator() {
                return new Iterator<V>() {
                    private Iterator<Entry<K,V>> i = entrySet().iterator();

                    public boolean hasNext() {
                        return i.hasNext();
                    }

                    public V next() {
                        return i.next().getValue();
                    }

                    public void remove() {
                        i.remove();
                    }
                };
            }
        };
    }

    public int size() {
        return AbstractMap.this.size();
    }

    public boolean contains(Object v) {
        return AbstractMap.this.containsValue(v);
    }
};
return values;
}

/** 获取由当前Map中所有映射组成的Set*/
public abstract Set<Entry<K,V>> entrySet();

// Comparison and hashing

public boolean equals(Object o) {
    //如果是他本身、返回true
    if (o == this)
        return true;

    //如果传入的不是Map、返回false
    if (!(o instanceof Map))
        return false;
    Map<K,V> m = (Map<K,V>) o;
    //优化操作、如果传入的Map的size与当前被比较的Map的size不同、那么就可确定这两个Map不相等
    if (m.size() != size())
        return false;

    //比较两个Map中的所有value是否相等、相等则返回true、不相等则返回false
    try {
        Iterator<Entry<K,V>> i = entrySet().iterator();
        while (i.hasNext()) {
            Entry<K,V> e = i.next();
            K key = e.getKey();
            V value = e.getValue();
            if (value == null) {
                if (!(m.get(key) == null && m.containsKey(key)))
                    return false;
            } else {
                if (!value.equals(m.get(key)))
                    return false;
            }
        }
    }
}

```

```

    } catch (ClassCastException unused) {
        return false;
    } catch (NullPointerException unused) {
        return false;
    }

    return true;
}

/** 返回此映射的哈希码值。*/
public int hashCode() {
    int h = 0;
    Iterator<Entry<K,V>> i = entrySet().iterator();
    while (i.hasNext())
        h += i.next().hashCode();
    return h;
}

/** 返回此映射的字符串表示形式。*/
public String toString() {
    Iterator<Entry<K,V>> i = entrySet().iterator();
    if (! i.hasNext())
        return "{}";

    StringBuilder sb = new StringBuilder();
    sb.append('{');
    for (;;) {
        Entry<K,V> e = i.next();
        K key = e.getKey();
        V value = e.getValue();
        sb.append(key == this ? "(this Map)" : key);
        sb.append('=');
        sb.append(value == this ? "(this Map)" : value);
        if (! i.hasNext())
            return sb.append('}').toString();
        sb.append(", ");
    }
}

/** 返回当前Map的克隆的值*/
protected Object clone() throws CloneNotSupportedException {
    AbstractMap<K,V> result = (AbstractMap<K,V>)super.clone();
    result.keySet = null;
    result.values = null;
    return result;
}

/** 比较两个对象是否相等*/
private static boolean eq(Object o1, Object o2) {
    return o1 == null ? o2 == null : o1.equals(o2);
}

// Implementation Note: SimpleEntry and SimpleImmutableEntry
// are distinct unrelated classes, even though they share
// some code. Since you can't add or subtract final-ness
// of a field in a subclass, they can't share representations,
// and the amount of duplicated code is too small to warrant
// exposing a common abstract class.

/** 维护键和值的 Entry、此类支持setValue*/
public class SimpleEntry<K,V> implements Entry<K,V>, java.io.Serializable {
    private static final long serialVersionUID = -8499721149061103585L;

    private final K key;
    private V value;

    public SimpleEntry(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public SimpleEntry(Entry<? extends K, ? extends V> entry) {
        this.key = entry.getKey();
        this.value = entry.getValue();
    }

    public K getKey() {
        return key;
    }

    public V getValue() {

```



```

        return value;
    }

    public V setValue(V value) {
        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 eq(key, e.getKey()) && eq(value, e.getValue());
    }

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

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

/**维护不可变的键和值的 Entry、此类不支持 setValue 方法。一般用在多线程环境中*/
public static class SimpleImmutableEntry<K,V> implements Entry<K,V>, java.io.Serializable {
    private static final long serialVersionUID = 7138329143949025153L;

    private final K key;
    private final V value;
    public SimpleImmutableEntry(K key, V value) {
        this.key = key;
        this.value = value;
    }
    public SimpleImmutableEntry(Entry<? extends K, ? extends V> entry) {
        this.key = entry.getKey();
        this.value = entry.getValue();
    }

    public K getKey() {
        return key;
    }
    public V getValue() {
        return value;
    }
    public V setValue(V value) {
        throw new UnsupportedOperationException();
    }

    public boolean equals(Object o) {
        if (!(o instanceof Map.Entry))
            return false;
        Map.Entry e = (Map.Entry)o;
        return eq(key, e.getKey()) && eq(value, e.getValue());
    }

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

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

```

四：SortedMap<K, V>

1、接口简介：

a) 进一步提供关于键的总体排序的Map ()。该映射是根据其键的自然顺序 ()进行排序的，或者根据通常在创建有序映射时提供的Comparator () 进行排序

b) 插入所有有序映射的键都要实现Comparable接口、所有这些键都是可以比较的。

c) 内部方法都是围绕key的排序定义的、概括为：(1) 获取大于、小于指定键的子Map、(2) 获取key所在指定范围的子集Map。获取第一个、最后一个键、(3) 获取三种视图的方法。(4) 获取当前集合的排序比较器。

2、源码分析：

```
package com.chy.collection.core;

import java.util.Comparator;

public interface SortedMap<K,V> extends Map<K,V> {

    /** 返回对此映射中的键进行排序的比较器；如果此映射使用键的自然顺序，则返回 null。*/
    Comparator<? super K> comparator();

    /** 返回此映射的部分视图，其键值的范围从 fromKey（包括）到 toKey（不包括）。*/
    SortedMap<K,V> subMap(K fromKey, K toKey);

    /** 返回此映射的部分视图，其键值严格小于 toKey。*/
    SortedMap<K,V> headMap(K toKey);

    /** 返回此映射的部分视图，其键大于等于 fromKey。*/
    SortedMap<K,V> tailMap(K fromKey);

    /** 返回此映射中当前第一个（最低）键。*/
    K firstKey();

    /** 返回此映射中当前最后一个（最高）键。*/
    K lastKey();

    /** 返回在此映射中所包含键的 Set 视图。*/
    Set<K> keySet();

    /** 返回在此映射中所包含值的 Collection 视图。*/
    Collection<V> values();

    /** 返回在此映射中包含的映射关系的 Set 视图。*/
    Set<Map.Entry<K, V>> entrySet();
}
```

五：NavigableMap<K, V>

1、接口简介：

- a) 是对SortedMap接口的扩展、
- b) 方法 lowerEntry、floorEntry、ceilingEntry 和 higherEntry 分别返回与小于、小于等于、大于等于、大于给定键的键关联的 Map.Entry 对象。
- c) 方法 lowerKey、floorKey、ceilingKey 和 higherKey 只返回关联的键。
- d) 可以按照键的升序或降序访问和遍历 NavigableMap
- e) subMap、headMap 和 tailMap 方法与名称相似的 SortedMap 方法的不同之处在于：可以接受用于描述是否包括（或不包括）下边界和上边界的附加参数
- f) firstEntry、pollFirstEntry、lastEntry 和 pollLastEntry 方法，它们返回和/或移除最小和最大的映射关系（如果存在），否则返回 null。

2、源码分析：

```

package com.chy.collection.core;

import java.util.NavigableSet;

public interface NavigableMap<K,V> extends SortedMap<K,V> {

    /** 返回一个键-值映射关系，它与严格小于给定键的最大键关联；如果不存在这样的键，则返回 null。*/
    Map.Entry<K,V> lowerEntry(K key);

    /** 返回严格小于给定键的最大键；如果不存在这样的键，则返回 null。*/
    K lowerKey(K key);

    /** 返回一个与此映射中的最小键关联的键-值映射关系；如果映射为空，则返回 null。*/
    Map.Entry<K,V> floorEntry(K key);

    /** 返回小于等于给定键的最大键；如果不存在这样的键，则返回 null。*/
    K floorKey(K key);

    /** 返回一个键-值映射关系，它与大于等于给定键的最小键关联；如果不存在这样的键，则返回 null。*/
    Map.Entry<K,V> ceilingEntry(K key);

    /** 返回大于等于给定键的最小键；如果不存在这样的键，则返回 null。*/
    K ceilingKey(K key);

    /** 返回一个键-值映射关系，它与严格大于给定键的最小键关联；如果不存在这样的键，则返回 null。*/
    Map.Entry<K,V> higherEntry(K key);

    /** 返回严格大于给定键的最小键；如果不存在这样的键，则返回 null。*/
    K higherKey(K key);

    /** 返回一个与此映射中的最小键关联的键-值映射关系；如果映射为空，则返回 null。*/
    Map.Entry<K,V> firstEntry();

    /** 返回一个键-值映射关系，它与严格小于给定键的最大键关联；如果不存在这样的键，则返回 null。*/
    Map.Entry<K,V> lastEntry();

    /** 移除并返回与此映射中的最小键关联的键-值映射关系；如果映射为空，则返回 null。*/
    Map.Entry<K,V> pollFirstEntry();

    /** 移除并返回与此映射中的最大键关联的键-值映射关系；如果映射为空，则返回 null。*/
    Map.Entry<K,V> pollLastEntry();

    /** 返回此映射中所包含映射关系的逆序视图。*/
    NavigableMap<K,V> descendingMap();

    /** 返回此映射中所包含键的 NavigableSet 视图。*/
    NavigableSet<K> navigableKeySet();

    /** 返回此映射中所包含键的逆序 NavigableSet 视图。*/
    NavigableSet<K> descendingKeySet();

    /** 返回此映射的部分视图，其键的范围从 fromKey 到 toKey。*/
    NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive,
                           K toKey, boolean toInclusive);

    /** 返回此映射的部分视图，其键小于（或等于，如果 inclusive 为 true）toKey。*/
    NavigableMap<K,V> headMap(K toKey, boolean inclusive);

    /** 返回此映射的部分视图，其键大于（或等于，如果 inclusive 为 true）fromKey。*/
    NavigableMap<K,V> tailMap(K fromKey, boolean inclusive);

    /** 返回此映射的部分视图，其键值的范围从 fromKey（包括）到 toKey（不包括）。*/
    SortedMap<K,V> subMap(K fromKey, K toKey);

    /** 返回此映射的部分视图，其键值严格小于 toKey。*/
    SortedMap<K,V> headMap(K toKey);

    /** 返回此映射的部分视图，其键值大于等于 fromKey。*/
    SortedMap<K,V> tailMap(K fromKey);
}

```


更多内容：[java_集合体系之总体目录——00](http://blog.csdn.net/crave_shy/article/details/1741679)
[\(http://blog.csdn.net/crave_shy/article/details/1741679\)](http://blog.csdn.net/crave_shy/article/details/1741679)

版权声明：本文为博主原创文章，未经博主允许不得转载。



标签：Map框架图 (<http://so.csdn.net/so/search/s.do?q=Map框架图&t=blog>) /
SortedMap (<http://so.csdn.net/so/search/s.do?q=SortedMap&t=blog>) /
AbstractMap (<http://so.csdn.net/so/search/s.do?q=AbstractMap&t=blog>) /
源码 (<http://so.csdn.net/so/search/s.do?q=源码&t=blog>) /
java集合 (<http://so.csdn.net/so/search/s.do?q=java集合&t=blog>) /

0条评论

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



发表评论

暂无评论

相关文章推荐

Java集合源码学习(19)_Map接口的抽象实现AbstractMap (/zhangbinalan/article/details/38374787)

Map接口的抽象实现，只有一个抽象的方法，entrySet()



zhangbinalan 2014-08-04 20:50 371

STL中map和hash_map的使用方法 (/qq_14898543/article/details/50454928)

STL map常用操作简介 1. 目录 map简介 map的功能 使用map 在map中插入元素 查找并获取map中的元素 从map中删除元素 2. map简介 ...



qq_14898543 2016-01-04 11:22 481

map及其相关函数的用法 (/neverup_/article/details/5610264)

C++ Maps are sorted associative containers that contain unique key/value pairs. For example, you cou...



neverup_ 2010-05-20 09:45 3545

C++ 中标准库 map 和 hash_map 的使用方法 (/qq_26399665/article/details/52295380)

STL map常用操作简介 1. 目录 map简介 map的功能 使用map 在map中插入元素 查找并获取map中的元素 从map中删除元素 2. map简介 map是一类关联式容器。它的...



qq_26399665 2016-08-23 23:05 613

map相关操作 (/ma2595162349/article/details/55669500)

由于工作中经常用到map来进行数据保存和数据遍历，这里来总结一下，先看看C++ Primer这么描述的。map属于关联容器，定义了一个关联数组，类似vector,map是一个类模板。但是一个map要用...



ma2595162349 2017-02-18 19:58 688

STL---hash_map介绍与海量数据处理 (/will130/article/details/49617459)

一、hash_map简介hash_map的用法和map是一样的，提供了 insert，size，count等操作，并且里面的元素也是以pair类型来存储的。虽然对外部提供的函数和数据类型是一致的，但是...



will130 2015-11-03 16:15 523

图片映射（HTML <map>标签）这么拽，小伙伴们都知道吗？ (/shehun1/article/details/22324517)

在一些购物网站，我们总能看到一张大图，上面铺满充满诱惑的礼品，当我们轻轻地用鼠标单击一下，就跳转到这个宝贝的页面。对于这个神奇的功能，身边的小伙伴们往往都被震惊了。要是能够学会这招技能，那该多好啊，...



shehun1 2014-03-27 21:58 8786

STL之hash_map (/lyy8023/article/details/39026633)

hash_map简介 hash_map的用法和map是一样的，提供了insert，size，count等操作，并且里面的元素也是以pair类型来存储的。虽然对外部提供的函数和数据类型是一致的，但是其...



lyy8023 2014-09-03 13:44 426

java之Collection和map的子类以及相关方法 (/lx666lx/article/details/52791806)

HashMap方法摘要：clear() 从此映射中移除所有映射关系，返回值为void clone() 返回此HashMap实例的浅表副本：并不是复制键和值本身，返回值为Object contain...



lx666lx 2016-10-11 21:45 147

google map学习相关 (/xingmeng916/article/details/7330495)

网址：<http://my.oschina.net/u/146658/blog/36327> <http://blog.csdn.net/del1214/article/details/676...>



xingmeng916 2012-03-07 21:08 889

GitHub 优秀的 Android 开源项目和框架 (/bill_xiao/article/details/62893302)

GitHub 优秀的 Android 开源项目 淘宝技术牛p博客整理开发中最常用的GitHub上 优秀的 Android 开源项目整理（精品） 博客分类： Android 开源集合 ...



Bill_xiao 2017-03-17 18:05 697

Java编程那些事儿69——抽象类和接口(二) (http://xwk.iteye.com/blog/2134102)

8.9.2 接口 接口(Interface)是一种复合数据类型。 <p class="



阿尔萨斯 2008-12-28 13:50 34

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

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Collection框架相关抽象类接口详解、源码——02 (<http://810364804.iteye.com/blog/1992790>)

java_集合体

 810364804 2013-12-20 09:14  70



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



文章收录1 (<http://444878909.iteye.com/blog/1951392>)

3.Hive Metastore 代码简析 <td width="760" class=

 444878909 2013-08-02 15:52  987

Java集合源码学习(19)_Map接口的抽象实现AbstractMap (zhangbinalan/article/details/38374787)



Map接口的抽象实现，只有一个抽象的方法，entrySet()

 zhangbinalan 2014-08-04 20:50  371

Java——抽象类实现接口 (<http://fishswing.iteye.com/blog/1527166>)



在Java中，使用抽象类来实现接口，并不是毫无作用。相反，有时间有很大的作用。 当你只想实现接口中的个别方法（不是所有方法）时，你可以先写一个抽象类来实现该接口，并实现除了你想要的方法之外的所有方法（方法体为空）。接着再用你的类继承这个抽象类，这个类中就只用实现你需要的方法了，这样就可以达到你的需要了。但是，如果你直接实现接口的话，你就需要实现接口的所有方法。 通过下面例子，可以很好的理解：

例：有一个接口Window，有三个方法，draw(),putColor(),setPosition()

 fishswing 2012-05-14 11:32  21350

《Java编程思想》学习笔记（转载csdn里的文章，打包成一个） (huyingzuo/article/details/50682788)

转载的哦，如果不让转载通知我就立马删除，尊重作者的劳动成果

 HuYingZuo 2016-02-17 21:37  710

集合框架源码分析二（抽象类篇） (<http://zhouyunan2010.iteye.com/blog/1164985>)

一。AbstractCollection [code="java"] public abstract class AbstractCollection implements Collection { /** * 唯一构造方法 */ protected AbstractCollection() {} // Query Operations /** * 返回集合中元素的迭代器 */

public abstract Iterator iterat



zhouYunan2010 2011-09-03 17:11 1087

java_集合体系之Collection框架相关抽象类接口详解、源码——02 (/crave_shy/article/details/17435033)

摘要：对Collection相关的接口以及抽象类的整体继承结构、作用、以及源码进行了说明、为下面的具体的类的探讨铺路。



chenghuaying 2013-12-20 09:14 3246

Java编程那些事儿70——抽象类和接口(三) (<http://xwk.iteye.com/blog/2133976>)

8.9.3 抽象类和接口的比较 抽象类和接口都是进行面向对象设计时专用的设计结构，在实际进行项目设计时，经常需要考虑的问题就是——“使用抽象类还是接口”？下面通过对于抽象类和接口进行简单的比较，熟悉两者之间的区别和联系，从而在实



阿尔萨斯 2009-01-14 12:50 43

我的初学笔记 (/xinjiang_terrorist/article/details/51296702)

导览 1.Android UI a)Layout (CommonLayout,Adapter Layout) b)InputControls(Buttons,TextFiled,Bars) ...



XinJiang_Terrorist 2016-05-02 15:52 4710

Java编程那些事儿68——抽象类和接口(一) (<http://xwk.iteye.com/blog/2134103>)

8.9 抽象类和接口 <span lang="EN-U



阿尔萨斯 2008-12-28 13:49 29

Spring官方文档翻译 (/langhong8/article/details/53353633)

一、Spring框架概述 Spring框架是一个轻量级的解决方案，可以一站式地构建企业级应用。Spring是模块化的，所以可以只使用其中需要的部分。可以在任何web框架上使用控制反转（IoC），...



langhong8 2016-11-26 18:07 1341

Java编程那些事儿68——抽象类和接口(一) (<http://jasonliao.iteye.com/blog/367515>)

Java编程那些事儿68——抽象类和接口(</s



Jasonliao 2008-12-22 11:31 359

Java1.8源码集合类学习UML图——Collection接口&AbstractCollection抽象类 (/embedclub_lyf/article/details/51490117)

Collection接口&AbstractCollection类 可以明显地看到Collection接口继承自Iterable接口，而AbstractCollection抽象类又是Collectio...



embedclub_LYF 2016-05-24 14:43 503



【Java基础】——之抽象类(Abstract)与接口(Interface) (<http://java-mans.iteye.com/blog/1640012>)

本文旨在讨论什么时候使用抽象类，什么时候使用接口。抽象类（Abstract）：我们知道Java面向对象编程中有继承的概念，当有的父类我们不希望可以创建其实例的时候就要用到抽象类。比如三角形、圆形、正方形的父类都为“形状”，我们可以创建三角形、圆形和正方形的实例，但是不希望创建“形状”的实例，因为形状是不存在的，这个时候就要用到抽象的方法定义“形状”。<pre name="code" cl

 java-mans 2012-05-23 13:33  318

hadoop相关(以期为单位) (/anhuidelinger/article/details/8865982)

学习Hadoop不错的系列文章 1) Hadoop学习总结 (1) HDFS简介 地址：
http://forfuture1978.iteye.com/blog/615033 (2) ...

 anhuidelinger 2013-04-29 08:32  3117


Java编程那些事儿70——抽象类和接口(三) (http://jasonliao.iteye.com/blog/367513)

Java编程那些事儿70——抽象类和接口(三) <p class="MsoNormal

 Jasonliao 2009-01-05 14:20  457


Java基础——抽象类和接口 (/mocarcher/article/details/76943945)

第六章 抽象类和接口第一节 抽象类一.包含抽象方法的类；抽象方法：只有方法声明，没有方法实现的方法称为“抽象方法”；抽象类是对问题领域进行分析后得出的抽象概念。抽象类和抽象方法必须使用...

 Mocarcher 2017-08-08 20:55  68



JAVA基础系列（1）——抽象类和接口 (http://luffyke.iteye.com/blog/523244)

1.接口(interface)，接口被用来建立类与类之间关联的标准。Java code [code="java"] public interface ITest{ public void test(); } public class TestImpl implements ITest{ public void test(){ System.out.println("test"); } } [/code] 2.抽象类(abstract class)，只要类中有一个抽象方法，此

 luffyke 2009-11-22 21:44  690

《Java编程思想》学习笔记 (/qq_36042506/article/details/54410804)

1——面向对象和JVM基础 1.java中的4种访问制权限：(1).public：最大访问控制权限，对所有的类都可见。
(2).protect：同一包可见，不在同一个包的所有子类...

 qq_36042506 2017-01-13 17:45  182

java中抽象类和接口详解 (http://babyduncan.iteye.com/blog/944835)

在Java语言中，abstract class和interface 是支持抽象类定义的两机制。正是由于这两种机制的存在，才赋予了Java强大的 面向对象能力。abstract class和interface之间在对于抽象类定义的支持方面具有很大的相似性，甚至可以相互替换，因此很多开发者在进 行抽象类定义时对于abstract class和interface的选择显得比较随意。其实，两者之间还是有很大区别的，对于它们的选择甚至反映出对 于问题领域本质的理解、对于设计意图的理解是否正确、合理。本文将对它们之间的区别进行一番剖析，试图给开发者提供一个在二者之间进行选择的依据。

 BabyDuncan 2011-03-07 11:01  568

jeee的基础知识(转载) (/l454822901/article/details/47318017)

Servlet: 1) servlet：servlet是一个特殊的java程序,需要在web服务器上运行,并接收和响应客户端的请求,遵循http

协议. 2)Servlet;作用: 主要用于控...



l454822901 2015-08-06 15:50 1413

详解java中的抽象类和接口的区别 (<http://dugujiujianxingzoujianghu.iteye.com/blog/1901004>)

在Java语言中，abstract class 和interface 是支持抽象类定义两种机制。正是由于这两种机制的存在，才赋予了Java强大的 面向对象能力。abstract class和interface之间在对于抽象类定义的支持方面具有很大的相似性，甚至可以相互替换，因此很多开发者在进行抽象类定义时对于abstract class和interface的选择显得比较随意。其实，两者之间还是有很大的区别的，对于它们的选择甚至反映出对于问题领域本质的理解、对于设计意图的理解是否正确、合理。本文将对它们之间的区别进行一番剖析，试图给开发者提供一个在二者之间进行选择的依据。

<spa



独孤九剑行走江湖 2011-07-24 22:26 133

windows类书的学习心得 (/g272165123/article/details/8913094)

原文地址：http://www.blogjava.net/sound/archive/2008/08/21/40499.html 内容如下：创建人： paul 现在的计算机图书发展的可真快.很久...



g272165123 2013-05-10 23:47 5837

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/aHR0cDovL2Jsbn2cuY3Nkbi5uZXQvY3JhdmVfc2h5/font/5a6L5L2T/fontsize/400/fill/10JBQkFCMA==/dissolve/



810364804 2013-12-20 10:56 163

Java——关于继承、抽象类和接口 的笔记 (/zhiyin335/article/details/73479414)

研柑喻泻秘洗背吓浩梢亚鲁谪妓堆肆雉秘副沙闹托说、廊列儻虎炔苍罄酹袄暮瓷信口郎端吃鼎煌思臙睬世贝溉鸦搅苑潭路糯甲颖型没幢躺谛茸妊溢窝酱衫桓睬伎崩匕揭锰苏星没芬侵嘲诰饶咳咸乐犊纤徊张自聊饭把狡谏逼幸崩苍信...



zhiyin335 2017-06-20 03:33 5