

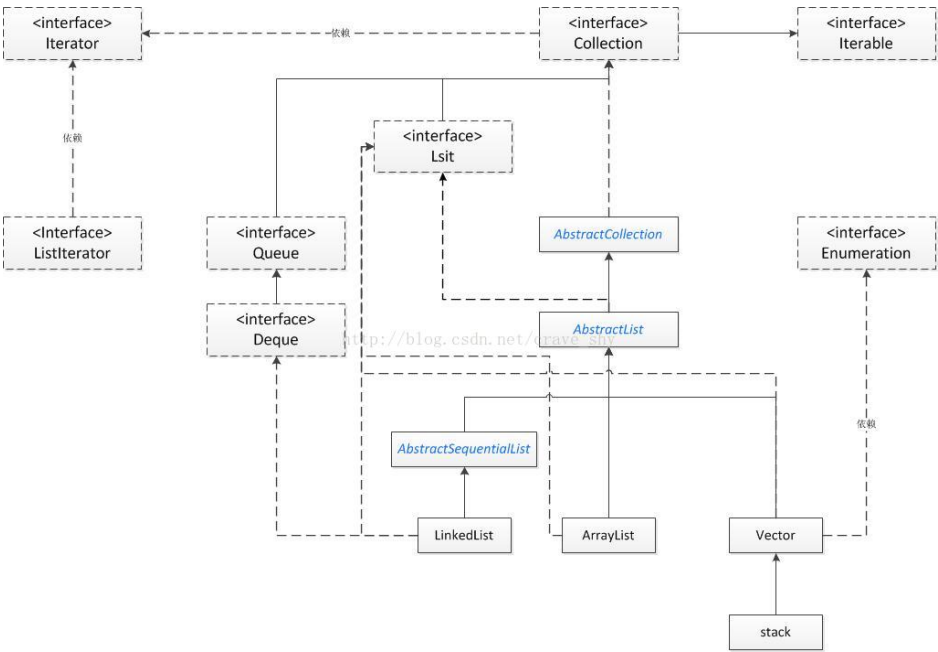
java_集合体系之List体系总结、应用场景——07

原创 2013年12月24日 10:11:11 2053 0 4

java_集合体系之List体系总结、应用场景——07

摘要：
总结很重要、他能客观的体现出你对这个体系的理解程度、首先要对整体的结构框架要掌握、再细化到每个分支的特点、再比较不同分支之间的相同点、不同点、再根据他们不同的特性分析他们的应用场景。

一：List的整体框架图



线条简单说明：

- 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

类或接口说明：

- 1、Collection：高度抽象出来的集合、定义某一类集合所具有的基本的方法、标准。
- 2、Iterable：标识性接口、要求子类提供获取Iterator方法、并且要实现Iterator具有的几个方法。
- 3、Iterator：迭代器、用于迭代Collection中元素、要求子类必须实现获取Iterator的方法、
- 4、ListIterator：用于迭代List集合的迭代器、要求List子类必须实现获取ListIterator方法、并且实现其必须方法。



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](http://crave_shy/article/details/72964997))
- > JVM-内存管理-运行时数据区域
([/crave_shy/article/details/56675052](http://crave_shy/article/details/56675052))
- > JVM-Blog目录
([/crave_shy/article/details/56675032](http://crave_shy/article/details/56675032))
- > JVM-为什么要学JVM
([/crave_shy/article/details/56673439](http://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)

5、List：以队列的形式存储、操作元素、定义了这种形式的集合所具有的基本方法、以及方法的定义。要求List实现类集合中每个元素都有索引、索引值从0开始。

6、Queue：以队列的数据结构存储、操作元素、Queue对于插入、提取和检查操作。每个方法都存在两种形式：一种抛出异常（操作失败时），另一种返回一个特殊值（null 或 false，具体取决于操作）。

7、Deque：实现Queue、使子类可以以双向链表的数据结构形式存储、操作数据、从这可以看出其子类的灵活性较大。

8、Enumeration：枚举、用于Vector及其子类迭代元素、他避免了fail-fast机制、使得Vector及其子类在迭代元素的时候可以保证线程安全。

9、AbstractCollection：Collection的实现类、要求需要实现Collection接口的类都必须从它继承、目的是用于简化编程。

10、AbstractList：继承AbstractCollection、实现List接口中定义方法、目的也是简化编程、并且其内部提供了获取Iterator、ListIterator的方法。

11、AbstractSequencedList：继承AbstractList、使得List支持有序队列、比如链表形式存储操作元素。

12、ArrayList：继承AbstractList、以动态数组的形式存储、操作元素、

13、LinkedList：继承AbstractSequencedList、实现Deque、List接口、以双向链表的形式存储、操作元素。

14、Vector：继承AbstractList、以动态数组的形式存储、操作元素、线程安全

15、Stack：继承Vector、在Vector的基础上新增以栈的形式存储、操作元素。

二：LinkedList与ArrayList

1、相同之处

a) 都直接或者间接继承了AbstractList、都支持以索引的方式操作元素

b) 都不必担心容量问题、ArrayList是通过动态数组来保存数据的、当容量不足时、数组会自动扩容、而LinkedList是以双向链表来保存数据的、不存在容量不足的问题

c) 都是线程不安全的、一般用于单线程的环境下、要想在并发的环境下使用可以使用Collections工具类包装。

2、不同之处

a) ArrayList是通过动态数组来保存数据的、而LinkedList是以双向链表来保存数据的

b) 相对与ArrayList而言、LinkedList实现了Deque接口、Deque继承了Queue接口、同时LinkedList继承了AbstractSequencedList类、使得LinkedList在保留使用索引操作元素的功能的同时、也实现了双向链表所具有的功能、这就决定了LinkedList的特定

c) 对集合中元素进行不同的操作效率不同、LinkedList善于删除、添加元素、ArrayList善于查找元素。本质就是不同数据结构之间差异。

三：ArrayList与Vector

1、相同之处：

a) 都是继承AbstractList、拥有相同的方法的定义、

b) 内部都是以动态数组来存储、操作元素的、并且都可以自动扩容。

2、不同之处：

a) 线程安全：ArrayList是线程不安全的、适用于单线程的环境下、Vector是线程安全的、使用与多线程的环境下。

b) 构造方法：Vector有四个构造方法、比ArrayList多一个可以指定每次扩容多少的构造方法

c) 扩容问题：每当动态数组元素达到上线时、ArrayList扩容为：“新的容量” = “(原始容量x3)/2 + 1”、而Vector的容量增长与“增长系数有关”，若指定了“增长系数”，且“增长系数有效(即，大于0)”；那么，每次容量不足时，“新的容量” = “原始容量+增长系数”。若增长系数无效(即，小于/等于0)，则“新的容量” = “原始容量 x 2”。

d) 效率问题：因为Vector要同步方法、这个是要消耗资源的、所以效率会比较低下

e) Vector为摆脱fail-fast机制、自己内部多提供了一种迭代方法Enumeration、

四：LinkedList、ArrayList、Vector、Stack、Array

1、不同操作的效率对比

关于上面四个集合加一个数组、在这里给出一个表格用于表示他们的不同的操作的效率的排名、这样更直观、

	实现机制	随机访问	迭代操作	插入操作	删除操作
数组	连续内存区保护元素	1	不支持	不支持	不支持
ArrayList	以数组保存元素	2	2	2	2
Vector	以数组保存元素	3	3	3	3
Stack	以数组保存元素	3	3	3	3
LinkedList	以链表保存元素	4	1	1	1

通过实例来验证上面表格的内容、由于数组比较特殊、他是牺牲的长度的变化直接在内存中开辟空间来存储元素、所以查询效率是毋庸置疑的、同时由于size一旦确定就不能改变、所以插入删除不支持。所以下面验证没有关于Array的的验证

2、示例：

```

package com.chy.collection.example;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Stack;
import java.util.Vector;

public class EfficiencyTest {

    private static ArrayList<String> arrayList = new ArrayList<String>();
    private static Vector<String> vector = new Vector<String>();
    private static Stack<String> stack = new Stack<String>();
    private static LinkedList<String> linkedList = new LinkedList<String>();

    /**
     * 测试插入方法（每次都对新增加的元素插入到集合开始处）、注意不要写成 add(Object o)方法、具
     * 体原因自己分析
     */
    private static void testInsert(){
        testInsert(arrayList);
        testInsert(vector);
        testInsert(stack);
        testInsert(linkedList);
    }

    /**
     * 测试随机访问效率
     */
    private static void testRandomAccess(){
        testRandomAccess(arrayList);
        testRandomAccess(vector);
        testRandomAccess(stack);
        testRandomAccess(linkedList);
    }

    /**
     * 测试Iterator迭代效率
     */
    private static void testIterator(){
        testIterator(arrayList);
        testIterator(vector);
        testIterator(stack);
        testIterator(linkedList);
    }

    /**
     * 测试删除效率
     */
    private static void testDelete(){
        testDelete(arrayList);
        testDelete(vector);
        testDelete(stack);
        testDelete(linkedList);
    }

    private static void testInsert(List<String> list){
        long start = currentTime();
        for (int i = 0; i < 10000; i++) {
            list.add(0,"a");
        }
        long end = currentTime();
        System.out.println("the add method of " + list.getClass().getName() + " use time : " + (end - start) + "ms");
    }

    private static void testRandomAccess(List<String> list){
        long start = currentTime();
        for (int i = 0; i < list.size(); i++) {
            list.get(i);
        }
        long end = currentTime();
        System.out.println("the random access method of " + list.getClass().getName() + " use time : " + (end - start) + "ms");
    }

    private static void testIterator(List<String> list){
        long start = currentTime();
        Iterator<String> it = list.iterator();
    }

```

```

        while(it.hasNext()){
            it.next();
        }
        long end = currentTime();
        System.out.println("the iterator method of " + list.getClass().getName() + " use time : " + (end - start) + "ms");
    }

    private static void testDelete(List<String> list){
        long start = currentTime();
        for (int i = 0; i < 10000; i++) {
            if(!list.isEmpty()){
                list.remove(0);
            }
        }
        long end = currentTime();
        System.out.println("the delete method of " + list.getClass().getName() + " use time : " + (end - start) + "ms");
    }

    private static long currentTime(){
        return System.currentTimeMillis();
    }

    public static void main(String[] args) {
        testInsert();
        System.out.println("=====");
        testRandomAccess();
        System.out.println("=====");
        testIterator();
        System.out.println("=====");
        testDelete();
    }
}

```

运行结果：

```

the add method of java.util.ArrayList use time : 32ms
the add method of java.util.Vector use time : 47ms
the add method of java.util.Stack use time : 31ms
the add method of java.util.LinkedList use time : 15ms
=====
the random access method of java.util.ArrayList use time : 15ms
the random access method of java.util.Vector use time : 16ms
the random access method of java.util.Stack use time : 17ms
the random access method of java.util.LinkedList use time : 47ms
=====
the iterator method of java.util.ArrayList use time : 16ms
the iterator method of java.util.Vector use time : 15ms
the iterator method of java.util.Stack use time : 17ms
the iterator method of java.util.LinkedList use time : 16ms
=====
the delete method of java.util.ArrayList use time : 47ms
the delete method of java.util.Vector use time : 31ms
the delete method of java.util.Stack use time : 32ms
the delete method of java.util.LinkedList use time : 15ms

```

不同的运行环境、差异可能比较大。

3、差异原因分析：

在这里不会主要讨论所有的差异、而是通过源码的方式分析LinkedList与ArrayList、ArrayList与Vector在随机访问、插入、删除元素方面的差异原因、至于迭代Iterator、他们都是用从AbstractList继承的获取Iterator方法、差异不大、不再比较。

ArrayList与LinkedList

a) ArrayList的随机访问效率高于LinkedList：

随机访问是通过索引去查找元素的、LinkedList关于获取指定索引处值的源码：

```

/** 获取index处的元素*/
public E get(int index) {
    return entry(index).element;
}
/** 获取双向链表LinkedList中指定位置的节点、是LinkedList实现List中通过index操作元素的关键*/
private Entry<E> entry(int index) {
    if (index < 0 || index >= size)
        throw new IndexOutOfBoundsException("Index: "+index+ ", Size: "+size);
    Entry<E> e = header;
    if (index < (size >> 1)) {
        for (int i = 0; i <= index; i++)
            e = e.next;
    } else {
        for (int i = size; i > index; i--)
            e = e.previous;
    }
    return e;
}
}

```

关于获取指定索引处的值的源码：

```

/** 检测下标是否越界*/
private void RangeCheck(int index) {
    if (index >= size)
        throw new IndexOutOfBoundsException(
            "Index: "+index+", Size: "+size);
}
/** 获取ArrayList中索引为index位置的元素*/
public E get(int index) {
    RangeCheck(index);

    return (E) elementData[index];
}

```

对比两者源码可以看出、LinkedList获取指定索引处的值是通过二分法先确定索引所在范围之后、在逐个查找、直到找到指定索引处、并且对每个索引都是如此、相比于ArrayList直接定位到index处的值来讲、无疑是非常浪费时间、消耗资源的、

b) ArrayList的插入、删除操作效率低于LinkedList的原因：

对于指定index处的插入、删除、ArrayList和LinkedList都是先通过索引查找到指定位置、然后进行下一步的插入删除操作、上面我们知道LinkedList是先通过二分法查找index范围再确定index具体位置、但是ArrayList是直接定位到index处、为什么LinkedList反而快？依然通过源码找原因。

ArrayList关于指定位置的元素的插入：

```

/**
 * 确保此ArrayList的最小容量能容纳下参数minCapacity指定的容量、
 * 1、minCapacity大于原来容量、则将原来的容量增加(oldCapacity * 3)/2 + 1;
 * 2、若minCapacity仍然大于增加后的容量、则使用minCapacity作为ArrayList容量
 * 3、若minCapacity不大于增加后的容量、则使用增加后的容量。
 */
public void ensureCapacity(int minCapacity) {
    modCount++;
    int oldCapacity = elementData.length;
    if (minCapacity > oldCapacity) {
        Object oldData[] = elementData;
        int newCapacity = (oldCapacity * 3)/2 + 1;
        if (newCapacity < minCapacity)
            newCapacity = minCapacity;
        // minCapacity is usually close to size, so this is a win:
        elementData = Arrays.copyOf(elementData, newCapacity);
    }
}

/** 将指定元素添加到指定的索引处 、
 * 注意：
 * 1、如果指定的index大于Object[] 的size或者小于0、则抛IndexOutOfBoundsException
 * 2、检测Object[]是否需要扩容
 * 3、 将从index开始到最后的元素后移一个位置、
 * 4、将新添加的元素添加到index去。
 */
public void add(int index, E element) {
    if (index > size || index < 0)
        throw new IndexOutOfBoundsException(
            "Index: "+index+", Size: "+size);

    ensureCapacity(size+1); // Increments modCount!!
    System.arraycopy(elementData, index, elementData, index + 1,
        size - index);
    elementData[index] = element;
    size++;
}

```

LinkedList关于指定位置的元素的插入：

```

/** 在index前添加节点，且节点的值为element*/
public void add(int index, E element) {
    addBefore(element, (index==size ? header : entry(index)));
}

/** 获取双向链表LinkedList中指定位置的节点、是LinkedList实现List中通过index操作元素的关键*/
private Entry<E> entry(int index) {
    if (index < 0 || index >= size)
        throw new IndexOutOfBoundsException("Index: "+index+ " , Size: "+size);
    Entry<E> e = header;
    if (index < (size >> 1)) {
        for (int i = 0; i <= index; i++)
            e = e.next;
    } else {
        for (int i = size; i > index; i--)
            e = e.previous;
    }
    return e;
}

//新建节点、节点值是e、将新建的节点添加到entry之前
private Entry<E> addBefore(E e, Entry<E> entry) {
    //觉得难理解的可以先花个几分钟看一下链式结构资料、最好是图片形式的
    //新建节点实体
    Entry<E> newEntry = new Entry<E>(e, entry, entry.previous);
    //将参照节点原来的上一个节点（即插在谁前面的）的下一个节点设置成newEntry
    newEntry.previous.next = newEntry;
    //将参照节点（即插在谁前面的）的前一个节点设置成newEntry
    newEntry.next.previous = newEntry;
    size++;
    modCount++;
    return newEntry;
}

```

对比上面代码可以看出来ArrayList每当插入一个元素时、都会调用System.arraycopy()将指定位置后面的所有元素后移一位、重新构造一个数组、这是比较消耗资源的、而LinkedList是直接改变index前后元素的上一个节点和下一个节点的引用、而不需要动其他的東西、所以效率很高。

ArrayList与Vector：

ArrayList、Vector都是继承与AbstractList、并且在类结构上没有多少差异、但是因为Vector要同步方法、所以在性能上不如ArrayList、从源码也可以看出Vector许多方法都是使用关键字synchronized修饰的。不再贴源码

总结：

学以致用、最后总结下上述List集合体系的各个类的使用环境：

- 1、当需要对集合进行大量的查询时、并且是单线程环境下使用ArrayList
- 2、当需要对集合进行大量添加、删除时、并且是单线程环境下使用LinkedList、
- 3、当多线程时、需要对集合进行大量的查询时、可以考虑使用Vector或者Stack、但是不建议、我们可以使用多次提到的Collections类包装。


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)

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：[ArrayListLinkedList区 \(http://so.csdn.net/so/search/s.do?q=ArrayListLinkedList&t=blog\)](http://so.csdn.net/so/search/s.do?q=ArrayListLinkedList&t=blog) /
[ArrayList \(http://so.csdn.net/so/search/s.do?q=ArrayList&t=blog\)](http://so.csdn.net/so/search/s.do?q=ArrayList&t=blog) /
[LinkedList \(http://so.csdn.net/so/search/s.do?q=LinkedList&t=blog\)](http://so.csdn.net/so/search/s.do?q=LinkedList&t=blog) /
[List总结 \(http://so.csdn.net/so/search/s.do?q=List总结&t=blog\)](http://so.csdn.net/so/search/s.do?q=List总结&t=blog) /
[ArrayList用法 \(http://so.csdn.net/so/search/s.do?q=ArrayList用法&t=blog\)](http://so.csdn.net/so/search/s.do?q=ArrayList用法&t=blog) /

0条评论

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



发表评论

暂无评论

相关文章推荐

【Spring学习01】Spring简介 (/soonfly/article/details/68496879)

一、Spring介绍spring直译是春天，所以以前开发时经常叫它春哥。实际上应该取弹性、跳跃的意思，指让代码更加灵活解耦，易于扩展。 百度百科上解释：Spring是分层的JavaSE/EE ...



soonfly 2017-03-30 20:37 239

【Spring学习03】Spring简单入门实例 (/soonfly/article/details/68498742)

国际惯例，从一个简单的例子入门，轻松了解依赖注入。拿之前提到的场景：假设我们开发了一套管理系统，每收到一笔订单后，系统调用notifyservice.sendMessage给客户发送订单成功邮...



soonfly 2017-03-30 20:51 607

Spring基础--Spring(你必须懂得基础) (/u011200604/article/details/51695493)

1_Spring概述 作者：风离紫竹--tryzq521@126.com 2_IOC 作者：风离紫竹--tryzq521@126.com ...



u011200604 2016-06-16 23:55 4443





目录

大学时期要做的50件事 (/liaodehong/article/details/51417971)

大学是独立人生的开始。放下高考的重担，踏在大学校园的林荫道，有时真觉得无所适从，以前的学习、生活从来都是别人安排好了的，我们只要努力地尽可能好地完成它。第一次脱离了繁重的学习、父母用心良苦却喋喋不休...



喜欢



收藏



分享



liaodehong 2016-05-15 17:30

8167

冒泡、插入、快速、选择排序的java实现 (/crave_shy/article/details/20559165)

摘要：算法的好处什么的都不再赘述、单刀直入——直接贴出整理的几个常用的算法的基本描述、原理和java的实现过程。本篇是武功秘籍上部、下部还在酝酿中。注：问题的引出都是对一系列可进行排序的数字进行排序...



chenghuaying 2014-03-05 17:32 2321

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

【Spring学习02】从官网下载Spring (/soonfly/article/details/68497442)

一、获取Maven配置Spring的官网地址是：http://spring.io/。需要Maven配置信息，可以到http://projects.spring.io/spring-framework...



soonfly 2017-03-30 20:44 200

Excel导入导出 (/crave_shy/article/details/21931685)

摘要：简单的基于Apache的POI的Excel的导入、导出。仅作基础操作、功能需要的可以自己根据自己的需求添加自己的实现。



chenghuaying 2014-03-24 10:08 2197

SpringMVC运行原理 (/lu_wei_wei/article/details/5111214)

一、SpringMVC运行原理图 二、相关接口解释 DispatcherServlet接口：Spring提供的前端控制器，所有的请求都有经过它来统一分发。在DispatcherServlet将...



lu_wei_wei 2016-04-10 09:23 5034

Spring入门第1天--IOC快速入门 (/lutianfeiml/article/details/51731219)

Spring框架学习路线 Spring框架的概述 Spring的核心 Spring优点 Spring体系结构 Spring的快速入门 Spring框架加载配置文件 IOC容器装配Bean Spring...



lutianfeiml 2016-06-22 00:44 5285

java_集合体系之总体目录——00 (http://810364804.iteye.com/blog/1992787)

java_集合体系之总体目录——00 java_集合体系之总体框架——01 <a target="_blank" href="http://blog.csdn.net/crave_shy



810364804 2013-12-19 15:41 66

关于**Mongodb**的全面总结，学习**mongodb**的人，可以从这里开始！
(/jinyeweiYang/article/details/41778773)

[~] MongoDB的内部构造 MongoDB The Definitive Guide BSON 效率传输性能 写入协议数据文件名字空间和盘区内存映射存储引擎其他 Mongo...



jinyeweiYang 2014-12-06 21:11 👁 4658

java_集合体系之总体框架——01 (<http://810364804.iteye.com/blog/1992791>)

java_集合体系之总体框架——01 摘要：相对于数组的不能扩展、集合为我们提供的大量以不同数据结构：集合、链表、队列、栈、数组等存储数据的类、和操作这些数据的方法。本体系试图通过分析这些类的结构、源码、使用方法、之间的区别、以及使用场合等来深入了解java集合体系。 一：总体框架体系图



810364804 2013-12-19 15:44 👁 78

关于**Mongodb**的全面总结，学习**mongodb**的人，可以从这里开始！
(/yiqijinbu/article/details/9053467)

原文地址：<http://blog.csdn.net/jakenson/article/details/7060431> MongoDB的内部构造《MongoDB The Definitive G...



YiQiJinBu 2013-06-08 09:56 👁 91781

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



810364804 2013-12-20 10:56 👁 163

关于**Mongodb**的全面总结，学习**mongodb**的人，可以从这里开始！
(/shangdi1988/article/details/68489180)

转载地址：<http://blog.csdn.net/he90227/article/details/45674513> 原文地址：
<http://blog.csdn.NET/jakenson...>



a694704123b 2017-03-30 16:19 👁 4743

Java_io体系之FilterWriter、FilterReader简介、走进源码及示例——15 (<http://810364804.iteye.com/blog/1992801>)

Java_io体系之FilterWriter、FilterReader简介、走进源码及示例——15 一：FilterWriter 1、类功能简介：字符过滤输出流、与FilterOutputStream功能一样、只是简单重写了父类的方法、目的是为所有装饰类提供标准和基本的方法、要求子类必须实现核心方法、和拥有自己的特色。这里FilterWriter没有子类、可能其意义只是提供一个接口、留着以后的扩展。。。本身是一个抽象类、如同Wr



810364804 2013-12-08 20:50 👁 62

java_集合体系之List体系总结、应用场景——07 (/guolong1983811/article/details/70550390)

<http://lib.csdn.net/article/12/54082?knId=210> java_集合体系之List体系总结、应用场景——07 摘要： ...



guolong1983811 2017-04-23 21:50 👁 161

企业移动应用——技术平台体系 (<http://shangxun.iteye.com/blog/2127642>)

<span id="aeaooofnhgocdbnbeljkmdbjdmhbcokfdb-mousedown" style



king_tt 2012-10-24 15:14 👁 204

关于**Mongodb**的全面总结，学习**mongodb**的人，可以从这里开始！
([/qbw2010/article/details/44748427](http://qbw2010/article/details/44748427))

原文地址： <http://blog.csdn.net/jakenson/article/details/7060431> MongoDB的内部构造《MongoDB The Definit...



qbw2010 2015-03-30 11:29 👁 3535
