[Java 集合框架与数据结构 3](#_Toc18357)

[1.1 Jdk集合框架分析 4](#_Toc1042)

[1.1.1 JDK集合框架类图 4](#_Toc7948)

[1.1.2 数据结构 5](#_Toc4316)

[1.2 javolution框架分析 6](#_Toc6749)

[1.2.1 集合类图 7](#_Toc11132)

[1.2.1 源码分析 7](#_Toc20460)

1、数据结构，研究学习对比几个知名集合框架：谷歌Guava (java.util.Collections)，sun (jdk中的java.util.Collections)，以及一个实时的框架Javolution，三个集合框架的对比。  
2、源码，3种集合框架的常用数据结构的用法比较，测试查10万数据的速度，随机查询速度，内存占用的大小不同。

javolution则是比较知名的一个解决Java实时性的方案

Java有实时性规范

实时性，集合框架，围绕它的性能，案例，最优用法。

有没比Hash方式更快的 字符串查找的数据结构？

堆排序是操作数组么？为什么排序中，堆排序会比较重要，它比选择排序好在哪里？？

http://blog.csdn.net/yutianzuijin/article/details/22876017

https://www.zhihu.com/question/27064078/answer/35156109

3 Javolution 集合类图 与guava不变集合进行对比

<http://outofmemory.cn/java/guava/Collections/Immutable>

<https://github.com/google/guava/wiki/NewCollectionTypesExplained#bimap>

<https://github.com/google/guava/wiki/ImmutableCollectionsExplained>

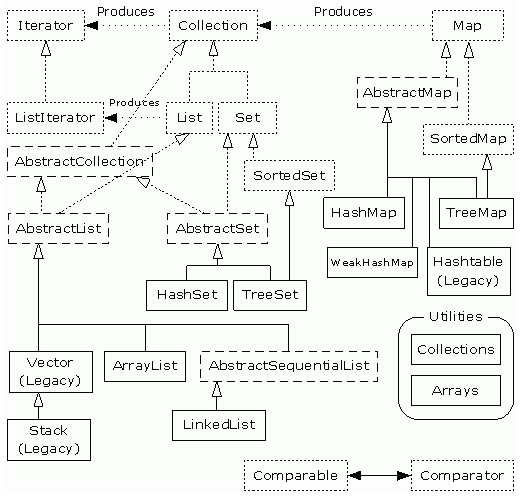
# Java 集合框架与数据结构

<https://dzone.com/articles/biglist-scalable-high>

<http://www.importnew.com/7528.html>

## 1.1 Jdk集合框架分析

### 1.1.1 JDK集合框架类图



首先从数据结构层面来理解整个类图，线性数据结构中两种最重要的两种数据结构就是数组与链表，List列表即是该数据结构的抽象。

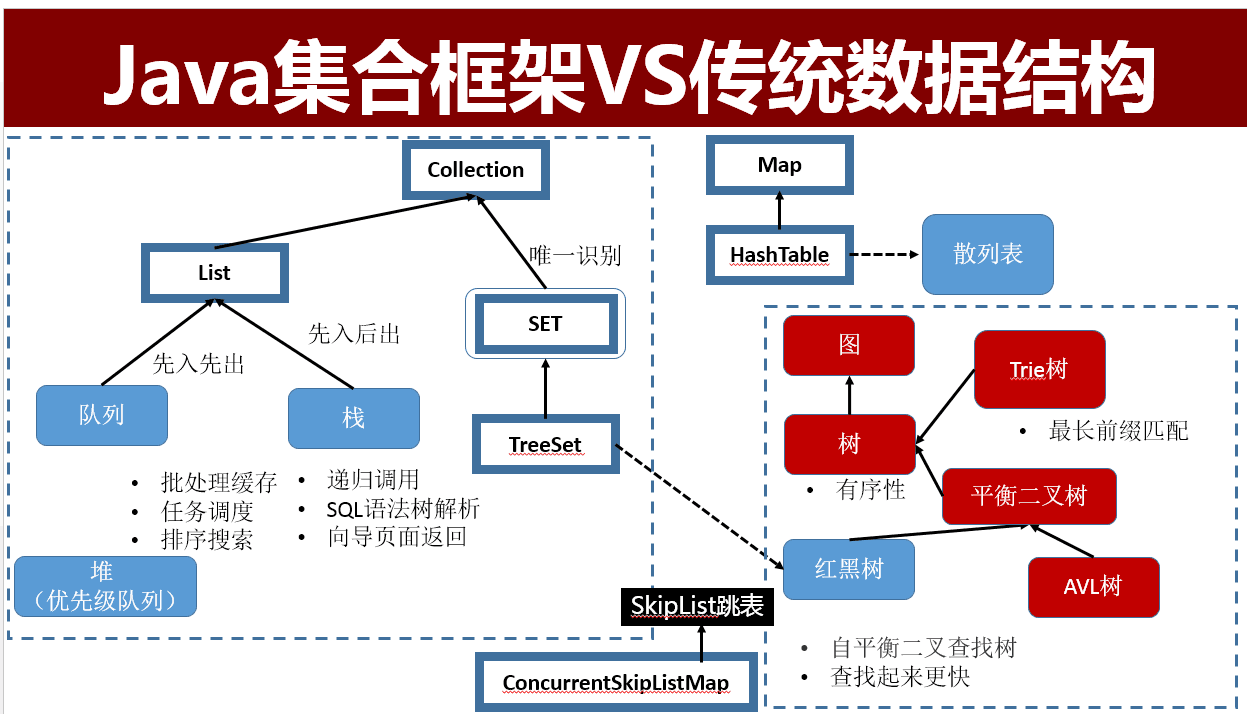
实现List接口的最终工具类ArrayList、Vetor、Stack、LinkedList前三者是数组，后者则是链表。

数组排序算法？

数组在内存里是怎样分配的？连续分配空间。那如何让里面存储的元素也是连续的呢？

ArrayList 与Vetor区别与应用场景？

### 1.1.2 数据结构



为什么树不容易实现高并发

## 1.3 guava 学习

### 1.3.1 基础工具类

#### 1.3.1.1 前置检查

<http://ifeve.com/google-guava-using-and-avoiding-null/>

<http://code.google.com/p/guava-libraries/wiki/PreconditionsExplained>

方法不能处理Null ，就快速失败拒绝null，尤其是底层工具类，若业务上层调用若返回Null从性能方面更具优势，

前置检查增加代码可读性，避免很多的if判断.

花费部分性能，针对性能开销进行举例说明

验证框架分为：1、主要对接口参数进行统一前置检查，入参可以是对象级别，可以用于普通参数。

2、业务级别条件检查，如何将业务条件检查与代码耦合度更低

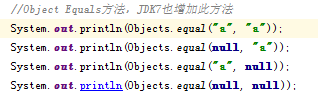
调研是否引入检查框架checker freamwork，结合现有业务进行验证梳理

#### 1.3.1.2 排序

#### 1.3.1.3 Object

##### 1.3.1.3.1 Equals

Equals使用[Objects.equal](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/base/Objects.html" \l "equal(java.lang.Object, java.lang.Object))帮助你执行null敏感的equals判断，从而避免抛出NullPointerException.



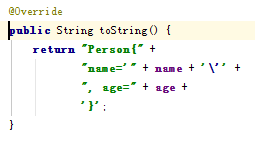
##### 1.3.1.3.2 HashCode

Hashcode,JDK 7也引入此方法

Objects.*hashCode*(**"1"**,12,**"test"**)

##### 1.3.1.3.3 toString

传统的实现：

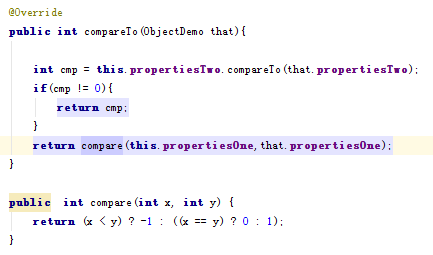


Guava

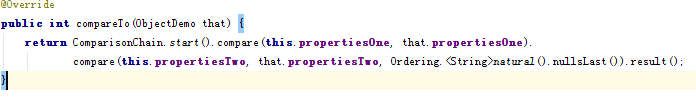
MoreObjects.*toStringHelper*(**this**.getClass()).add(**"propertiesOne"**,**propertiesOne**).add(**"propertiesTwo"**,**propertiesTwo**).toString();

传统若是手动编写非常麻烦

##### 1.3.1.3.4 compare/compareTo



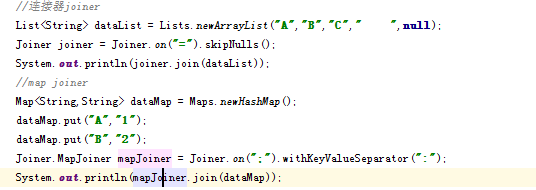
ComparisonChain执行一种懒比较：它执行比较操作直至发现非零的结果，在那之后的比较输入将被忽略, 这种[Fluent接口](http://en.wikipedia.org/wiki/Fluent_interface)风格的可读性更高，发生错误编码的几率更小，并且能避免做不必要的工作.



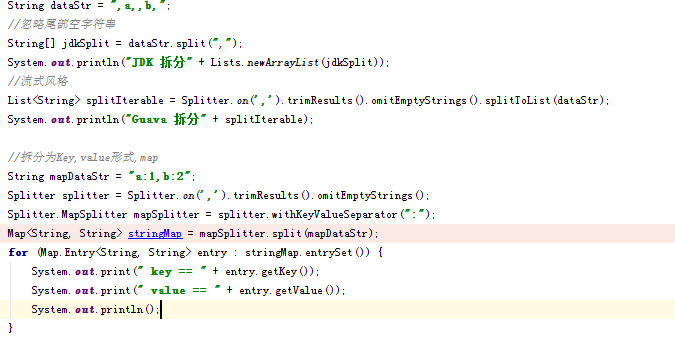
#### 1.3.1.4 Strings

字符串工具，包括分割、连接、填充等操作.

连接器：



Split

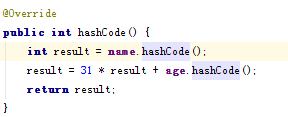


#### 1.3.1.5 Hash

##### 1.3.1.5.1 散列

Object默认的hashcode，返回的是对象的内部地址转换得到的int，可以看看Object的hashcode方法说明。不同的JVM实现不一样。

默认实现

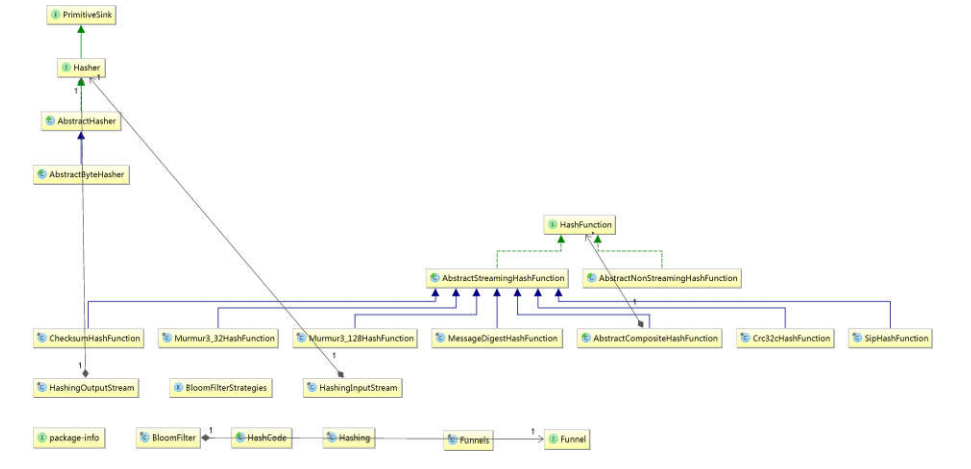


为什么选择31，参见Effective java？

从以上HashCode的实现可以看出数据和散列耦合。

对象内部地址会有什么问题呢？

Guava散列函数主要API

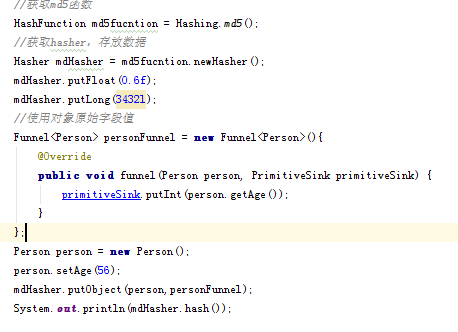


Hashing包含HashFunction相关的静态方法以及散列相关的工具方法

HashFunction是一个单纯的（引用透明的）、无状态的方法，它把任意的数据块映射到固定数目的位指，并且保证相同的输入一定产生相同的输出，不同的输入尽可能产生不同的输出。

[Hasher](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/hash/Hasher.html)

HashFunction的实例可以提供有状态的[Hasher](http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/hash/Hasher.html)，Hasher提供了流畅的语法把数据添加到散列运算，然后获取散列值。Hasher可以接受所有原生类型、字节数组、字节数组的片段、字符序列、特定字符集的字符序列等等，或者任何给定了Funnel实现的对象。



##### 1.3.1.5.2 布隆过滤器

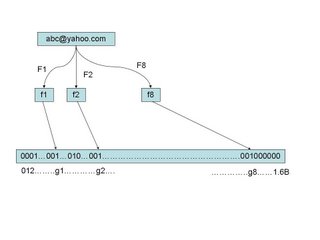
如何实现快速判断元素是否存在？

1. 二分查找
2. Hash table,如何高效解决碰撞？数据量大了存储效率如何解决？

关注暴雪Hash函数实现，如何解决上述问题

1. 布隆过滤器

由一个很长的二进制向量和一系列随机映射的函数组成，通过多个Hash函数将一个元素映射到一个Bit Array中的多个点，查询的时候仅当所有的映射点都为1才能判断元素存在于集合内；BF用于检索一个元素是否在一个集合中，记忆集合求交集；优点是空间和时间效率都超过一般查询算法，缺点是有一定的误判概率和删除困难。



<http://billmill.org/bloomfilter-tutorial/>

Google 黑板报<http://googlechinablog.blogspot.com/2007/07/bloom-filter_7469.html>

主要用于垃圾邮件过滤，大数据元素判断。

### 1.3.2 集合

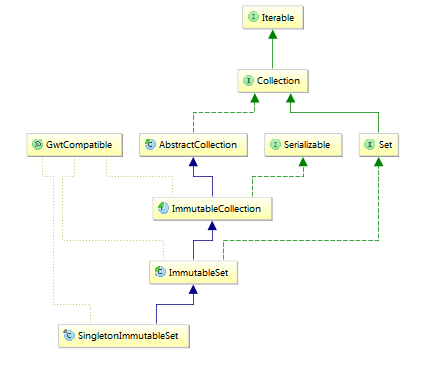
#### 1.3.1 Immutable

不变集合：

1. 多线程下不存在竟态条件
2. 节约时间和空间，不需要考虑变化
3. 第三方不信任库调用时

创建对象的不可变拷贝是一项很好的防御性编程技巧

Immutable 实现类图



与JDK Collections.unmodifiableCollection区别：

1. 防御性copy，无论原集合怎样改变，经过ImmutableCollections.copyOf()方法返回的集合，无论原集合怎样变化，新集合都不会在变化，而Collections.unmodifiableCollection()与之相反，通过源码可以得知前者是copy 且new 新元素，后者是返回一个不可变的集合引用，数据集还是指向原始数据集引用。
2. Collections.unmodifiableCollection()修饰后的集合，仍然具有原集合的特性，而不是将集合转化为常量
3. 扩容算法不一样

#### multiset

Guava提供了一个新集合类型 [Multiset](http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/collect/Multiset.html)，它可以多次添加相等的元素，从提供的API来看当把Multiset看成普通的Collection时，它表现得就像无序的ArrayList：

add(E)添加单个给定元素

iterator()返回一个迭代器，包含Multiset的所有元素（包括重复的元素）

size()返回所有元素的总个数（包括重复的元素）

当把Multiset看作Map<E, Integer>时，它也提供了符合性能期望的查询操作：

count(Object)返回给定元素的计数。

entrySet()返回Set<Multiset.Entry<E>>，和Map的entrySet类似。

elementSet()返回所有不重复元素的Set<E>，和Map的keySet()类似。

所有Multiset实现的内存消耗随着不重复元素的个数线性增长。



### 限流

<http://www.cnblogs.com/LBSer/p/4083131.html>

<http://www.it610.com/article/1256236.htm>

<http://www.it610.com/article/2299243.htm>

<http://www.jianshu.com/p/0d7ca597ebd2>

### 1.3.4 事件总线

### 1.3.5 缓存

## 1.4 javolution框架分析

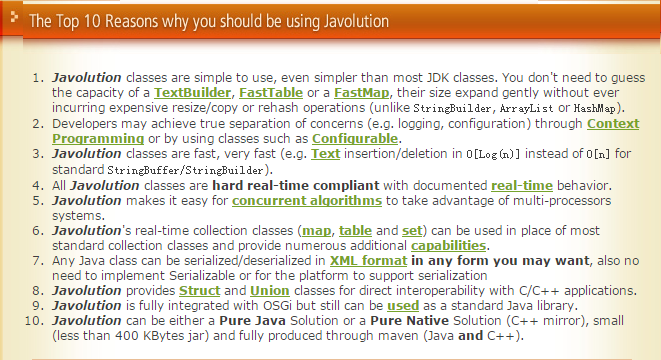
Javolution 提供一个高性的Java集合（collection ）类库和一些实用的工具类。虽然这个类包只提供非常少的几个集合类，但是这些类就能够代替大部分java.util类。javolution可以让你的应用程序更加快速和更实时。

Javolution 主要用在对实时性要求很强的应用中，对集合操作的耗时更加可预测

Javolution高性能实时框架，具体介绍参照官方网址

<http://wjm0729.iteye.com/blog/1561491>

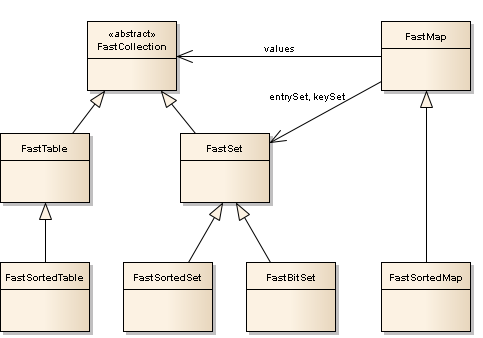
<http://javolution.org/>



实时性是什么？java rts,rtsj

<http://www.oracle.com/technetwork/cn/java/javase/tech/index-jsp-139921-zhs.html>

### 1.4.1 集合类图



### 1.4.1 源码分析

#### 1.4.1.1 FastTable

1.2.1.1 用法与性能对比

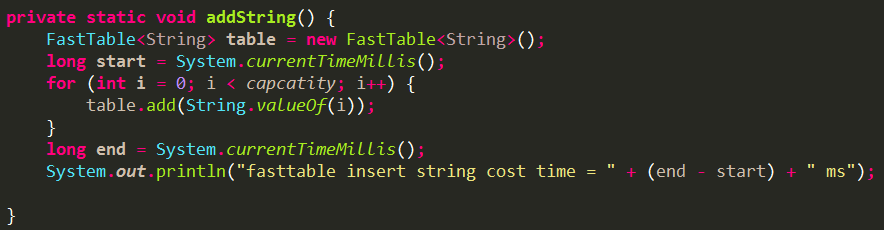
Jvm 分析命令

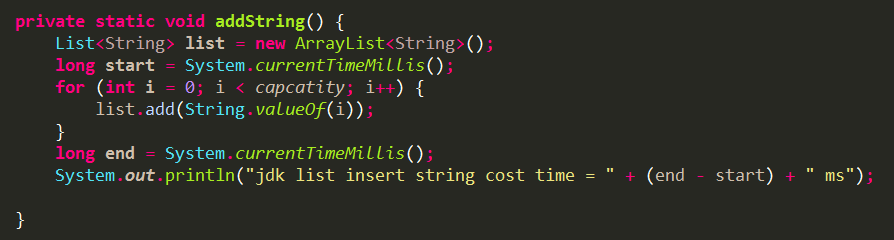
jps

Jstat –gc pid

Jstat –gcutil -pid

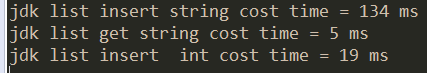
程序代码

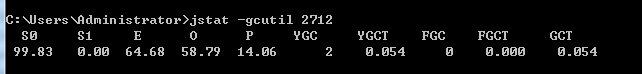




数据量100000，测试结果

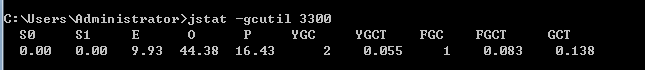
ArrayList：





fastTable:





ArrayList添加删除时间复杂度是o(size)

FastTable 添加删除时间复杂度O(log(size))

实验结果为什么反而慢些？

对比性能、内存占用、内存回收

数据量增加到千万级,10000000

测试结果

ArrayList 直接堆内存溢出



FastTable



官方性能对比



类介绍：

解决的数据结构数组、链表、队列，线程安全，支持并发修改、删除。提供函数式操作。

如何分析源码过程？

功能点：

**fractal : 它承认世界的局部可能在一定条件下，在某一方面（形态，结构，信息，功能，时间，能量等）表现出与整体的相似性，它承认空间维数的变化既可以是离散的也可以是连续的，因而拓展了视野。**

**1省存储空间，图形可以用几个简单的参数存储  
2省运算时间，渲染一个图形时，只需要渲染局部然后复制到整体**

一个类同时满足这么多数据结构有没有增加复杂性？类设计如何满足优雅？

为什么内存占用更少？空间与性能如何同时满足？

#### 1.2.1.2 FastSortedTable

#### 1.2.1.3 FastMap

#### 1.2.1.4 FastSet

#### 1.2.1.5 FastSortedMap

#### 1.2.1.6 FastSortedSet

## 1.5 高性能集合分析对比

## 1.6 JMH基准测试

<http://openjdk.java.net/projects/code-tools/jmh/>

<http://hg.openjdk.java.net/code-tools/jmh/file/tip/jmh-samples/src/main/java/org/openjdk/jmh/samples/>

# 分享大纲

1. JDK集合框架介绍
2. 集合框架与数据结构的关系
3. 如何选择集合库

<https://plumbr.eu/blog/memory-leaks/selecting-your-collections-library>

jdk 1.6

高性能集合库

<http://www.findbestopensource.com/tagged/java-collection?start=0>

1. Guava基础工具类

4.1 Guava功能全面介绍

4.2 重点 排序、检查、Hash、Strings操作、**ListenableFuture、EventBus**

1. Guava集合
2. Guava限流特技，及常用限流的方法
3. 大数据量下集合的挑战对比？与实时框架的对比？

Jdk6、guava、javolution、magicwerk-collections、JDK8、Trove

高性能集合其他库对比

<http://java-performance.com/>

<http://www.magicwerk.org/page-collections-overview.html>

<https://dzone.com/articles/biglist-scalable-high>

<https://jaxenter.de/high-performance-lists-fuer-java-28342>

<https://dzone.com/articles/high-performance-libraries>

<http://www.mobile-open.com/2016/951597.html>

自动封箱解箱性能测试

<http://www.javaspecialists.co.za/archive/Issue090.html>

magicwerk-collections

<http://2015.javazone.no/details.html?talk=3b2c46f15ec292bfbed82adb2984ef17e2c47748768167f9060b263a74e37659>

<http://2015.javazone.no/details.html?talk=3b2c46f15ec292bfbed82adb2984ef17e2c47748768167f9060b263a74e37659>

<https://dzone.com/articles/brownies-collections-097>

<https://dzone.com/articles/biglist-scalable-high>