# Collections 和 Arrays

java.util.Collection是集合的顶层接口，定义了对集合对象操作的通用方法，这些方法包括：

size()

contains(Object)

iterator()

toArry()

toArray(T[])

add(E)

remove(Object)

containsAll(Collection)

addAll(Collection)

removeAll(Collection)

clear()

retainAll(Collection)（保留）

**java 8 新增的：**

spliterator()（并行遍历元素）

stream()

removeIf(Predicate)

parallelStream()（并行流）

forEach(Consumer)（默认方法）

## toArray()和toArray(T[])方法

如下代码会报错：ClassCastException

ArrayList<Integer> list = new ArrayList<Integer>();  
for (int i = 0; i < 10; i++) {  
 list.add(i);  
}  
Integer[] array = (Integer[]) list.toArray();

Java中的类型转换是针对单个对象的，要将一个数组（**数组为引用类型**）对象强转为另一种数组类型，就必须确保这个数组在初始化时就是目标类型。

查看数组类型：

(new List[]{new ArrayList(), new LinkedList(), new ArrayList()}).getClass().getName()

类型名为：java.util.List

用上面的方法可以知道toArray方法返回的数组类型为Object，因而无法将装有Object类型对象的数组转换为指定类型的数组。

而toArray(T[])方法会借助泛型创建装有T类型的数组，返回的数组类型也为T类型，因此**应该使用toArray(T[])方法**。

结论：**toArray(T[])方法会返回指定类型的数组对象，toArray(T[])无需强转；而toArray()方法返回Object类型的数组对象，不能直接强转，只能对返回数组的具体元素进行强转。**

## Collections

Collections是一个工具类，包含很多实用的有关集合操作的方法（及其重载方法）。

如：

交换元素：swap(List,int,int)

填充集合：fill(List,T)

最大/最小值：max/min(Collection,Comparator)

打乱：shuffle(List)

排序：sort(List,Comparator)

二分查找：binarySearch(List,T)

创建一些特殊的集合（单元素集合、只读集合、同步集合etc）

…

## Array和Arrays

### Java.lang.reflect.Array

定义为final，最终类。该类的所有方法都为本地方法（native）。是通用数组的统一类实现形式。

### Java.util.Arrays

工具类，包含大量的方法来快捷的操作数组。

如：

拥有多个重载方法的排序实现：sort(int[]/double[]…)，sort(T[],comparator)

获得并行处理对象：spliterator(int[]/double[]…)

二分查找：binarySearch(byte[]/int,byte/int)

判断深度相等（多维数组）：deepEquals(Object[],Object[])

填充：fill(byte[],byte)

…

# assert

断言，一种常用的调试方式，包含一个boolean表达式，执行时假定表达式是true，若计算为false，系统会报告一个AssertionError并退出，用于调试。

1. assert expression1;

2. assert expression1: expression2;

assert 2 > 0 : "this is not true";  
assert false; // AssertionError

一个正确的程序必须保证expression为true，为false就表明程序处于一种不正确的情况下，系统会给出警告并退出。

一般情况下，assertion功能是关闭的，

* 要在编译时启用需加上 –source 1.4 参数（表示使用JDK1.4编译源码，assertion是JDK1.4添加的特性）。
* 要在运行时启用断言，可使用 -enableassertions 或者 -ea 标记。
* 要在运行时选择禁用断言，可使用 -da 或者 -disableassertions 标记。

# java GC（Garbage Collection）

1. [垃圾回收介绍](http://www.importnew.com/13504.html)
2. [垃圾回收是如何工作的？](http://www.importnew.com/13493.html)
3. [垃圾回收的类别](http://www.importnew.com/13827.htm)
4. [垃圾回收监视和分析](http://www.importnew.com/13838.html)

Java 垃圾回收是一项自动化的过程，用来管理程序所使用的运行时内存。通过这一自动化过程，JVM 解除了程序员在程序中分配和释放内存资源的开销。

每种JVM实现会采用不同的方法实现垃圾回收机制，基于HotSpot JVM分析GC。

与垃圾回收相关的主要组件为**堆内存**和**垃圾回收器**。在运行时，java实例被存放在堆内存区域，当一个对象不再被引用时，满足条件就会从堆内存移除。堆内存分为三个主要区域：新生代、老年代、永久代（javaSE 8 中被移除）。

GC会自动启动，程序员不需要在代码中显示启动垃圾回收过程。System.gc()和Runtime.getRuntime().gc()用来**请求**JVM启动垃圾回收。（启动由JVM负责，JVM可能拒绝这个请求）

**垃圾回收中实例的终结**

在释放一个实例和回收内存空间之前，Java 垃圾回收器会调用实例各自的 finalize() 方法，从而该实例有机会释放所持有的资源。虽然可以保证 finalize() 会在回收内存空间之前被调用，但是没有指定的顺序和时间。多个实例间的顺序是无法被预知，甚至可能会并行发生。**程序不应该预先调整实例之间的顺序并使用 finalize() 方法回收资源**。

* 任何在 finalize过程中未被捕获的异常会自动被忽略，然后该实例的 finalize 过程被取消。
* JVM 规范中并没有讨论关于弱引用的垃圾回收机制，也没有很明确的要求。具体的实现都由实现方决定。
* 垃圾回收是由一个守护线程完成的。

**对象什么时候符合垃圾回收的条件？**

所有实例都没有活动线程访问。

没有被其他任何实例访问的循环引用实例

[Java 中有不同的引用类型](http://javapapers.com/core-java/java-weak-reference/)。判断实例是否符合垃圾收集的条件都依赖于它的**引用类型**。

| **引用类型** | **垃圾收集** |
| --- | --- |
| 强引用（Strong Reference） | 不符合垃圾收集 |
| 软引用（Soft Reference） | 垃圾收集可能会执行，但会作为最后的选择 |
| 弱引用（Weak Reference） | 符合垃圾收集 |
| 虚引用（Phantom Reference） | 符合垃圾收集 |

Java编译器会将实例后从未被使用的对象赋值为null（java编译器的一种优化策略），标记实例可回收。

# String str = new String(“str”)

<http://www.cnblogs.com/OnlyCT/p/5433410.html>

执行语句String str = new String(“str”);时，不管字符串池中是否存在字符串”str”，会直接新建一个字符串（不是在字符串池中），然后赋给str。

**总之，创建字符串有两种方式：两种内存区域(pool,heap)**

1. ""创建的字符串在**字符串池**中；
2. new 创建字符串时，首先查看池中是否有相同的字符串，如果有则**拷贝**一份放到堆中，然后返回堆中的地址；如果池中没有则在堆中创建一分，然后返回堆中的地址；（**用构造器创建的对象是不会被放入常量池的**）
3. 在对字符串赋值时，如果右操作数含有一个或一个以上的字符串引用时，则在堆中再建立一个字符串对象，返回引用如：String s= str1+"blog";（StringBuilder的toString方法会new新的String对象）

如：

String a = “a”;

String b = “b”;

String ab = a + b;

ab == “ab” **// false，ab的结果是运行时确定的，且通过StringBuilder拼接的。**

# equals 和hashcode