链滴

Java 面试常见问题整理

作者：douniwan

原文链接：https://ld246.com/article/1576502576148

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="null" data-src="https://b3lo
file.com/bing/20180322.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
记录一些常见的面试题，为以后面试做准备
<h3 id="目录">目录</h3>
2019 - 12 -11
1.jdk，jre 的区别是什么？jvm 又是什么？
2.<code>static</code> 关键字是什么意思？Java 中是否可以覆盖（override
一个 <code>private</code> 或者是 <code>static</code> 的方法？
3.是否可以在 <code>static</code> 环境中访问非 <code>static</code>
量？
4.Java 中支持的数据类型有哪些？什么是自动拆装箱？
2019 - 12 -12
5.什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？ <
p>
6.Java 中的方法覆盖（Overriding）和方法重载（Overload）是什么意思？ </a

7.java 中，什么是构造方法？什么是构造方法重载？什么是复制构造方法？ </a

8.Java 支持多继承吗？
9.接口和抽象类的区别是什么？
10.什么是值传递和引用传递
11.进程和线程的区别是什么？
12.在 Java 中创建线程有几种不同的方式？你喜欢哪一种？为什么？
2019 - 12 -13
13.概括的解释下线程的几种可用状态
14.同步方法和同步代码块的区别是什么呢？
15.在监视器（Monitor）的内部，是如何做线程同步的，程序应该做到哪种级
的同步？
2019 -12 -17
16.什么是死锁（deadlock）？
17.如何确保 N 个线程可以访问 N 个资源同时有不导致死锁？
18.Java 集合类框架的基本接口有哪些？
19.为什么集合类没有实现 Cloneable 和 Serializable？
20.什么是迭代器（Iterator）？
21.Iterator 和 ListIterator 的区别是什么？
2019 -12 -18
22.快速失败（fail-fast）和安全失败（fail-safe）的区别是什么
23.Java 中的 HashMap 的工作原理是什么？
24.hashcode()和 equals()方法的重要性体现在什么地方？
25.HashMap 和 HashTable 有什么区别？
26.数组（Array）和列表（ArrayList）有什么区别？什么时候应该使用 Array
不是 ArrayList？
27.ArrayList 和 LinkedList 有什么区别？
 28.Comparable 和 Comparator 接口是干什么的？列出他们的区别。 <b
>

<hr>
<h4 id="1-jdk-jre的区别是什么-jvm又是什么-">1.jdk，jre 的区别是什么？jvm 又是什么
</h4>
<blockquote>
JDK: <code>Java Development ToolKit</code>(Java 开发工具包)。JDK 是整个 JAVA 的
心，包括了 Java 运行环境（Java Runtime Envirnment），一堆 Java 工具（javac/java/jdb
）和 Java 基础的类库（即 Java API 包括 rt.jar）。可以让开发者开发，编译，执行 Java 应

程序
JRE:<code>Java Runtime Enviromental</code>(java 运行时环境)。也就是我们说的 JAVA 台，所有的 Java 程序都要在 JRE 下才能运行。包括 JVM 和 JAVA 核心类库和支持文件。 JDK 相比，它不包含开发工具——编译器、调试器和其它工具。
JVM：<code>Java Virtual Mechinal</code>(JAVA 虚拟机)。JVM 是 JRE 的一部分，它是一虚构出来的计算机，是通过在实际的计算机上仿真模拟各种计算机功能来实现的。
</blockquote>
<h4 id="2-关键字是什么意思-Java中是否可以覆盖-override-一个或者是的方法-">2.<cod>static</code> 关键字是什么意思？Java 中是否可以覆盖（override）一个 <code>private</cod> 或者是 <code>static</code> 的方法？</h4>
<blockquote>
在 Java 中变量的类型有类变量，成员变量，局部变量，用 <code>static</code> 修饰的变量是类变量，他不属于类的具体实例，而是属于类的。
Java 中的 <code>private</code> 方法不能被覆盖，因为 <code>private</code> 修饰的量或者方法只在当前类中可以访问，如果是其他的类继承当前类是不能访问到 <code>private</cod> 变量或者方法的，因此当然不能覆盖。
Java 中的 <code>static</code> 方法也不能被覆盖，因为方法覆盖是基于运行时动态绑定的而 <code>static</code> 方法是编译时静态绑定的。<code>static</code> 方法跟类的任何实例不相关，所以概念上不适用。
</blockquote>
<h4 id="3-是否可以在环境中访问非变量-">3.是否可以在 <code>static</code> 环境中问非 <code>static</code> 变量？</h4>
<blockquote>
<code>static</code> 变量在 Java 中是属于类的，他在所有的实例中的值都是一样的，当类被ava 虚拟机载入的时候，会对 <code>static</code> 变量进行初始化。如果你的代码尝试不用实例访问非 <code>static</code> 的变量，编译器会报错，因为这些变量还没有被创建出来，还没有跟何实例绑定。所以不可以在 <code>static</code> 环境中访问非 <code>static</code> 变量
</blockquote>
<h4 id="4-Java中支持的数据类型有哪些-什么是自动拆装箱-">4.Java 中支持的数据类型有些？什么是自动拆装箱？</h4>
<blockquote>
Java 中一共有两大类数据类型，一种是基本数据类型，一种是引用数据类型。其中基本数据类有 8 种，它们分别是，<code>byte</code>,<code>short</code>,<code>int</code>,<code>ong</code>,<code>float</code>,<code>double</code>,<code>boolean</code>,<code>har</code>，除了 8 种基本数据类型外，还有 3 种引用类型，它们分别是，类，接口，数组
自动装箱是 Java 编译器在基本数据类型和对应的包装类型之间做得一个转化。比如: 把 <code>it</code> 转化为 <code>Integer</code>，<code>double</code> 转化为 <code>Double</cde>，等等。反之就是自动拆箱。
</blockquote>
<h4 id="5-什么是Java虚拟机-为什么Java被称作是-平台无关的编程语言--">5.什么是 Java 虚拟机？为什么 Java 被称作是“平台无关的编程语言”？</h4>
<blockquote>
Java 虚拟机是一个可以执行 Java 字节码的虚拟机进程。Java 源文件被编译成能被 Java 虚拟机行的字节码文件。Java 被设计成允许应用程序可以运行在任意的平台，而不需要程序员为每一个平台独重写或者是重新编译。而这写能够实现，基本全靠 Java 虚拟机，Java 虚拟机屏蔽了底层操作系统硬件的差异，以此让 Java 编译后的字节码在 Java 虚拟机上运行的结果一致。值得注意的是，Java 然是平台无关性的，但是 Java 虚拟机是平台有关的，正因为 Java 虚拟机平台有关性，才造就了 Java 语言的平台无关性。
</blockquote>
<h4 id="6-Java中的方法覆盖-Overriding-和方法重载-Overload-是什么意思-">6.Java 中方法覆盖（Overriding）和方法重载（Overload）是什么意思？</h4>
<blockquote>
Java 中的方法重载发生在同一个类里面，两个或者是多个方法的方法名相同但是参数不同的情。与此相对，方法覆盖是说子类重新定义了父类的方法。方法覆盖必须有相同的方法名，参数列表和

回类型。覆盖者可能不会限制他所覆盖的方法的访问。
</blockquote>
<h4 id="7-java中-什么是构造方法-什么是构造方法重载-什么是复制构造方法-">7.java 中什么是构造方法？什么是构造方法重载？什么是复制构造方法？</h4>
<blockquote>
当新对象对创建的时候，构造方法会被调用。每一个类都有构造方法。在程序员没有给类提供构方法的情况下，Java 编译器会为这个类创建一个默认的构造方法。Java 中构造方法重载和方法重载类似。可以为一个类创建多个构造方法。每一个构造方法必须有它自己唯一的参数列表。Java 不支持 C++ 中那样的复制构造方法，因为如果你不自己写构造方法的情况下，Java 不会创建默认的复制构方法。
</blockquote>
<h4 id="8-Java支持多继承吗-">8.Java 支持多继承吗？</h4>
<blockquote>
Java 中类不支持多继承，只支持单继承，即一个类只有一个父类。这是基于安全性的考虑，如子类继承的多个父类里面有相同的方法或者属性，子类将不知道具体要继承哪个。而接口是可以多实，接口的作用是用来扩展对象的功能，一个子接口实现多个父接口，因为接口只定义方法，而没有具的逻辑实现，多实现也要重新实现方法。
</blockquote>
<h4 id="9-接口和抽象类的区别是什么-">9.接口和抽象类的区别是什么？</h4>
<blockquote>
接口中的所有方法隐含的都是抽象的。而抽象类这可以同时包含抽象和非抽象的方法。类可以实多个接口，但是只能继承一个抽象类。类可以不实现抽象类和接口声明的所有方法，当然，在这种情下，类也必须的声明是抽象的。在 jdk8 之前，接口之中可以定义变量和方法，量必须是 <code>public</code>、<code>static</code>、<code>final</code> 的，方法必须 <code>public</code>、<code>abstract</code> 的。JDK8 及以后，允我们在接口中定义 <code>static</code> 方法和 <code>default</code> 方法，当定义一个实现实现该接口后，由于 <code>default</code> 方法不是 <code>abstract</code> 的，所以可以实现类中不重写。抽象类可以有 <code>private</code>，<code>protected</code> 和 <code public</code> 等修饰符
</blockquote>
<h4 id="10-什么是值传递和引用传递">10.什么是值传递和引用传递</h4>
<blockquote>
值传递（pass by value）：是指在调用函数时将实际参数复制一份传递到函数中，这样在函数如果参数进行修改，将不会影响到实际参数
引用传递（pass by reference）：是指在调用函数的时候将实际参数的地址直接传递到函数中那么在函数中对参数的修改，将影响到实际参数。
区别：值传递会创建副本，引用传递不会在 Java 中不管是基本数据类型还是引用类型，都是值递！关于此点请看 <a href="https://ld246.com/forward?goto=http%3A%2F%2Fblog.zhqy.xyz%Farticles%2F2019%2F10%2F28%2F1572216852597.html" target="_blank" rel="nofollow ugc"为什么 java 中只有值传递
</blockquote>
<h4 id="11-进程和线程的区别是什么-">11.进程和线程的区别是什么？</h4>
进程是执行着的程序，而线程是进程内部的一个执行序列。一个进程可以包含多个线程。线程又做轻量级进程。线程与进程的区别归纳：
<blockquote>

地址空间和其他资源：进程之间相互独立，同一进程的各线程间共享。某进程内的线程在其它进不可见。

通信：进程间通信 IPC，线程之间可以直接读取数据段（如全局变量）来进行通信——需要进程步和互斥手段的辅助，以保证数据的一致性。

调度和切换：线程上下文切换比进程上下文切换要快得多。

在多线程 OS 中，进程不是一个可执行的实体。

</blockquote>
<h4 id="12-在Java中创建线程有几种不同的方式-你喜欢哪一种-为什么-">12.在 Java 中创线程有几种不同的方式？你喜欢哪一种？为什么？</h4>
<blockquote>
有四种方式可以用来创建线程：

继承 <code>Thread</code> 类,实现方法 <code>run()</code> 不可以抛异常 无返回值</p

实现 <code>Runnable</code> 接口,实现方法 <code>run()</code> 不可以抛异常 无返回

实现 <code>Callable<T></code> 接口,要覆盖的方法是 <code>public <T> call()/code> 可以抛异常,可以有返回值

使用 <code>Executor</code> 接口的 <code>ThreadPoolExecutor</code> 来创建线程池一般通过实现 <code>Runnable</code> 接口，因为这种方式避免了单继承的局限，一个类可以继多个接口。

</blockquote>
<h4 id="13-概括的解释下线程的几种可用状态">13.概括的解释下线程的几种可用状态</h4>
<blockquote>

新建（new）：初始状态，线程被创建，没有调用 <code>start()</code>

就绪（Runnable）：线程调用了 <code>start()</code>，此时位于可运行线程池中，等待线被调度选中获取 CPU 的时间片。

运行（Running）：就绪状态的线程获得了 CPU 时间片，执行程序代码

阻塞（Blocked）：线程进入等待状态，线程因为某种原因，让出了 CPU 的使用权，停止运行直到线程进入就绪状态，才有可能再次获得 CPU 时间片进入运行状态。

死亡（dead）：线程 <code>run()</code>、<code>main()</code> 方法执行完毕，或者为异常退出了 <code>run()</code> 方法，则线程结束生命周期。
其中阻塞的情况分成三种

等待阻塞：运行的线程执行了 wait()，JVM 会把当前线程放入等待队列

同步阻塞：运行的线程在获取对象的同步锁时，如果该同步锁被其他线程占用了，JVM 会把当线程放入锁池中

其他阻塞：运行的线程执行 sleep(),join()或者发出 IO 请求时，JVM 会把当前线程设置为阻塞状，当 sleep()执行完，join()线程终止，IO 处理完毕线程再次恢复

</blockquote>
<h4 id="14-同步方法和同步代码块的区别是什么呢-">14.同步方法和同步代码块的区别是么呢？ </h4>
<blockquote>
同步方法默认用 <code>this</code> 或者当前类 class 对象作为锁。同步代码块可以选择以什来加锁，比同步方法要更细腻度，我们可以选择只同步会发生同步问题的代码块而不是整个方法。同方法使用关键字 <code>synchronized</code> 修饰方法，而同步代码块主要是修饰需要进行同步代码，用 <code>synchronized(object){代码内容}</code> 进行修饰。
</blockquote>
<h4 id="15-在监视器-Monitor-的内部-是如何做线程同步的-程序应该做到哪种级别的同步-"><spa>15.在监视器（Monitor）的内部，是如何做线程同步的，程序应该做到哪种级别的同步？ /h4>
<blockquote>
监视器和锁在 Java 虚拟机中是一块使用的。监视器监视一块同步代码块，确保一次只有一个线执行同步代码块。每一个监视器都和一个对象引用相关联。线程在获取锁之前不允许执行同步代码！
</blockquote>
<h4 id="16-什么是死锁-deadlock--">16.什么是死锁（deadlock）？ </h4>
<blockquote>
所谓死锁就是指多个进程因为竞争资源而造成一种僵局（互相等待）。若无外力作用，这些进程将无法向前推进。死锁产生的四个条件。

互斥条件：进程要求对所分配的资源进行排他性控制，即在一段时间内，某个资源仅为一个进程占有。此时如果有其他进程请求该资源，则请求进程只能等待。
不剥夺条件：进程所获得资源在没有使用完毕之前，不能被其他进程强行夺走，即只能由获得该源的进程自己释放（即主动释放）
请求和保持条件：进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源又被其他程所占有，此时请求进程被阻塞。但对自己已获得资源保持不放。
循环等待条件：存在一种进程资源的循环等待链，链中每一个进程以获得的资源同时被链中下一进程所请求。

</blockquote>
<h4 id="17-如何确保N个线程可以访问N个资源同时有不导致死锁-">17.如何确保 N 个线可以访问 N 个资源同时有不导致死锁？ </h4>
<blockquote>
预防死锁，预先破坏产生死锁的四个条件，线程互斥基本不可避免，所以主要有破坏请求和保持件，破坏不剥夺条件和破坏循环等待条件。其中实现简单的就是破坏循环等待条件。这个可以通过指加锁顺序，即让所有的线程按照相同的顺序获取资源的锁，不先获得顺序靠前的锁，就无法获得后续锁。则先获得锁的线程不会请求后获得的锁的线程需要的资源，因为后获得锁的线程还没有获得先获锁的线程未释放的锁，更无法占用先获得锁的线程还没获得的顺序靠后的锁。
</blockquote>

<h4 id="18-Java集合类框架的基本接口有哪些-">18.Java 集合类框架的基本接口有哪些？<
span></h4>
<blockquote>
集合类接口指定了一组叫做元素的对象。集合类接口的每一种具体的实现类都可以选择以他自己方式对元素进行保存和排序。有的集合类允许重复的键，有些不允许。
Java 集合类里面最基本的接口有

Collection：代表一组对象，每一个对象都是他的子元素。
Set：不包含重复元素的 Collection。
List：有顺序的 Collection，并且可以包含重复元素。
Map：可以把键（key）映射到值（value）的对象，键不能重复。

</blockquote>
<h4 id="19-为什么集合类没有实现Cloneable和Serializable-">19.为什么集合类没有实现 C
oneable 和 Serializable？</h4>
<blockquote>
克隆（cloning）或者序列化（serialization）的语义和含义是跟具体的实现相关的。因此，应该集合类的具体实现类决定如何被克隆或者是序列化！
</blockquote>
<h4 id="20-什么是迭代器-Iterator--">20.什么是迭代器（Iterator）？</h4>
<blockquote>
Iterator 接口提供了很多对集合元素进行迭代的方法。每一个集合类都包含了可以返回迭代器实的迭代方法。迭代器可以在迭代过程中删除底层集合的元素，但是不可以直接调用集合的 remove(Ob
ect Obj)删除，可以通过迭代器的 remove（）方法删除。
</blockquote>
<h4 id="21-Iterator和ListIterator的区别是什么-">21.Iterator 和 ListIterator 的区别是什
？</h4>
<blockquote>
Iterator 可用来遍历 Set 和 Listen 集合，但是 ListIterator 只能用来遍历 List。
Iterator 对集合只能是前向遍历，ListIterator 既可以前向也可以后向。
ListIterator 实现了 Iterator 接口，并包含其他的功能，比如：增加元素，替换元素，获取前一个和一个元素的索引，等等。
</blockquote>
<h4 id="22-快速失败-fail-fast-和安全失败-fail-safe-的区别是什么">22.快速失败（fail-fas
）和安全失败（fail-safe）的区别是什么</h4>
<blockquote>

快速失败：
在使用迭代器遍历一个集合对象时，如果遍历过程中对集合的结构进行了修改（增加，删除），则会出 <code>ConcurrentModificationException</code>
原理：迭代器在遍历时直接访问集合中的内容，并且在遍历过程中使用一个 <code>modCoun</co
e> t 变量。集合在被遍历期间如果结构发生变化，就会改变 <code>modCount</code> 的值每当代器使用 <code>next()</code>/<code>hashNext()</code> 遍历下一个元素之前，都会检测 m
dCount 变量是否为 expectedmodCount 值，是的话就返回遍历；否则就抛出异常，终止遍历。<b
>
注意：这里的异常的抛出条件是检测到 `modCount != expectedmmodCount 值，则异常不会抛出因此，不能依赖于这个异常是否抛出而进行并发操作的编程，这个异常只建议用户检测并发修改的 bu
。
场景：Java.util 包下的集合类都是快速失败的，不能在多线程下发生并发修改（迭代过程中被修改）<
li>
安全失败：
采用安全失败机子的集合容器，再遍历时不是直接在集合内容上访问的，而是先复制原有的集合内容在拷贝的集合上进行遍历。
原理：由于迭代时是对原集合的拷贝进行遍历，所以在遍历过程中对元集合所作的修改并不能被迭代

检测到，所以不会触发 <code>ConcurrentModificationException</code>
缺点：基于拷贝内容的优点是避免了 <code>ConcurrentModificationException</code>，但同的，迭代器并不能访问到修改后的内容，即：迭代器遍历的是开始遍历那一刻拿到的集合拷贝，在便期间原集合发生的修改迭代器是不知道的。
场景：java.util.concurrent 包下都是安全失败，可以在多线程下并发使用，并发修改。

</blockquote>
<h4 id="23-Java中的HashMap的工作原理是什么-">23.Java 中的 HashMap 的工作原理什么？</h4>
<blockquote>
Java 中的 HashMap 是以键值对的（key-value）的形式存储元素的。Has Map 都需要一个 Hash 函数，它使用 <code>HashCode()</code> 和 <code>equals()</code> 法来向集合/从集合添加和检验元素。当调用 <code>put()</code> 方法的时候，HashMap 会计算 ey 的 HashMap 值，然后把键值对存储在集合中合适的索引上。如果 key 已经存在了，value 会被新成新值。HashMap 的一些重要的特性是它的容量(capacity)，负载因子（load factor）和扩容极（threshold resizing）
</blockquote>
<hr>
<blockquote>
hashmap 是一个 key-value 键值对的数据结构，从结构上来讲在 jdk1.8 之前是用数组加链表方式实现，jdk1.8 加了红黑树，hashmap 数组的默认初始长度是 16，hashmap 数组只允许一个 key 为 null，允许多个 value 为 null
hashmap 的内部实现，hashmap 是使用数组 + 链表 + 红黑树的形式实现的，其中数组是一个一个 ode[]数组，我们叫他 hash 桶数组，它上面存放的是 key-value 键值对的节点。HashMap 是用 hash 表来存储的，在 hashmap 里为解决 hash 冲突，使用链地址法，简单来说就是数组加链表的形式来决，当数据被 hash 后，得到数组下标，把数据放在对应下表的链表中。
然后再说一下 hashmap 的方法实现
put 方法，put 方法的第一步，就是计算出要 put 元素在 hash 桶数组中的索引位置，得到索引位置要三步，去 put 元素 key 的 hashcode 值，高位运算，取模运算，高位运算就是用第一步得到的值，用 h 的高 16 位和低 16 位进行异或操作，第三步为了使 hash 桶数组元素分布更均匀，采用取模算，取模运算就是用第二步得到的值和 hash 桶数组长度-1 的值取与。这样得到的结果和传统取模运结果一致，而且效率比取模运算高
jdk1.8 中 put 方法的具体步骤，先判断 hashmap 是否为空，为空的话扩容，不为空计算出 key 的 h sh 值 i，然后看 table[i]是否为空，为空就直接插入，不为空判断当前位置的 key 和 table[i]是否相，相同就覆盖，不相同就查看 table[i]是否是红黑树节点，如果是的话就用红黑树直接插入键值对，果不是开始遍历链表插入，如果遇到重复值就覆盖，否则直接插入，如果链表长度大于 8，转为红黑结构，执行完成后看 size 是否大于阈值 threshold，大于就扩容，否则直接结束
get 方法就是计算出要获取元素的 hash 值，去对应位置取即可。
扩容机制，hashmap 的扩容中主要进行两部，第一步把数组长度变为原来的两倍，第二部把旧数组元素重新计算 hash 插入到新数组中，在 jdk1.8 时，不用重新计算 hash，只用看看原来的 hash 值增的一位是零还是 1，如果是 1 这个元素在新数组中的位置，是原数组的位置加原数组长度，如果是就插入到原数组中。扩容过程第二部一个非常重要的方法是 transfer 方法，采用头插法，把旧数组的素插入到新数组中。
</blockquote>
<blockquote>
hashmap 大小为什么是 2 的幂次方
在计算插入元素在 hash 桶数组的索引时第三步，为了使元素分布的更加均匀，用取模操作，但是传取模操作效率低，然后优化成 h&(length-1)，设置成 2 幂次方，是因为 2 的幂次方-1 后的值一位上都是 1，然后与第二步计算出的 h 值与的时候，最终的结果只和 key 的 hashcode 值本身有，这样不会造成空间浪费并且分布均匀
如果 length 不为 2 的幂，比如 15。那么 length-1 的 2 进制就会变成 1110。在 h 为随机数的情况，和 1110 做&操作。尾数永远为 0。那么 0001、1001、1101 等尾数为 1 的位置就永远不可被 entry 占用。这样会造成浪费，不随机等问题。
</blockquote>

<hr>
<h4 id="24-hashcode--和equals--方法的重要性体现在什么地方-">24.hashcode()和 equals()方法的重要性体现在什么地方？</h4>
<blockquote>
Java 中 HashMap 使用 <code>HashCode()</code> 和 <code>equals</code> 方法来确定键值对的索引，当根据键获取值的时候也会用到这两个方法。如果没有正确的实现这两个方法，两个不同的键可能会有相同的 hash 值。而且，这两个方法也用来发现重复元素。所以这两个方法的实现对于 HashMap 的精确性和正确性是至关重要的。
</blockquote>
<h4 id="25-HashMap和HashTable有什么区别-">25.HashMap 和 HashTable 有什么区别？</h4>
HashMap 和 HashTable 都实现了 Map 接口，因此很多特性非常相似。但是，他们有以下不同点：
HashMap 允许键和值是 null，而 HashTable 不允许键或者是值为 null。
HashTable 是同步的，而 HashMap 不是。因此，HashMap 更适合单线程环境，而 HashTable 适合于多线程环境。
HashMap 提供了可供应迭代的键的集合，因此，HashMap 是快速失败的。另一方面，HashTable 提供了对键的列举（Enumeration）。
<h4 id="26-数组-Array-和列表-ArrayList-有什么区别-什么时候应该使用Array而不是ArrayList-">26.数组（Array）和列表（ArrayList）有什么区别？什么时候应该使用 Array 而不是 ArrayList？</h4>
Array 可以包含基本数据类型和对象类型，而 ArrayList 只能包含对象类型。
Array 是固定大小的，ArrayList 的大小是动态变化的。
ArrayList 提供了更多的方法和特性，比如：<code>addAll()</code>,<code>remove()</code>,<code>iterator()</code> 等等
对于基本数据类型，集合类使用自动装箱来减少编码工作量。但是，当处理固定大小的基本数据类型的时候，这种方式比较慢
<h4 id="27-ArrayList和LinkedList有什么区别-">27.ArrayList 和 LinkedList 有什么区别？</h4>
ArrayList 的实现用的是数组，LinkedList 是基于链表实现。ArrayList 适合查找，LinkedList 适合增删！
<blockquote>
ArrayList 和 LinkedList 都实现了 List 接口，他们有以下的不同点。
ArrayList 是基于索引的数据接口，他的底层是数组。他可以以 O(1)时间复杂度对元素进行随机访问。与此对应，LinkedList 是以元素进行随机访问。与此的语音，LinkedList 是以元素列表的形式存储他的数据，每一个元素都和他的前一个和后一链接在一起，在这种情况下，查找某个元素的时间复杂度是 O(n)。相对于 ArrayList，LinkedList 的插入，添加，删除操作更快，因为当元素被添加到集合任意位置的时候，不需要像数组那样重新计算大小或者是更新索引。
LinkedList 比 ArrayList 更占内存，因为 LinkedList 为每一个节点存储了两个引用，一个指向了前一个元素，一个指向了后一个元素。
</blockquote>
<h4 id="28-Comparable和Comparator接口是干什么的-列出他们的区别-">28.Comparable 和 Comparator 接口是干什么的？列出他们的区别。</h4>
<blockquote>
Java 提供了只包含一个 compareTo 方法的 Comparable 接口。这个方法可以给两个对象排序。具体来说，它返回负数，0，正数来表明已经存在的对象小于，等于，大于输入对象。
Java 提供了 compare()和 equals()两个方法的接口 Comparator。compare()方法用来给两个输入参数排序，返回负数，0，正数来表明第一个参数是小于，等于，大于第二个参数的。equals()方法需要一个对象作为参数，它用来决定输入参数是否和 comparator 相等。只有当输入参数也是一个 comparator 并且输入参数和当前 comparator 的排序结果是相同的时候，这个方法才返回 true。
</blockquote>
<hr>
<blockquote>
Comparable & Comparator 都是用来实现集合中元素的比较、排序的，只是 Comparable

是在集合内部定义的方法实现的排序，Comparator 是在集合外部实现的排序，所以，如想实现排序就需要在集合外定义 Comparator 接口的方法或在集合内实现 Comparable 接口的方法。 Comparator 位于包 java.util 下，而 Comparable 位于包 java.lang 下 Comparable 是一个对象本身就已经支自比较所需要实现的接口（如 String、Integer 自己就可以完成比较大小操作，已经实现了 Comparable 接口） 自定义的类要在加入 list 容器中后能够排序，可以实现 Comparable 接口，在用 Collection 类的 sort 方法排序时，如果不指定 Comparator，那么就以自然顺序排序， 这里的自然顺序就是实 Comparable 接口设定的排序方式。 而 Comparator 是一个专用的比较器，当这个对象不支持自比或者自比较函数不能满足你的要求时，你可以写一个比较器来完成两个对象之间大小的比较。 可以说个是自己完成比较，一个是外部程序实现比较的差别而已。 用 Comparator 是策略模式（strategy design pattern），就是不改变对象自身，而用一个策略对象（strategy object）来改变它的行为。 比：你想对整数采用绝对值大小来排序，Integer 是不符合要求的，你不需要去修改 Integer 类（实际你也不能这么做）去改变它的排序行为，只要使用一个实现了 Comparator 接口的对象来实现控制它排序就行了。</p>
</blockquote>
<h4 id="toc_h4_29"></h4>
<h4 id="-"></h4>