第二部分　CoreJava 基础

第 04 章　解读 API

4.1　Object 常用的方法

解析：面试刚开始都很简单，但也容易给面试官留下印象，就像这个题，如果你有三年以上经验，回答的少于六个，那么估计你下面将很危险了。Object 中的这些方法都很精典，务必记住。

参考答案：object 常用的方法有 clone(),equals(),hashCode(),notify(),notifyAll(), toString(),wait(),finalize()。

4.2　String 与 StringBuffer 区别

解析：这是一个很基础的题，String 字符串的问题几乎很多面试官都喜欢问，因为在实际项目开发中用到的最多的变量就是 String，所以读者要重点对待，熟练掌握。

参考答案：String 与 StringBuffer 在 java 中都是用来进行字符串操作的。

String 是一个很别的类，它被 final 修饰，意味着 String 类不能被继承使用，一旦声明是一个不可变的类。

StringBuffer 是一个长度可变的，通过追加的方式扩充，并且是线程安全；尤其在迭代循环中对字符串的操作，性能一般优于 String。但线程安全意味着开销加大，性能下降，所以在线程安全的情况下一般用 StringBuilder 来代替 StringBuffer 使用，StringBuilder 的速度要优于 StringBuffer。

拓展：javaAPI 中哪些类是 final 的？String 的"+"号操作性能是不是一定慢？

4.3Vector,ArrayList, LinkedList 的区别

解析：这里主要考察集合类的熟练程度，考察深度可深可浅；浅的主要是考察使用，深点可能考察底层的数据结构，算法实现等，感兴趣的朋友可以深究一下。

参考答案：

1、Vector、ArrayList 都是以类似数组的形式存储在内存中（Vector 不能存 null，但 ArrayList 可以），LinkedList 则以链表的形式进行存储。

2、Vector 线程同步，ArrayList、LinkedList 线程不同步；后两者一般用在线程安全的地方，也可以通过 Collections.synchronizedList(……);实现线程同步。

3、LinkedList 适合指定位置插入、删除操作，不适合查找；ArrayList、Vector 适合查找，不适合指定位置的插入、删除操作。

4、ArrayList 在元素填满容器时会自动扩充容器大小的 50%，而 Vector 则是 100%，因此 ArrayList 更节省空间。

5、LinkedList 还实现了 Queue 接口,该接口比 List 提供了更多的方法,包括 offer(),peek(),poll()等,多与一些线程池一起使用.

补充：以上都实现了序列化接口，这点在对象远程传输时很重要；而 Vector,ArrayList 实现了 AccessRandom 接口，这是一个标记接口，此接口的主要目的是允许一般的算法更改其行为，从而在将其应用到随机或连续访问列表时能提供良好的性能。

注意: 默认情况下 ArrayList 的初始容量非常小,所以如果可以预估数据量的话,分配一个较大的初始值属于最佳实践,这样可以减少调整大小的开销。

拓展：线性列表和双链表的数据结构实现？

4.4HashTable, HashMap,ConcurrentHashMap 区别

解析：这里还有一个 TreeMap 没有列出，TreeMap 能够把它保存的记录根据 Key 键排序，所以 Key 值不能为空，默认是按升序排序，也可以写自己的比较器（TreeMap 排序是根据红黑树实现的，有兴趣的可以研究一下它的算法实现）。而 ConcurrentHashMap 是从 jdk1.5 新增的，是 HashMap 的一种线程安全的实现类，经常在多线程的环境中使用，由于底层的实现使用的是局部锁技术，在线程安全的前提下比 HashTable 的性能要好，推荐使用（ConcurrentHashMap 局部锁技术实际上就是把 Map 分成了 N 个 Segment，put 和 get 的时候，都是现根据 key.hashCode()算出放到哪个 Segment 中，而这里的每个 segment 都相当于一个小的 HashTable,有兴趣的朋友可以读一下源代码），此题经常在中高级职位面试中问到，所以请慎重对待。

参考答案：

1、HashTable 线程同步，key,value 值均不允许为空；HashMap 非线程同步，key,value 均可为空,HashTable 因为线程安全的额外开销会造成性能下降，而 HashMap 由于线程不安全，在多线程的情况下，一般要加锁，或者使用 Collections.synchronizedMap()来创建线程安全的对象。

2、HashTable 继承的父类是 Dictionary，而 HashMap 继承的父类是 AbstractMap，而且又都实现了 Map 接口，所以 Map 接口中提供的方法，他们都可以使用；HashTable 的遍历是通过 Enumeration，而 HashMap 通常使用 Iterator 实现。

3、HashTable 有一个 contains(Object value)的方法，其功能和 HashMap 的 containsValue(Object value)一样，而 HashMap 提供的更加细致还有一个取 Key 值的方法为 containsKey(Object key)。

4、哈希值的使用不同，HashTable 直接使用对象的 hashCode，代码是这样的：

```
int hash = key.hashCode();
int index = (hash & 0x7FFFFFFF) % tab.length;
```

而 HashMap 重新计算 hash 值，而且用与代替求模：

```
   int hash = hash(k);          int i = indexFor(hash, table.length);
static int hash(Object x) {       int h = x.hashCode();
       h += ~(h << 9);
       h ^= (h >>> 14);
       h += (h << 4);
       h ^= (h >>> 10);
       return h;}
static int indexFor(int h, int length) {return h & (length-1);}
```

//网络提供，面试时不追问，可以不说出具体的实现代码。

5、ConcurrentHashMap 是 HashMap 线程安全的实现，并且逐渐取代 Hashtable 的使用，因为 Hashtable 锁的机制是对整个对象加锁，而 ConcurrentHashMap 使用的是局部锁技术，实际上就是把 Map 分成了 N 个 Segment，put 和 get 的时候，都是现根据 key.hashCode()算出放到哪个 Segment 中，而这里的每个 segment 都相当于一个小的 Hashtable，性能将高于 HashTable。

4.5equals()和 hashCode()作用

解析：这个问题每个面试求职者或许都被问到过，属于常见题型，但也是比较难回答的面试题之一；包括很多面试官本身也有时候并不能解释得很清楚，如果要回答好这个小问题，首先要了解这两个方法的实现原理（在第 I 版中不会太深入讨论）。在《Effective java》中有这样一句话：***在每个覆盖了 equals 方法的类中，也必须覆盖 hashCode 方法***，如果不这样做的话，就会违反 Object.hashCode 的通用约定，从而导致该类无法结合所有基于散列的集合一起正常动作，这样的集合包括 HashMap,Hashtable,HashSet。这里列出一下 equals 重写的规则：

1 自反性：对任意引用值 X，x.equals(x)的返回值一定为 true.

2 对称性：对于任何引用值 x,y,当且仅当 y.equals(x)返回值为 true 时，x.equals(y)的返回值一定为 true;

3 传递性：如果 x.equals(y)=true, y.equals(z)=true,则 x.equals(z)=true

4 一致性：如果参与比较的对象没任何改变，则对象比较的结果也不应该有任何改变

5 非空性：任何非空的引用值 X，x.equals(null)的返回值一定为 false。

参考答案：这两个方法都是 Object 类的方法，equals 通过用来判断两个对象是否相等。HashCode 就是一个散列码，用来在散列存储结构中确定对象的存储地址。HashMap，HashSet，他们在将对象存储时，需要确定它们的地址，而 HashCode 就是做这个用的。在默认情况下，由 Object 类定义的 hashCode 方法会针对不同的对象返回不同的整数，这一般是通过将该对象的内部地址转换成一个整数来实现的。

在 java 中 equals 和 hashCode 之间有一种契约关系：

1. 如果两个对象相等的话，它们的 hashcode 必须相等。

2. 但如果两个对象的 hashcode 相等的话，这两个对象不一定相等。

由于 java.lang.Object 的规范，如果两个对象根据 equals()方法是相等的，那么这

两个对象中的每一个对象调用 hashCode 方法都必须生成相同的整数结果。(下面作为解释内容可以不必回答，理解使用)

举例说，在 HashSet 中，通过被存入对象的 hashCode()来确定对象在 HashSet 中的存储地址，通过 equals()来确定存入的对象是否重复，hashCode()和 equals()都需要重新定义，因为 hashCode()默认是由对象在内存中的存储地址计算返回一个整数得到，而 equals()默认是比较对象的引用，如果不同时重写他们的话，那么同一个类产生的两个完全相同的对象就都可以存入 Set，因为他们是通过 equals()来确定的，这就是 HashSet 失去了意义。面试时简单的说，HashSet 的存储是先比较 HashCode，如果 HashCode 相同再比较 equals，这样做的目的是提交储存效率。所以换句方式问，如果用对象来做为 Key 值的话，要满足什么条件呢？答要重写 equals 和 HashCode 方法。

4.6sleep()和 wait()区别

解析：多线程可以说是 java 面试中的重点难点之一，也是最能筛选面试者的分水领，这个只是属于最基本的考点，尽可能的回答周全，面试的朋友可以对多线程这块多花点时间。

参考答案：

1、wait()是 Object 类的方法，在每个类中都可以被调用；而 sleep()是线程类 Thread 中的一个静态方法，无论 New 成多少对象，它都属于调用的类的。

2、sleep 方法在同步对象中调用时，会持有对象锁，其它线程必须等待其执行结束，如果时间不到只能调用 interrupt()强行打断；在 sleep 时间结束后重新参与 cpu 时间抢夺，不一定会立刻被执行。

3、wait()方法在同步中调用时，会让出对象锁。通常与 notify,notifyAll 一起使用。

4.7IO 与 NIO 的区别

解析：这是现在面试中常被问到的一个问题，尤其做后台开发的职位；NIO 是从 jdk1.4 就引入了，但是如果没有实际开发中使用过，不是很容易说的很清楚，这里从概念的层面介绍一下，希望有兴趣的朋友能更深层次的研究一下。

参考答案：IO 是早期的输入输出流，而 NIO 全名是 New IO 在 IO 的基础上新增了许多新的特性。提供的新特性包括：非阻塞 I/O，字符转换，缓冲以及通道。

1、IO 是面向流的，NIO 是面向缓冲区的。 Java IO 面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。 Java NIO 的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。

2、Java IO 的各种流是阻塞的。这意味着，当一个线程调用 read() 或 write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。 Java NIO 的非阻塞模式，使一个线程从某通道发送请求

读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取。线程通常将非阻塞 IO 的空闲时间用于在其它通道上执行 IO 操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。虽然是非阻塞的，但也会遇到一个问题就是服务器对最大连接的支持，但在线用户连接数大于系统支持数时，NIO 的默认实现是并不管是否还有足够的可用连接数，而是直接打开连接。在 netty 框架中经常会看到一个 open too many files 异常就是由此引起的，所以要灵活使用，合适配置。
3、Java NIO 的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

扩展：Netty 是 NIO 实现的一个经典框架，好多职位中都要求熟悉 netty，大家有时间可以研究一下。

4.8Synchronized 和 Lock 区别和用法

解析：这里相对于上面的 wait 与 sleep 的区别来说，难度明显增加，也是常见的面试题。

参考答案：
首先他们都是用来实现线程的同步操作的，synchronized 是 jdk1.2 以来一直存在，而且 Lock 是 jdk1.5 的新增；具体区别如下：

1、Lock 是 JDK1.5 才出现的，而在 jdk1.5 及之前的 synchronized 存在很大的性能问题，尤其在资源竞争激烈的条件下，性能下降十多倍，而此时的 Lock 还基本能维持常态。（synchronized 在 jdk1.5 后的版本，作了很大的性能优化）

2、使用 synchronized 时对象一定会被阻塞，其它线程必须等待锁释放后才能执行，在高并发的情况下，很容易降低性能，而且控制不当还容易造成死锁，所以要合适的利用 Synchronized，并且尽可能的精确到最小的业务逻辑块。而 Lock 提供了更细致的操作，其常用的实现类有读写锁，这里以 ReentrantLock 为例，它拥有 Synchronized 相同的并发性和内存语义，此外还多了锁投票，定时锁等候和中断锁等候等很多详细的方法操作。 ReentrantLock 获取锁定与三种方式：a) lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁 b) tryLock(), 如果获取了锁立即返回 true，如果别的线程正持有锁，立即返回 false；c)tryLock(long timeout,TimeUnit unit)，如果获取了锁定立即返回 true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回 true，如果等待超时，返回 false；d) lockInterruptibly: 如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到或者锁定，或者当前线程被别的线程中断。

3、synchronized 是在 JVM 层面上实现的，不但可以通过一些监控工具监控 synchronized 的锁定，而且在代码执行时出现异常，JVM 会自动释放锁定，但是使用 Lock 则不行，lock 是通过代码实现的，要保证锁定一定会被释放，就必须将 unLock()放到 finally{}中。

补充：
A．无论 synchronized 关键字加在方法上还是对象上，他取得的锁都是对象，而

不是把一段代码或函数当作锁——而且同步方法很可能还会被其他线程的对象访问。 B．每个对象只有一个锁（lock）和之相关联。
C．实现同步是要很大的系统开销作为代价的，甚至可能造成死锁，所以尽量避免无谓的同步控制。synchronized 可以加在方法上，也可以加在对象上，通常理解为，只有持有了锁才可以进行对应代码块的执行。
java.util.concurrent.locks 包下面提供了一些锁的实现，有读写锁，公平锁等。
将 synchronized 替换成 lock 的实现可以提升性能：
1. 大部分应用场景是读写互斥，写和写互斥，读和读不互斥。而 synchronized 则是都互斥。
可以利用读写锁来优化性能，读锁锁住读的代码块，写锁锁住写的代码块。
2. 要确保你在理解原来利用到 synchronized 的代码逻辑，避免一概而论地把 synchronized 替换成锁。
PS：被 synchronized 修饰的方法所在的对象，其它线程可以访问该对象的非 synchronized 修饰的方法，不能访问 synchronized 的任何方法。

说明：里面有一部分看网上总结的比较好，就直接拿过来使用了，当然面试时也不是说必须要把上面的点回答完，最重要要通过自己的理解表达出来。

4.9Final、Finally、Finalize 的区别

解析：这是一个很老很老就存在的面试题，但还是经常被问到，越简单的题在面式中越不允许答错。

参考答案：
Final 是 Java 中一个特别的关键字（修饰符），final 修饰的类不能被继承，final 修饰的变量不能被修改（是常量）,final 修饰的方法不能被重写，所以使用 final 要考虑清楚使用场景。
Finally 是 Java 异常处理 try{}catch(..){}finally{}中的一部分，finally 语句块中的语句一定会被执行到（Down 机除外），通常用于进行资源的关闭操作，如数据连接关闭等。
Finalize 是 Object 类中的一个方法，finalize() 方法是在垃圾收集器删除对象之前对这个对象调用的，设计的初终是可以进行一些资源的关闭，但这个方法很少被重写，因为垃圾回收器执行时并不一定会调用，比如抛异常时，而且垃圾回收本身就是随意的，不可控制的。

4.10 OverLoad 与 Override 的区别

解析：这里其实是考的是 Java 面向对象的三（四，如果算上抽象就是四大特征）大特征，封装、继承、多态、（抽象）中的多态，属于基础类型常考题之一。

参考答案：overload 和 override 是多态的两种表现形式；
overload 重载规则是***同名不同参***,方法名相同，参数类型、个数、顺序至少有一个不相同（与返回值、参数名、抛出的异常都没有关系）；
overload 可以发生在父类、同类、子类中。override 重写的规则是***方法名、参***

数、返回值相同，发生在父子类之间，而且子类不能抛出比父类更多的异常，被重写的方法一定不能定义成 final。
拓展：static 方法能不能被 overload,override?

4.11 Collection 与 Collections 的区别

参考答案：Collection 是一个集合的超级接口，它定义了一些集合常用的操作方法，如 add,size,remove,clear 等，我们经常用到的 List（可重复集合）,Set（不可重复集合）接口就是它的子接口。

Collections 此类完全由在 collection 上进行操作或返回 collection 的静态方法组成。它包含在 collection 上操作的多态算法，即“包装器”，包装器返回由指定 collection 支持的新 collection，以及少数其他内容。如果为此类的方法所提供的 collection 或类对象为 null，则这些方法都将抛出 NullPointerException。常用到的方法如 copy,sort,reverse,replaceall 等。

第 05 章 Java 基础知识

5.1 “==”与“equals”区别

解析：这是一个很简单的题，很多人都能答上来“==”通常用来比较地址，而且 equals 则是比较内容；如果仅仅这样回答，通常面试只能得到 60 分，可以通过下面的两个例子来看一下。

```
String s1 = "s";
String s2 ="s" ;
String s3 = new String("s");
String s4 = new String("s");
StringBuffer sb1 = new StringBuffer("s");
StringBuffer sb2 = new StringBuffer("s");
System.out.println(s1==s2) ;
System.out.println(s3==s4) ;
System.out.println(sb1==sb2) ;
System.out.println(s1.equals(s2)) ;
System.out.println(s3.equals(s4))
System.out.println(sb1.equals(sb2)) ;
```

读者可以自己判断一下结果;对于 s1,s2 会指向同一个地址，因为这里的 s1 会保存在常量池中，s2 会先读取常量池，发现里面已经存在”s”了，不会再进行重新开辟空间。对于 s3,s4 是在堆中分配两块空间，它们的地址是存放在栈中。sb1 与 sb2 的内在分配和 s3,s4 一样，但注意 StringBuffer 没有重写 equal 方法，而调用的是 Object 中的 equal 方法，看一下源码可以知道 O bject 中的 equal 的实现

return (this == obj)其实就是在比较对象的地址。

参考答案：“==”对于基本数据类型，通常用来比较值（基本数据类型是在栈中进行分配的）相等，对于对象通常用来比较对象的地址。equals 是 Object 中的方法，可以根据业务需要进行重写，比如 String 中重写了 equals 的方法，常用来比较两个字符串内容是否相等。
说明：重写 equal 的风险也很大的，上面也提到过重写的规则，所以根据需要进行重写。

5.2 接口和抽象类的区别

解析：这是一个很简单的题，面试时一定要想清楚再回答，这样的题答错，估计机会也就失去了。

参考答案：
含有 abstract 修饰符的 class 即为抽象类，abstract 类不能创建实例对象，含有 abstract 的方法的类必须定义为 abstract class ，abstract class 里的方法不必是抽象的，抽象类中定义抽象方法必须放在具体子类中实现，所以，不能有抽象的构造方法或抽象的静态方法，如果子类没有实现抽象父类中的所有方法，那么，子类也必须定义为抽象类。
接口（interface）可以说成是抽象类的特例。接口中的所有方法都必须是抽象的，接口中的方法定义默认为 public abstract 。接口中的变量是全局常量，即 public static final 修饰的。概括如下：
1、抽象类里可以有构造方法，而接口内不能有构造方法。
2、抽象类中可以有普通成员变量，而接口中不能有普通成员变量。
3、抽象类中可以包含非抽象的普通方法，而接口中所有的方法必须是抽象的，不能有非抽象的普通方法。
4、抽象类中的抽象方法的访问类型可以是 public ，protected 和默认类型，但接口中的抽象方法只能是 public 类型的，并且默认即为 public abstract 类型。
5、抽象类中可以包含静态方法，接口内不能包含静态方法。
6、抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static 类型，并且默认为 public static 类型。
7、一个类可以实现多个接口，但只能继承一个抽象类。
以上答案网上都是可以找的到的，也很有条理，如果能回答出五条左右就可以满分了。除此外，下面就是加分项了，在设计上接口的作用侧重于模块划分，比如：用户管理接口，信息管理接口，里面都相关用户的增删改查操作。而抽象类则是侧重于功能的划分，比如事先知道需要实现某个功能，但设计时却还没有想到如何实现则先抽象出一个方法，或者把子类中重复实现的功能放到抽象中去实现等。

5.3 运行时异常和一般异常的区别

解析：异常的优点是可以提高程序的健壮性，可读性，可维护性，但异常的设计初衷是用于不正常的情形，一般的 JVM 是不会试图对它进行优化，所以异常使用不当也会给系统带来负面影响。在谈异常之前，往往还会谈一下 Error，它们都是基于 Throwable 实现；但 Error 的出现通常是 JVM 层次无法自动恢复的严重例外，比如 JVM 内存泄露，一般都会造成系统荡机，所以对于 Error 进行继承设计的没有意义的。而 Exception 是可以处理的，合理的使用可以使用系统更健壮。

参考答案：运行异常（RuntimeException）和一般异常（CheckedException）都是从 Exception 继续来的，都是可以捕捉到，可以恢复的例外。

一般异常（CheckedException）：是 JVM 强制让我们处理的异常，在现在通常的开发 IDE 会提示我们必须用 try{}catch(..){}捕捉，常见的文件读写的 IO 导常，Socket 通信时的网络异常，数据操作的 JDBC 异常等。

运行异常（RuntimeException）：JVM 不强制我们去捕捉，异常一旦发生，会有 JVM 接管，一直往上抛，直到遇到异常处理的代码；如果没有处理块，到最上层，如果是多线程就由 Thread.run()抛出，如果是单线程就被 main()抛出。抛出之后，如果是线程，这个线程也就退出了。如果是主程序抛出的异常，那么这整个程序也就退出了。

异常处理的目的就是使程序从异常中恢复，不会中止线程退出系统，所以对于常的 Exception 要做一些合理的处理，但由于异常机制比较复杂而且往往得不到 JVM 的优化，会造成性能下降下，所以异常的使用最好对可能出现的异常位置进行捕捉，最好同时进行异常日志记录，便于系统维护。

5.4 序列化和反序列化

解析：对于经常接口开发的的来说，序列化这个词一定不会默生，对象本身是不能直接在网络中传输的，传递的都是一些字节流，而将对象转换成字节流的过程就叫做对象的序列化。这个考点的命中率属于一般，一般多出现于后台接口开发相关的职位面试中，但序列化与反序列化的使用却在开发中无处不在，需要深入了解。

参考答案：

1、序列化与反序列化的定义：把对象转换为字节序列的过程称为对象的序列化，把字节序列恢复为对象的过程称为对象的反序列化。

2、对象序列化的作用，比如远程接口调用，对象网络传输；深度克隆时，复杂对象的的复制；缓存对象的硬盘存储；Session 的硬盘序列化等。至于反序列化的作用，就是把序列化的对象重新加载到内存中使用。

3、序列化的实现，首先必须实现 Serializable 或者 Externalizable，而 Serializable 是超级接口，Externalizable 是其实现，Serializable 接口中什么也没有定义，它只是对对象生成一个唯一的 ID 标识，告诉 JVM 该对象是可以序列化的，如果不实现序列化接口，在序列化的过程中会抛出异常。

4、JavaAPI 提供的序列化方式，java.io.ObjectOutputStream 代表对象输出流，

它的 writeObject(Object obj)方法可对参数指定的 obj 对象进行序列化，把得到的字节序列写到一个目标输出流中。java.io.ObjectInputStream 代表对象输入流，它的 readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。对于实现了 Externalnalizable 接口的类，且类必须实现 readExternal(ObjectInput in)和 writeExternal(ObjectOutput out)方法，按照以下方式进行序列化与反序列化。ObjectOutputStream 调用该类生成对象的 writeExternal(ObjectOutput out))的方法进行序列化。ObjectInputStream 会调用对象的 readExternal(ObjectInput in)的方法进行反序列化。
扩展：Java 实现序列化带来了很多好处，但实际开发中要根据项目类型和业务场景慎重实现，因为实现序列化也会付出一定的“代价”，首先会降低类的灵活性，其实会增加出现 BUG 和漏洞的可能。

5.5 Java 实现浅克隆与深克隆

解析：这个面试命中的概率不算太高，此题目是笔者在某家公司的面试的原题，而笔者之前的工作中研究过克隆的使用，在这里总结一下供大家参考（里面用到序列化的一些知识，参考上面）。

参考答案：
所谓的浅克隆，顾名思义就是很表面的很表层的克隆，如果我们要克隆 Administrator 对象，只需要克隆他自身以及他包含的所有对象的引用地址。而深克隆，就是非浅克隆。克隆除自身以外所有的对象，包括自身所包含的所有对象实例。至于深克隆的层次，由具体的需求决定，也有“N 层克隆”一说。但是，所有的基本（primitive）类型数据，无论是浅克隆还是深克隆，都会进行原值克隆。毕竟他们都不是对象，不是存储在堆中。（Java 中所有的对象都是保存在堆中，而堆是供全局共享的。也就是说，如果同一个 Java 程序的不同方法，只要能拿到某个对象的引用，引用者就可以随意的修改对象的内部数据。）
潜克隆实现：1. 让该类实现 java.lang.Cloneable 接口；2. 重写（override）Object 类的 clone()方法。（并且在方法内部调用父对象的 clone()方法）。
对于复杂对象，比如数组、集合通过浅克隆是不能修改对象的引用的，这里要彩用深度克隆，一般是通过序列和反序列的方式来实现浓度克隆：

```
try
{
    if (src != null)
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(src);
        oos.close();

        ByteArrayInputStream bais = new ByteArrayInputStream(baos
                .toByteArray());
        ObjectInputStream ois = new ObjectInputStream(bais);

        o = ois.readObject();
        ois.close();
    }
```

收获offer，挑战高薪，更多请访问：http://shop113074087.taobao.com

```
} catch (IOException e)
{
    e.printStackTrace();
} catch (ClassNotFoundException e)
{
    e.printStackTrace();
}
return o;
```

如上面的代码，对序列化的对象写到内存或者研盘上，然后在需要的地方再从内存或者硬盘上反序列化成对象。

5.6 枚举可以序列化吗

解析：这个是笔者参加某一个著名的互联网公司的面试题，冷不丁上来这个小问题，如果你平时没太关注，很容易认为是陷阱，大江大河都淌过的人很容易死在小水塘里；而且在两家的互联网开发设计规范中都提到多使用 Enum，所以像这样的小知识点平时还是要多留意。

参考答案：因为 Enum 实现了序列化接口 Serializable，所以枚举是可以被序列化的。

5.7 Java 创建对象的方式

解析：这个没什么深意，面试官多是想了解一下你的知识面，一般人都应该至少能回答出来两种。

参考答案：

(1) 用 new 语句创建对象，这是最常见的创建对象的方法。

(2) 运用反射手段,调用 java.lang.Class 或者 java.lang.reflect.Constructor 类的 newInstance()实例方法。
(3) 调用对象的 clone()方法。
(4) 运用反序列化手段，调用 java.io.ObjectInputStream 对象的 readObject()方法。

5.8 能否自己定义一个 java.lang.String 类

解析：这个是考察面试人员对类加载机制的了解，问题不难，但需要真正了解 JVM 对类的加载原理，这个在下面 JVM 相关部分还会详细提及，这里就简答一下。

参考答案：
一般情况下是不可以的，类的 APP 加载器会根据双亲代理机制委托父类加载器去加载此类，而在父亲加载器中已经加载过此类(因为 java.lang.String 类是 JVM 定义的类)，会报该类已经存。处理方法，自定义加载器加载，指定一个其它路径。

5.9 Java 线程池的了解

解析：这里还是多线程考察的延续，在互联网职位、高并发分布式相关的职位中几乎必然会面试多线程题目，而且有资格面试这块的面试官一般都是设计、架构级别的资深人士，难度中偏上，是面试有效筛选题目类型；线程池在 jdk1.5 之前的版本并不是特别成熟，但在 Jdk1.5 的版本中作了很大的优化，在 jdk1.6 的版本中更是引入了 java.util.concurrent 包，有兴趣的读者可以详细研究一下这个包；而回答此问题可以从线程池的作用、为什么要使用线程池及线程池的优越性、线程池的实现原理和使用等几方面回答。

参考答案：
1、线程池的作用：在没用使用线程池的情况下，开发人员通常自己来开发管理线程，很容易将线程开启过多或者过少；过多将造成系统拥堵，过少又浪费系统资源。用线程池控制线程数量，其他线程排队等候。一个任务执行完毕，再从队列的中取最前面的任务开始执行。若队列中没有等待进程，线程池的这一资源处于等待。当一个新任务需要运行时，如果线程池中有等待的工作线程，就可以开始运行了；否则进入等待队列。
2、由于手动维护线程成本很高（大家都知道多线程的开发难度大，对程序员要求高，而且代码不易调试维护），所以我们应尽可能用 JDK 提供的比较成熟的线程池技术,它减少了创建和销毁线程的次数，每个工作线程都可以被重复利用，可执行多个任务。可以根据系统的承受能力，调整线程池中工作线线程的数目，以达到系统性能最优。
3、线程池的父接口是 Executor，它只定义了一个 Execute 方法，而通过使用的是它的扩展接口 ExecutorService，它有几个常用的实现类 AbstractExecutorService,

ScheduledThreadPoolExecutor, ThreadPoolExecutor（详见 API）。而在实际开发中，通过使用的是一个 java.util.concurrent 中的一个工具类 Executors，里面提供了一些静态工厂，生成常用的线程池(下面这些作为了解，说出一两个就可以了)。

i. newSingleThreadExecutor

创建一个单线程的线程池。这个线程池只有一个线程在工作，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

ii.newFixedThreadPool

创建固定大小的线程池。每次提交一个任务就创建一个线程，直到线程达到线程池的最大大小。线程池的大小一旦达到最大值就会保持不变，如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

iii. newCachedThreadPool

创建一个可缓存的线程池。如果线程池的大小超过了处理任务所需要的线程，那么就会回收部分空闲（60 秒不执行任务）的线程，当任务数增加时，此线程池又可以智能的添加新线程来处理任务。此线程池不会对线程池大小做限制，线程池大小完全依赖于操作系统（或者说 JVM）能够创建的最大线程大小。

iv.newScheduledThreadPool

创建一个大小无限的线程池。此线程池支持定时以及周期性执行任务的需求。

5.10Return 和 finally 语句块的顺序

解析：这个面试题就是考查基本功的，而且出现频率较高，面试官大部分也不会深问，但希望读者关注一下拓展中的问题，在面试笔试中都有很高的命中率。

参考答案：Finally 是在 return 语句之前被执行的,但是 finally 与 return 是相互独立的；return 语句位置会影响到返回值的结果，finally 中的计算将影响到其后 return 结果,不会影响到 try 里面 return 的结果。

扩展：Finally 做为 try{}catch(..){}finally{}一部分，如果 finally 删掉可以吗？catch()删掉呢？只留 try{}可以吗？try{}finally{}连用呢？请诸位试一下，这里也是一个考点，这里就不详细说明了。

5.11 Java 静态变量与实例变量的区别

解析：这个属于 java 面向对象中语法级别的考查，是一种基本的面试题，也是不容许失分的题目。

参考答案：Java 成员变量中静态变量用 static 修饰，而没有 static 修饰的变量称为实例变量。

Java 静态变量也叫类变量，在类加载时被分配空间（从 java 内存分配上可以知道，它是存放在静态域中），由于它是属于类的，为所有实例共享，它的特性是一改全改（任何一个实例中修改了这个属性，其它实例都能看到修改的结果），调用方式通过类名.变量名。

Java 实例变量属于对象的，在对象被实例化的时候（new 的时候）分配空间，

每个实例化的对象都有一套自己的实例变量，调用方法通常通过生成的 set/get 方法，或者通过 this.实例变量名。

5.12 同步与异步区别及使用场景

解析：同步一定是发生在多线程条件下，对共同资源不安全的访问进行控制一种手段；异步是要求减少请求方时间的等待。

参考答案：如果数据将在线程间共享.例如正在写的数据以后可能被另一个线程读到,或者正在读的数据可能已经被另一个线程写过了,那么这些数据就是共享数据,必须进行同步存取。
当应用程序在对象上调用了一个需要花费很长时间来执行的方法,并且不希望让程序等待方法的返回时,就应该使用异步编程,在很多情况下采用异步途径往往更有效。
使用场景：同步，最简单最常见的就是火车票销售，不可能多个窗口下允许出现卖两张完全一样的票。异步，在注册页面，当我们写其它信息时，我希望后台已经完成了对我姓名是否重复的验证，而不是我完全写完提交后再去验证。

5.13 Java 堆和栈的区别

解析：要想详细了解堆和栈的区别，就要了解 JavaVM 的内存分配机制，一般会涉及以下几个区域寄存器、堆、栈、静态域、常量池、非 RAM 存储（这些概念性的东西，理解就好），静态域主要是存放一些 static 静态成员、常量池存放常量，下面主要说一下堆、栈。

参考答案：栈与堆都是 Java 用来在 Ram 中存放数据的地方，与 C 和 C++不同 Java 自动管理栈和堆，程序员不能直接地设置栈或堆。
栈中存放局部变量（基本类型的变量）和对象的引用地址。栈的优势是，存取速度比堆要快，仅次于寄存器，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性；**栈是跟随线程的，有线程就有栈**。
堆中存放对象，包括对象变量以及对象方法（大学时老师可能都讲过一个类似的笑话，一 New 一堆，说的就是堆是用来存放对象的）。堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，Java 的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。**堆是跟随 JVM 的，有 JVM 就有堆内存**。
http://www.csdn123.com/html/exception/723/723513_723516_723514.htm

5.14Java 多线程快速问答

解析：多线程中有很多知识点，面试经常问到，而且会被连续问到，这里简单介绍一下。

1、Java 实现多线程的方式有哪些？

答：Java 实现多线程的方式通常有实现 Runnable 接口，或者继承 Thread 类，或者从线程 ExecutorService 中获取。

2、你通常用哪种方式来实现线程，为什么？

答：通常使用实现 Runnable 接口或者从线程池中来获得经程；实现接口是因为接口是支持多实现，我们知道“一个类只能有一个父类，但是却能实现多个接口”，因此 Runnable 具有更好的扩展性，而且 Thread 也是实现了 Runnable 接口，所以推荐通过接口实现来实现线程。另一种方式就是从线程池中获取线程，优点是将线程交给线程池去管理，可以减轻大量的精力去维持线程状态的管理，而且手动干预的越多，后期的维护成本越高。

3、run()和 start()哪个是线程的启动方法？

答：run()方法，在实现一个线程类时必须重写的方法，但它本身只是一个普通的方法，如果通过对象.run()去调用，它只是一个属于调用线程中的普通方法，可以调用多次，而且是按顺序执行。而 start()方法是线程的启动方法，当他把调用时，线程的状态就变成了就绪，一旦获取了时间片，就会立刻执行（执行 run()方法，这里的 run 是线程体，执行完成表示线程执行结束）。

4、多线程状态有哪些？

答：Java 线程有五种状态**创建、就绪、运行、阻塞和死亡**。创建：可以理解我们 new 了一线程对象；就绪：new 的线程对象调用了 start()方法，但并没有立即抢到 CPU 时间片；运行：线程启动后，线程体 run 方法在执行；阻塞：阻塞状态是指线程因为某些原因放弃 CPU，暂时停止运行。当线程处于阻塞状态时，Java 虚拟机不会给线程分配 CPU，直到线程重新进入就绪状态，它才会有机会获得运行状态；死亡：当线程执行完 run()方法中的代码或者调用了 stop()方法，又或者遇到了未捕获的异常，就会退出 run()方法，此时就进入死亡状态，该线程结束生命周期。

5、多线程的同步方法有哪些？

答：**wait():**使一个线程处于等待状态，并且释放所持有的对象的 lock。**sleep():**使一个正在运行的线程处于睡眠状态，是一个静态方法，调用此方法要捕捉 InterruptedException 异常。**notify():**唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且不是按优先级。**notifyAll():**唤醒所有处入等待状态的线程。实现线程同步一个是 synchronized 关键字，一个是通过对象 Lock。

6、你了解死锁吗（如果是笔试可能让你写一段死锁代码）？

答：死锁可以这样认为：多个线程同时被阻塞，它们中的一个或者全部都在等待某个资源被释放（例如有 a,b 两个线程，a 等待 b 让出了某个资源才可以执行；而 b 同样等待 a 让出了某个资源才可以继续，ab 都在等待对方就是线程同时阻塞，结局就是死锁）。

7、当一个线程进入到一个对象的 synchronized 方法，那么其他线程是否可以进入该对象的其它方法？

答：当一个线程进入到一个对象的 synchronized 方法，那么其他线程是可以进入这个对象的非 synchronized 方法，但不可能进入 synchronized 方法

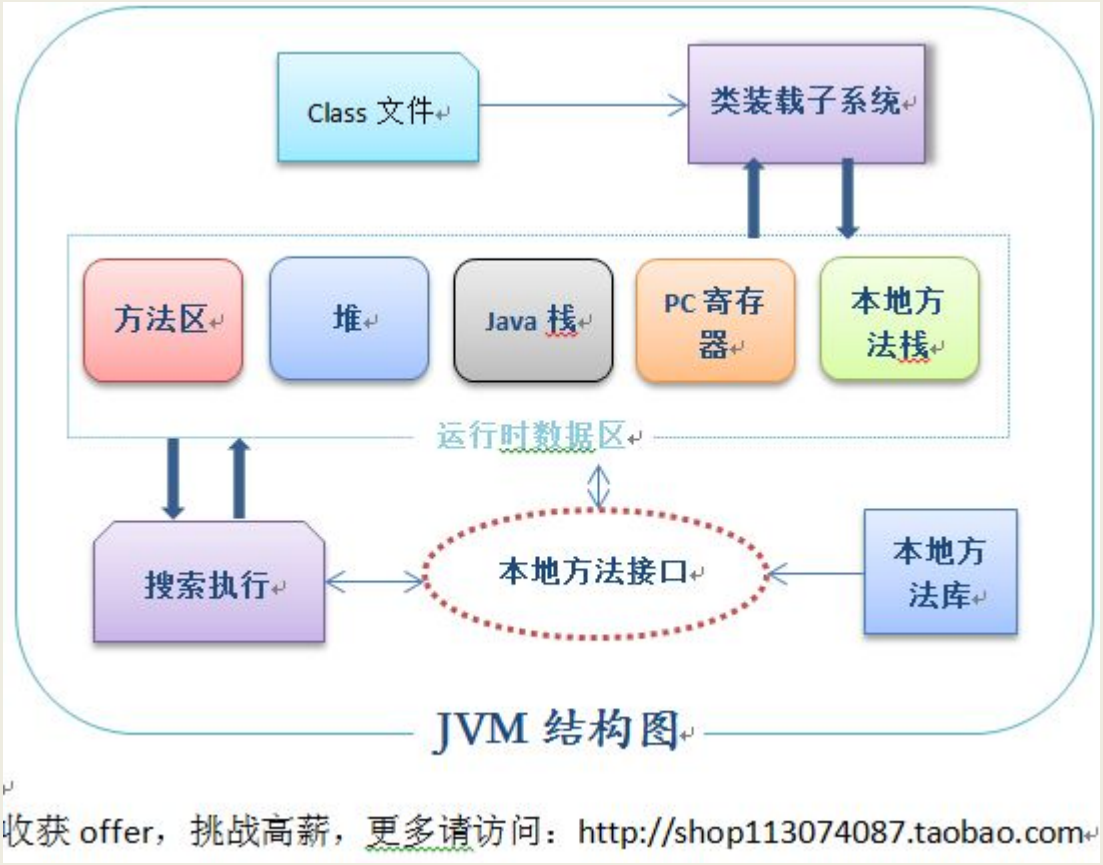
扩展：多线程面试是 Java 面试中不可缺少的一部分，也是属于面试中的难点，绝对是面试中的淘汰型问题，需要多花点时间准备。

第 06 章　JVM 相关

6.1　JVM 的工作原理

解析：很多大的互联网公司笔试时，会让你画一下 JVM 的结构图，然后让你解释一下各部分的意义，所以这个面试命中率还是很高的，作为基础扎实的求职者是应该要会的。纠错：搜索执行--->执行引擎。

参考答案：下面是我画的一个 JVM 的结构图



收获 offer，挑战高薪，更多请访问：http://shop113074087.taobao.com

1. 类装载器子系统：

负责查找并装载 Class 文件到内存，最终形成可以被虚拟机直接使用的 Java 类型。

2. 方法区：

在类装载器加载 class 文件到内存的过程中，虚拟机会提取其中的类型信息，并将这些信息存储到方法区。方法区用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。由于所有线程都共享方法区，因此它们对方法区数据的访问必须被设计为是线程安全的。

3. 堆：

存储 Java 程序创建的类实例。所有线程共享，因此设计程序时也要考虑到多线程访问对象(堆数据)的同步问题。

4. Java 栈：
Java 栈是线程私有的。每当启动一个新线程时，Java 虚拟机都会为它分配一个 Java 栈。Java 栈以帧为单位保存线程的运行状态。虚拟机只会直接对 Java 栈执行两种操作：以帧为单位的压栈或出栈。当线程调用 java 方法时，虚拟机压入一个新的栈帧到该线程的 java 栈中。当方法返回时，这个栈帧被从 java 栈中弹出并抛弃。一个栈帧包含一个 java 方法的调用状态，它存储有局部变量表、操作栈、动态链接、方法出口等信息。
5. 程序计数器：
一个运行中的 Java 程序，每当启动一个新线程时，都会为这个新线程创建一个自己的 PC(程序计数器)寄存器。程序计数器的作用可以看做是当前线程所执行的字节码的行号指示器。字节码解释器工作时就是通过改变这个计数器的值来选取下一条需要执行的字节码指令，分支、循环、跳转、异常处理、线程恢复等基础功能都需要依赖这个计数器来完成。如果线程正在执行的是一个 Java 方法，这个计数器记录的是正在执行的虚拟机字节码指令的地址；如果正在执行的是 Natvie 方法，这个计数器值则为空(Undefined)。
6. 本地方法栈：
本地方法栈与虚拟机栈所发挥的作用是非常相似的，其区别不过是虚拟机栈为虚拟机执行 Java 方法（也就是字节码）服务，而本地方法栈则是为虚拟机使用到的 Native 方法服务。任何本地方法接口都会使用某种本地方法栈。当线程调用 Java 方法时，虚拟机会创建一个新的栈帧并压入 Java 栈。然而当它调用的是本地方法时，虚拟机会保持 Java 栈不变，不再在线程的 Java 栈中压入新的帧，虚拟机只是简单地动态链接并直接调用指定的本地方法。如果某个虚拟机实现的本地方法接口是使用 C 连接模型的话，那么它的本地方法栈就是 C 栈。
7. 执行引擎：
负责执行字节码。方法的字节码是由 Java 虚拟机的指令序列构成的。每一条指令包含一个单字节的操作码，后面跟随 0 个或多个操作数。执行引擎执行字节码时，首先取得一个操作码，如果操作码有操作数，取得它的操作数。它执行操作码和跟随的操作数规定的动作，然后再取得下一个操作码。这个执行字节码的过程在线程完成前将一直持续。

6.2 类的加载机制

解析：这个题目看上去不太容易，但实际上在学习 Java 第一节课的时候，这些知识老师都已经告诉我们了，第一节课就是安装 JDK 配置环境变量，PATH,JAVA_HOME 大家肯定不默生，但随着 IDE 的使用，CLASSPATH 环境变量被慢慢遗忘了，CLASSPATH 就是配置我们自己开发的项目路径，JVM 在加载我们写的类时，会根据 classpath 从左到右各个项目中依次查找是否有这个包，包下是否有这个类，找到类后会验证其中的 packeage 包的路径是否与参数中的路径一致，是则加载，不是会报 NotFoundClassException 的异常。（这个刚毕业的同学可能比有工作经验的朋友印象更深刻）

参考答案：要了解类的加载机制，首先要了解 JVM 使用到的类的加载器；这里主要有 bootstrap classloader、extension cassLoader、AppClassLoader 三个类的加载器，第一个是 C++写的，两后两个都是 JAVA 写的，JVM 还给我们开放一个

ClassLoader 来扩展自己的加载器,当然后两个加载器也是继承了 ClassLoader。bootstrap classloader 在 JVM 运行的时候加载 java 核心 API 以满足 java 程序最基本的需要，其中就包括用户定义的 ClassLoader， ExtClassLoader 是用来加载 java 的扩展 API 的，也就是/lib/ext 中的类。AppClassLoader 是用来加载用户机器上 CLASSPATH 设置目录中的 Class 的，通常在没有指定 ClassLoader 的情况下，程序员自定义的类就由该 AppClassLoader 来进行加载。

JVM 为了避免类的重复加载，引入了委托加载机制，比如 A 中定义 B 的引用，对于 B 的加载是通过委托 A 的加载器去加载;在上面我们也提到了双亲委托加载机制，APP 加载器加载一个类时，首先会调 ExtClassLoader, ExtClassLoader 会调用 bootstrap classloader 如果发现类已经加载就不再加载。

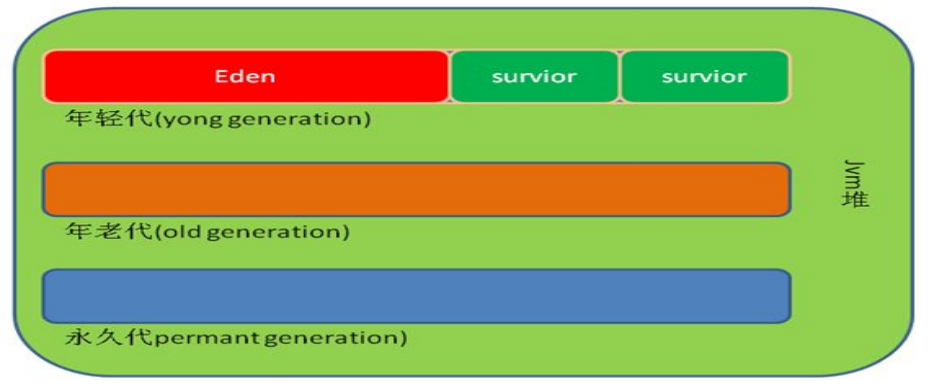
扩展：回答完上面这些应该已经差不多了，求职者可以多关注一下，比如预加载，被动加载等。

6.3 GC 的工作原理

解析：Java 的垃圾回收机制是 java 语言优越性的一大特色，也奠基它的高级语言的地位。这是一个初级、中级、高级、专家级职位都喜欢面试的问题，因为只有熟练了解 JavaGC 工作原理，才能写出质量更高的代码，才能更好的在代码级别做性能优化

参考答案：现在市场上 JVM 的实现厂家很多，不同的厂家对垃圾回收的算法实现也不同，目前主流上常的算法有**标记清除、分代、复制、引用记数、增量回收**等，而目前最常用的 JVM 是 SUN 公司(现被 Oracle 收购)的 HotSport，它采用的算法为分代回收。

HotSport 将 jvm 中堆空间划分为三个代：年轻代（Young Generation）、年老代（Old Generation）和永久代（Permanent Generation）。年轻代和年老代是存储动态产生的对象。永久带主要是存储的是 java 的类信息，包括解析得到的方法、属性、字段等等。永久带基本不参与垃圾回收。我们这里讨论的垃圾回收主要是针对年轻代和年老代。具体如下图（网络提供）



年轻代又分成 3 个部分，一个 eden 区和两个相同的 survior 区。刚开始创建的对象都是放置在 eden 区的。分成这样 3 个部分，主要是为了生命周期短的对象尽量留在年轻代。当 eden 区申请不到空间的时候，进行 minorGC，把存活的对象拷贝到 survior。年老代主要存放生命周期比较长的对象，比如缓存对象。具

体 jvm 内存回收过程描述如下（可以结合上图）：
1、对象在 Eden 区完成内存分配
2、当 Eden 区满了，再创建对象，会因为申请不到空间，触发 minorGC，进行 young(eden+1survivor)区的垃圾回收
3、minorGC 时，Eden 不能被回收的对象被放入到空的 survivor（Eden 肯定会被清空），另一个 survivor 里不能被 GC 回收的对象也会被放入这个 survivor，始终保证一个 survivor 是空的
4、当做第 3 步的时候，如果发现 survivor 满了，则这些对象被 copy 到 old 区，或者 survivor 并没有满，但是有些对象已经足够 Old，也被放入 Old 区 XX:MaxTenuringThreshold
5、当 Old 区被放满的之后，进行 fullGC。
由于 fullGC 会造成很大的系统资源开销，所以一般情况下通过代码规范和 JVM 参数设置来减少 fullGC 的调用，提高性能。
上面只是从算法实现上来说明的，目前主流的垃圾收集器主要有三种：串行收集器、并行收集器、并发收集器。 这里有兴趣的朋友可以查一下，绝对是加分点，如果在串行的垃圾回收器进行 fullGC 会造成应用的短暂中断，这是一个很危险的事情，所以一定要了解清楚各种实现方式的区别。

6.4 Java 的反射机制

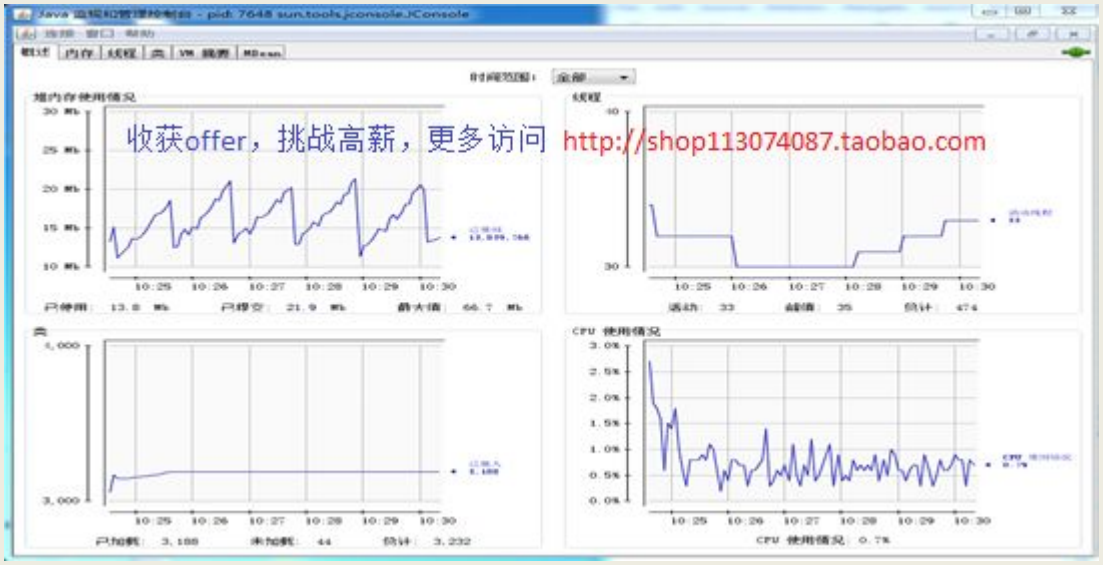
解析：这个概念定义起来比较简单，但是使用却不容易，现在很多主流的开源框架中 DI,IOC 的实现很多都是用 Java 反射机制来做的，可以适当深入研究一下。

参考答案：Java 发射机制可以让我们在运行时加载，探知，使用编译期间完全未知的 classes。换句话说就是 Java 程序可以加载一个在运行时才得知名称的 class，获悉其完整构造，并生成其对象实体，或对其 fields 设值，或调用其 methods。
反射的作用：在运行的时判定任意一个对象所属的类；运行时，构造任意一个类的对象；运行时，判定一个类所属的成员变量和方法；在运行时调用任意的一个方法；生成动态代理。
反射的实现步骤（不问不需要答），1、获取类的常用方式有三种：a) Class.forName("包名.类名")，最常用、推荐；b) 包名.类名.class 最简捷；c) 对象.getClass 的方式获得。2、对象的实例化,上面已经获取了类，只需要调用类的实例化方法,类.newInstance()便可。3、获取属性和构造等，可以参考 JavaApi 的调用，类. getDeclaredFields,类. getConstructor(..)等。

6.5 JDK 性能分析工具及应用

解析：这里主要针对一定工作经验的人，而且工作经验越久可能问的越详细，主要考察对 JDK 性能的监控，用以性能调优。

参考答案：JDK 性能分析工具很多，像 **jprofiler** 及 **yourkit** 是性能十分优越，但使用费用也不低，对于开发人员的对性能这块的监控 JDK 本身就给提供了很多优秀的免费的监控工具，如 Visualvm（如果 jdk 自带版本低，可以单独下载）是一款堪比上面商业性能的监控工具、**jconsole**（下面是我的一个应用截图）能够监视和管理查看堆内存，线程，类，CPU 状况。直接双击就可以启动了，然后选择连接本地 local 还是远程 remote，而且支持控制台管理；**jstack** 主要用于线程死锁的监控；**jmap** 主要用于监控内存泄露时候对象占用的字节数；**jstat** 主要用于监控 jvm 的 gc 使用情况；**jhat** 主要用于分析 jmap 产生的 dump 并提供 web 页面查看分析结果。



Java