JAVA 面试题

基础

1、String 编码 UTF-8 和 GBK 的区别?

UTF8 是国际编码，它的通用性比较好
GBK 是国家编码，通用性比 UTF8 差，不过 UTF8 占用的数据库比 GBK 大~

GBK 的文字编码是双字节来表示的，即不论中、英文字符均使用双字节来表示，
UTF-8 英文使用 8 位（即一个字节），中文使用 24 为（三个字节）来编码。

2、字节流与字符流的区别

InputStream 和 OutputStream,两个是为字节流设计的,主要用来处理**字节或二进制**对象,
Reader 和 Writer.两个是为字符流（**一个字符占两个字节**）设计的,主要用来处理**字符或字符串.**
实际上字节流在操作时本身不会用到缓冲区（内存），是文件本身直接操作的，而**字符流在操作时使用了缓冲区**，通过缓冲区再操作文件，

3、什么是 java 序列化，如何实现 java 序列化

序列化就是一种用来处理对象流的机制，所谓对象流也就是将对象的内容进行**流化**。可以对流化后的对象进行读写操作，也可将流化后的对象传输于网络之间。

集合

1、ArrayList 和 Vector 的区别

1、ArrayList 和 Vector 都实现了 List 接口，底层都是基于 Java 数组来存储集合元素
2、ArrayList 使用 transient 修饰了 elementData 数组，而 Vector 则没有
3、Vector 是 ArrayList 的线程安全版本
4、容量扩充 ArrayList 为 0.5 倍+1，而 Vector 若指定了增长系数，则新的容量=”原始容量+增长系数”，否则增长为原来的 1 倍

参考：http://blog.itmyhome.com/2017/07/java-arraylist-vector

2、HashMap 和 Hashtable 的区别

相同点：
都是存储键值对(key-value)的散列表，通过 table 数组存储，数组的每一个元素都是一个 Entry 而一个 Entry 就是一个单向链表，Entry 链表中的每一个节点保存了 key-value 键值对数据

不同点：
1、继承和实现方式不同
2、线程安全不同
3、对 null 值处理不同
4、支持的遍历种类不同
5、容量的初始值和增加方式不同
6、添加 key-value 时的 hash 值算法不同

参考：http://blog.itmyhome.com/2017/08/hashmap-hashtable-difference

3、List 和 Map 区别

一个是存储单列数据的集合，另一个是存储键和值这样的双列数据的集合
List 中存储的数据是有顺序的，并且**允许重复**，Map 中存储的数据是没有顺序的，其键是不能重复的他的值是可以重复的。

4、List、Map、Set 三个接口，存取元素时，各有什么特点

首先，List 与 Set 具有相似性，他们都是单列元素的集合，所以，他们有一个共同的父接口叫 Collection，Set 里面不允许有重复的元素，List 可以重复。List 表示有先后顺序的集合，Set 无序的。
Map 与 List 和 Set 不同，它是双列的集合，其中有 put 方法，每次存储时，要存储一堆 key/value，不能存储重复的 key

5、说出 ArrayList、Vector、LinkedList 的存储性能和特性

ArrayList 和 Vector 都是使用**数组**方式存储数据，此数据元素大于实际存储的数据以便增加和插入元素，他们都运行直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以**索引数据快而插入数据慢**
Vector 由于使用了 **synchronized 方法(线程安全)** 通常性能上较 ArrayList 差

而 LinkedList 使用**双向链表**实现存储，按序号索引数据需要进行前后或后向遍历，但是插入数据时只需记录本项的前后项即可，所以**插入速度较快**。

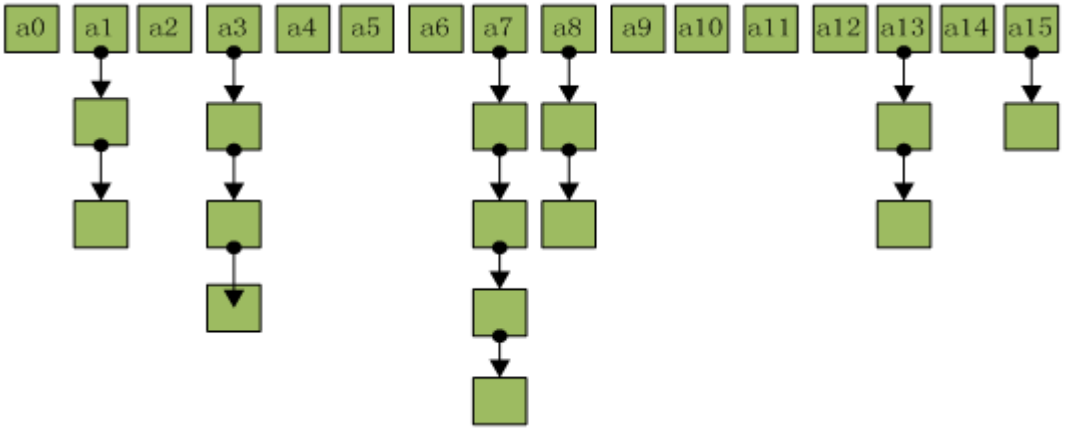
LinkedList 也是线程不安全的，LinkedList 提供了一些方法(get、remove、insert)，使得 LinkedList 可以被当作堆栈和队列来使用

6、HashMap 底层

HashMap 是基于**哈希表的 Map 接口的非同步实现**，并允许使用 null 值和 null 键，此类不保证映射的顺序 HashMap 底层就是一个数组结构，数组中的每一项又是一个链表，Entry 就是数组中的元素，每个 Map.Entry 就是一个 key-value 对 当我们给 put()方法传递键和值时，我们先对键调用 hashCode()方法得到这个对象的 hashcode 值之后，系统会根据该 hashCode 值来决定该元素的存储位置。

7、HashMap 存储结构

HashMap 采取数组加链表的存储方式来实现。亦即数组（散列桶）中的每一个元素都是链表，如下图：



8、Collection 和 Collections 的区别

Collection 是集合类的上级接口，继承与它的接口主要有 Set 和 List
Collections 是针对集合类的一个帮助类，提供一系列静态方法(addAll、binarySearch、copy、fill、reverse，sort)实现对各种集合的搜索、排序、线程安全化等操作

线程、并发

1、Java 中有几种方法可以实现一个线程？

1、继承 Thread 类
2、实现 Runnable 接口
3、使用 ExecutorService、Callable、Future 实现有返回结果的多线程(JDK5.0 以后)

2、如何停止一个正在运行的线程？

使用 interrupt

3、notify()和 notifyAll()有什么区别？

notify 表示当前的线程已经放弃对资源的占有，通知等待的线程来获得对资源的占有权。但是只有线程能够从 wait 状态中回复，然后继续运行 wait 方面的语句。

notifyAll 表示当前的线程已经放弃对资源的占有，通知所有的等待线程来 wait 方法后面的语句开始运行

4、sleep()和 wait()有什么区别?

sleep 是 Thread 类的静态方法，sleep 的作用是让线程休眠指定的时间，在时间到达时恢复，也就是说 sleepj 将在时间到达后恢复线程执行。
wait 是 Object 中的方法，也就是说可以对任意一个对象调用 wait 方法，调用 wait 方法将会将线程挂起直到使用 notify 方法才能重新唤醒。

5、Java 如何实现多线程之间的通讯和协作

生产者和消费者模式，wait()、notify()方法

6、什么是线程安全？线程安全是怎么完成的

线程安全就是说多线程访问同一代码，不会产生不确定的结果。编写线程安全的代码是依靠线程同步，线程安全一般都涉及到 synchronized 就是一段代码同时只能有一个线程来操作

7、servlet 是线程安全的吗

不是。当 Tomcat 接收到 Client 的 HTTP 请求时，Tomcat 从线程池中取出一个线程，之后找到该请求对应的 Servlet 对象并初始化，
调用 service()方法，只有一个实例对象。如果在 servlet 中定义了实例变量或静态变量那么可能会发生线程安全问题

8、同步有几种实现方法

- wait() 使一个线程处于等待状态
- sleep()使一个正在运行的线程处于睡眠状态
- notify()唤醒一个处于等待状态的线程
- allnotify()唤醒所有处于等待状态的线程

9、volatile 有什么用？能否用一句话说明下 volatile 的应用场景？

关键字 volatile 的主要作用是使变量在这个线程中可见

10、HashMap 与 ConcurrentHashMap 的区别

我们知道 HashMap 不是线程安全的，Hashtable 是线程安全的，同步的，synchronized 是针对整张 Hash 表的，即每次锁住 整张表让线程独占，ConcurrentHashMap 允许多个修改操作并发进行，其关键在于使用了**锁分离技术**。它使用了多个锁来控制对 hash 表的不同部分进行的修改。
ConcurrentHashMap 内部使用**段(Segment)**来表示这些不同的部分，每个段其实就是一个小的 hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

ConcurrentHashMap 是 Java5 中新增的一个线程安全的 Map 集合，可以用来代替 Hashtable，它使用了锁分离技术

11、Lock 与 synchronized 的区别

ReentrantLock 与 synchronized 有相同的并发性和内存语义。由于 synchronized 是在 JVM 层面实现的，因此系统可以监控锁的释放与否，而 ReentrantLock 使用代码实现的，系统无法自动释放锁，需要在代码中 finally 子句中显式释放锁 lock.unlock();
在并发量比较小的情况下，使用 synchronized 是个不错的选择，但是在并发量比较高的情况下，其性能下降很严重，此时 ReentrantLock 是个不错的方案。

(使用 synchronized 获取锁的线程由于要等待 IO 或者其他原因被阻塞了，但是又没有释放锁，其他线程只能干巴巴的等待，而 Lock 可以只等待一定的时间或者能够响应中断)

---补充---
1）synchronized 是 Java 的关键字，因此是 Java 的内置特性，是基于 JVM 层面实现的。而 Lock 是一个 Java 接口，是基于 JDK 层面实现的，通过这个接口可以实现同步访问；
2）采用 synchronized 方式不需要用户去手动释放锁，当 synchronized 方法或者 synchronized 代码块执行完之后，系统会自动让线程释放对锁的占用；而 Lock 则必须要用户去手动释放锁，如果没有主动释放锁，就有可能导致死锁现象。

12、什么是死锁

死锁就是两个或两个以上的线程被无限的阻塞，线程之间相互等待所需资源。这种情况可能发生在当两个线程尝试获取其它资源的锁，而每个线程又陷入无限等待其它资源锁的释放，除非一个用户进程被终止。就 JavaAPI 而言，线程死锁可能发生在以下情况。
*当两个线程相互调用 **Thread.join ()**
*当两个线程使用嵌套的同步块，一个线程占用了另外一个线程必需的锁，互相等待时被阻塞就有可能出现死锁。

13、什么是线程饿死，什么是活锁？

线程饿死和活锁虽然不像是死锁一样的常见问题，但是对于并发编程的设计者来说就像一次邂逅一样。当所有线程阻塞，或者由于需要的资源无效而不能处理，不存在非阻塞线程使资源可用。JavaAPI 中线程活锁可能发生在以下情形：
*当所有线程在程序中执行 Object.wait (0)，参数为 0 的 wait 方法。程序将发生活锁直到在相应的对象上有线程调用 Object.notify ()或者 Object.notifyAll ()。
*当所有线程卡在无限循环中。

14、Java 中的队列都有哪些，有什么区别。

阻塞队列与普通队列的区别在于，当队列是空的时，从队列中获取元素的操作将会被阻塞，或者当队列是满时，往队列里添加元素的操作会被阻塞。试图从空的阻塞队列中获取元素的线程将会被阻塞，直到其他的线程往空的队列插入新的元素。同样，试图往已满的阻塞队列中添加新元素的线程同样也会被阻塞，直到其他的线程使队列重新变得空闲起来，如从队列中移除一个或者多个元素，或者完全清空队列。

Queue 接口与 List、Set 同一级别，都是继承了 Collection 接口。LinkedList 实现了 Queue 接口。我们平时使用的一些常见队列都是非阻塞队列，比如 PriorityQueue、LinkedList(LinkedList 是双向链表，它实现了 Dequeue 接口)

ArrayBlockingQueue、LinkedBlockingQueue、PriorityBlockingQueue、DelayQueue

15、线程和进程的区别

线程是进城的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。

JVM 虚拟机

1、JVM 有什么了解，底层是怎么实现的

Java 内存区分为堆内存(Heap)和栈内存(Stack)
Java 堆是 Java 虚拟机所管理的内存中最大的一块，Java 堆是被所有线程共享的一块内存区域，在虚拟机启动时创建。此内存区域的唯一目的就是存放对象实例，几乎所有的对象实例都在这里分配内存。
Java 堆是垃圾收集器管理的主要区域，从内存回收的角度来看，由于现在收集器基本都采用分代收集算法， 所以 Java 堆还可以细分为 新生代和老年代

2、JVM 加载 class 文件的原理机制

JVM 中类的装载是由 ClassLoader 和它的子类来实现的，Java ClassLoader 是一个重要的 Java 运行时系统组件。它负责在运行时查找和装入类文件的类。

3、heap 和 stack 有什么区别

Java 的内存分为两类，一类是栈内存，一类是堆内存。栈内存是指程序进入一个方法时，会为这个方法单独分配一块私属存储空间，用于存储这个方法内部的存储变量，当这个方法结束时，分配给这个方法的栈会释放，这个栈中的变量也将随之释放。
堆是与栈作用不同的内存，一般用于存放不放在当前方法栈中的那些数据，例如，使用 new 创建的对象都放在堆里，所以，它不会随方法的结束而消失。方法中的局部变量使用 final 修饰后，放在堆中而不是栈中。

4、GC 是什么？为什么要有 GC？

GC 是垃圾收集的意思，内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。

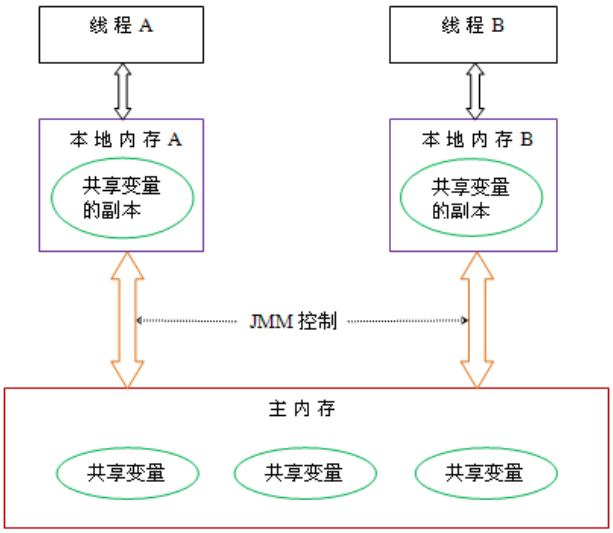
5、垃圾回收期的基本原理是什么

对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆中的所有对象。通过这种方式确定哪些对象是“可达的”哪些对象是“不可达的”当 GC 确定一些对象为“不可达”时，GC 就有责任回收这些内存空间。程序员可以手动执行 System.gc() 通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行

6、Java 内存模型

Java 内存区分为堆内存和栈内存，所有实例域、静态域和数组元素存储在堆内存中，堆内存在线程之间共享，局部变量，方法定义参数和异常处理器参数不会在线程之间共享，它们不会有内存可见性问题，也不受内存模型的影响。

Java 线程之间的通信由 java 内存模型控制，JMM 决定一个线程对共享变量的写入何时对另一个线程可见。从抽象的角度来看，JMM 定义了线程和主内存之间的抽象关系，线程之间的共享变量存储在主内存中，每个线程都有一个私有的本地内存，本地内存中存储了该线程以读/写共享变量的副本。



从上图来看，线程 A 与线程 B 之间如要通信的话，必须要经历下面 2 个步骤：
1、首先，线程 A 把本地内存 A 中更新过的共享变量刷新到主内存中去。
2、然后，线程 B 到主内存中去读取线程 A 之前已更新过的共享变量。

参考：http://www.infoq.com/cn/articles/java-memory-model-1

MySQL

1、MySQL 事务隔离级别

隔离级别	脏读	不可重复读	幻读
读未提交（Read uncommitted）	V	V	V
读已提交（Read committed）	X	V	V
可重复读（Repeatable read）	X	X	V
可串行化（Serializable ）	X	X	X

2、MySQL 锁机制

MySQL 有三种锁的级别：页级、表级、行级
MyISAM 和 MEMORY 存储引擎采用的表级锁；InnoDB 存储引擎既支持行级锁，也支持表级

锁，但默认情况下是采用行级锁。

3、MySQL 存储引擎

MyISAM

它不支持事务，也不支持外键，访问速度快，对事务完整性没有要求或者以 SEIECT INSERT 为主的应用基本都可以使用这个引擎来创建表。

InnoDB

InnoDB 存储引擎提供了具有提交，回滚和崩溃恢复能力的事务安全。但是对比 MyISAM 的存储引擎 InnoDB 写的处理效率差一些并且会占用更多的磁盘空间以保留数据和索引

MEMORY

如名字所指明的，MEMORY 表存储在内存中，且默认使用哈希索引。这使得它们非常快，并且对创建临时表非常有用。可是，当服务器关闭之时，所有存储在 MEMORY 表里的数据丢失。

4、MySQL Limit 分页优化

索引、子查询、表连接

5、共享锁与排他锁

http://blog.itmyhome.com/2017/06/mysql-share-lock

6、乐观锁与悲观锁

http://blog.itmyhome.com/2017/06/mysql-optimistic-lock-and-pessimistic-lock

算法

1、冒泡排序

```java
public class Test {
    public static void main(String[] args) {
        int temp[] = {13,52,3,8,5,16,41,29};
        //执行temp.length次
        for (int i = 0; i < temp.length; i++) {
            for (int j = 0; j < temp.length-i-1; j++) {
                if(temp[j]>temp[j+1]){ //前一个数和后一个数比较
                    int a = temp[j];
                    temp[j] = temp[j+1];
                    temp[j+1] = a;
                }
            }
        }
        for (int i = 0; i < temp.length; i++) {
            System.out.print(temp[i]+" ");
        }
    }
}
```

2、选择排序

原理：首先在未排序列中找到最小元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小元素，然后放到已排序序列的末尾。依次类推，直到所有元素均排序完毕。

3、快速排序

所谓的快速排序的思想就是，首先把数组的第一个数拿出来作为一个 key,在前后分别设置一个 i,j 作为标识，然后拿这个数组从后面往前遍历， 及 j- -,直到找到第一个小于这个 key 的那个数然后交换这两个值，交换完成后，我们拿着这个 key 要从 i 往后遍历了，及 i++ 一直循环到 i=j 结束，
当结束后，我们会发现大于这个 key 的值都会跑到这个 key 的后面，小于这个 key 的值就会跑到这个值的前面,然后我们对这个分段的数组再进行
递归调用就可以完成这个数组的排序。

Spring

1、事务传播属性和隔离级别

Spring 在 TransactionDefinition 接口中规定了 7 种类型的事务传播行为， 它们规定了事务方法和事务方法发生嵌套调用时事务如何进行传播：

事务传播行为类型	说明
PROPAGATION_REQUIRED	如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。
PROPAGATION_SUPPORTS	支持当前事务，如果当前没有事务，就以非事务方式执行。
PROPAGATION_MANDATORY	使用当前的事务，如果当前没有事务，就抛出异常。
PROPAGATION_REQUIRES_NEW	新建事务，如果当前存在事务，把当前事务挂起。
PROPAGATION_NOT_SUPPORTED	以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
PROPAGATION_NEVER	以非事务方式执行，如果当前存在事务，则抛出异常。
PROPAGATION_NESTED	如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则执行与PROPAGATION_REQUIRED类似的操作。

在 Spring 中定义了四种不同的事务隔离级别：

事务隔离级别	说明
read_uncommitted	读未提交，一个事务可以操作另外一个未提交的事务，不能避免脏读，不可重复读，幻读，隔离级别最低，并发性能最高
read_committed	读已提交，一个事务不可以操作另外一个未提交的事务， 能防止脏读，不能避免不可重复读，幻读。
repeatable_read	能够避免脏读，不可重复读，不能避免幻读
serializable	隔离级别最高，消耗资源最低，代价最高，能够防止脏读， 不可重复读，幻读

2、Spring 注解@Resource 和@Autowired 区别对比

1、@Autowired 与@Resource 都可以用来装配 bean，都可以写在字段上或写在 setter 方法上
2、@Autowired 默认按类型装配(这个注解是属于 Spring 的)
@Resource(这个注解属于 J2EE)的，默认按照名称进行装配。

3、Spring 常用注解

@Component
是所有收 Spring 管理组件的通用形式，@Component 注解可以放在类的头上，不推荐使用。

@Controller
对应表现层的 Bean，也就是 Action

@Service
对应的是业务层 Bean

@Repository
对应数据访问层 Bean

@Resource 和@Autowired
都是做 bean 的注入时使用

@RequestMapping
是一个用来处理请求地址映射的注解，可用于类或方法上

SpringMVC

1、SpringMVC 工作原理

1、客户端请求提交到 DispatcherServlet
2、由 DispatcherServlet 控制器查询一个或多个 HandlerMapping，找到处理请求的 Controller
3、DispatcherServlet 将请求提交到 Controller
4、Controller 调用业务逻辑处理后，返回 ModelAndView
5、DispatcherServlet 查询一个或多个 ViewResoler 视图解析器，找到 ModelAndView 指定的视图
6、视图负责将结果显示到客户端 DispatcherServlet 是整个 Spring MVC 的核心。它负责接收 HTTP 请求组织协调 Spring MVC 的各个组成部分。

其主要工作有以下三项：
1. 截获符合特定格式的 URL 请求。
2. 初始化 DispatcherServlet 上下文对应的 WebApplicationContext，并将其与业务层、持久化层的 WebApplicationContext 建立关联。
3. 初始化 Spring MVC 的各个组成组件，并装配到 DispatcherServlet 中。

持续更新

2017-10-09