1. equals和==的区别

equals比较的是字符串内容是否相同

==比较的是值

1. 多态
2. 多态性的核心在于方法的覆写
3. 通过对象的向上转型可以实现接收参数的统一, 向下转型可以实现子类扩充方法的调用
4. 两个没有关系的类对象是不能进行转型的,回产生ClassCastException

方法的多态性:

override：覆写

1. 继承关系中，父类中有，子类模仿父类写一个。
2. 方法名和参数列表都和父类相同。
3. 覆写的访问权限不能比父类严格。

Overload：

1. 一个类中，觉得不够再写一个
2. 方法名相同，参数列表不同，与返回值无关

对象的多态性：

1. 向上转型：父类 父类对象 = 子类对象。可以实现接收参数的统一。
2. 向下转型：子类 子类对象 = （子类）父类实例。存在安全隐患，最好的做法是先进行判断，然后进行转型，可以依靠instanceof关键字
3. 抽象和接口的区别：
4. 抽象类：在面向对象概念中，所有的对象都是通过类来描述的，但是反过来，并不是所有的类都是用来描绘对象的，如果一个类中没有包含足够的信息来描述一个具体的对象，这个类就是抽象类。
5. 抽象类除了不能实例化对象，类的其他功能依然存在，成员变量，成员方法和构造方法的访问方式和普通类一样。
6. 由于抽象类不能实例化对象，所以抽象类必须被继承才能使用。
7. 接口是指被别人调用的方法或者函数
8. 接口中可以含有变量和方法，但是接口的。变量会被隐式的指定为public static final变量，方法则会被隐式指定为public abstract方法，所以接口中的所有方法都是抽象方法，不能有具体实现。
9. 区别：
   1. 一个类可以继承一个抽象类，但是可以实现多个接口。
   2. 抽象类中的成员变量可以有很多类型，而接口只能是public static final。
   3. 抽象类可以定义普通属性和静态常量属性，而接口只能定义public static final。
   4. 抽象类可以提供成员方法的实现细节，而接口只能存在public abstract。
10. List Set Map的区别：

1. list和set都是实现了collection接口。

2. list：

1. 有get方法
2. 可以允许重复的对象；
3. 可以插入多个null值；
4. 有序容器，保证了每个元素的插入和输出顺序；
5. 常用的实现类有ArrayList（索引的随机访问），LinkedList（list中添加或删除元素）和Vector。

3. set：

1. 没有get方法
2. 不允许重复对象；
3. 只允许一个null值；
4. 无序容器。TreeSet通过Comparator或Comparable维护了一个排序顺序；
5. 常用实现类：HashSet、LinkedHashSet、TreeSet（实现了SortedSet接口）。

4. Map：

1. Map不是collection的子接口或者实现类，它是一个接口；
2. Map的每个Entry都持有两个对象，key和value，key只能有一个，value可以有多个；
3. TreeMap也通过Comparator或者Comparable维护了一个排序顺序；
4. 常见实现类：HashMap、LinkedHashMap、HashTable、TreeMap。

5. 接口Comparable<E>：

此接口强行对实现它的每个类的对象进行整体排序，此排序被称为类的自然排序，类的compareTo方法被称为它的自然比较方法。

1. ArrayList和LinkedList的区别

ArrayList和LinkedList都是List的实现类.

ArrayList：

1. 数组(顺序表的存储结构)。
2. 顺序表的存储地址是连续的,所以查找比较快。
3. 在插入删除的时候，由于需要把其他元素往后或者前移动，比较耗时。
4. 空间花费：要在list列表的结尾预留一定的容量空间
5. 适合索引的随机访问；尾插尾删

LinkedList：

1. 链表。
2. 存储地址不连续，每个存储地址通过指针指向，在查找的时候需要通过指针遍历元素，查找比较慢。
3. 插入删除不需要移动其他元素，比较快。
4. 空间花费：每个元素都需要消耗空间
5. 适合对指定位置进行插入删除

Vector：

1. 采用synchronized，线程安全，性能较低，读读互斥
2. set接口和map接口的关系

1. set实际上就是map，保存的单个元素存储在map的key

2. set不允许数据重复

3. hashSet判断两个对象是否重复，用equals和hashCode

4. 元素想要保存到TreeSet中，要么本身所在的类实现Comparable，要么通过外部传一个比较器

1. HashMap HashTable TreeMap的关系 区别

1. 底层实现：HashMap和HashTable基于哈希表实现，TreeMap基于红黑树实现

2. null值：HashMap的key和value都可以为空，key-1 value-N；

HashTable都不能为空；

TreeMap只允许value为空

3. 线程安全：

HashMap和TreeMap异步处理，不安全，性能高；

HashTable用synchronized，安全，性能低，读读互斥；

1. Java多线程的四种实现方法
2. 继承Thread类
3. 实现Runnable接口：run方法没有返回值
4. 实现Callable接口：如果线程结束后需要返回一些结果，就用从方法
5. 使用线程池
6. 如何启动线程

线程只有一种启动的方法，就是调用start方法

Run方法仅仅是启动一个类

1. 线程停止：

设置标记位（正常退出）；

Stop（强行退出）；

Interrupt（中断传输）。

1. 多线程常用操作方法
2. 线程的同步处理
3. Synchronized关键字处理：同步代码块，同步方法

使用同步代码块：必须要设置一个要锁定的对象，一般是syn（this）

使用方法：

1. synchronized底层实现: 对象Monitor机制

任意object及其子类对象内部在JVM中都附加Monitor, 获取一个对象的锁, 实际上就是获取改对象Monitor。同一时刻只有一个线程可以获取到monitor

当一个线程尝试获取对象Monitor时

1. 若此时Monitor值为0, 该对象未被任何线程获取, 当前线程获取Monitor, 将持有线程置为当前线程, Monitor值+1

2. 若此时Monitor值不为0, 此时该Monitor已被线程持有

a. 若当前线程恰好时持有线程, Monitor值再次+1, 当前线程继续进入同步块(锁的可重入性)

b. 若持有线程不是当前线程, 当前线程进入同步队列, 等待Monitor值减为0

加锁: monitorenter

解锁: monitorexit

任意时刻只有当Monitor值为0的时候表示无锁状态

1. Syn的优化（锁：重量级锁，轻量级锁，偏向锁）
2. 重量级锁：会阻塞，唤醒请求加锁的线程， 针对的是多个线程同时竞争同一把锁的情况。JVM采用了自适应自旋，来避免线程在面对非常小的syn代码块时，仍会呗阻塞，唤醒的情况。
3. 轻量级锁：采用CAS操作，将锁对象的标记字段替换为一个指针，指向当前线程栈上的一块空间，存储着锁对象原本的标记字段， 它针对的时多个线程在不同时段申请同一把锁的情况。
4. 偏向锁：只在第一次请求的时候采用CAS操作，在锁对象的标记字段中记录下当前线程的地址，在之后的运行过程中，持有该偏向锁的线程的加锁操作将直接返回。 针对的是锁仅会被同一线程持有的情况。
5. 其他优化（锁粗化、锁消除）

锁粗化：将多次连续的加减锁过程合并在一起，减少无用的加减锁过程，提高效率。

锁消除：删除不必要的加锁操作。当变量为线程私有变量的时候，将原方法的syn消除掉。，

1. 死锁：线程彼此等待

产生死锁的四个条件：互斥；占有且等待；不可抢占；循环等待（同时满足）

解决：syn无法解决，所以产生了lock体系

1. Lock
2. Lock是一个接口，接口不可以直接new，要用子类，所以是Lock lock = new ReetrantLock()；
3. 常用方法：

Lock：加锁

Unlock：解锁

1. 生产者消费者模型：

wait()方法：使线程停止运行。

1. 只能在同步方法或同步块中调用，如果没有合适的锁，会抛异常。
2. wait()方法执行后，当前线程释放锁，线程与其他线程竞争重新获取锁。

notify()：使停止的线程继续运行

1. 也要在同步方法或者同步代码块中调用。
2. 用来通知那些可能等待该对象的对象锁的其他线程，对他们发出通知notify，并使它们重新获取该对象的对象锁。如果有多个线程等待，则由线程规划器随机挑选出一个呈wait状态的线程。
3. 使用notify后，当前线程不会马上释放该对象锁，要等到执行notify的线程将程序执行完才会释放对象锁。

notifyAll()：

当有多个线程在等待的时候，调用notifyAll可以一次唤醒所有的等待线程。

1. 异常
2. 所有的异常都是Throwable继承而来，有两个子类Error和Exception
3. Error描述了Java运行时内部错误和资源耗尽错误
4. Exception下分为RuntimeException和IOException。程序错误导致的异常属于RuntimeException，如果程序本身没有问题，但由于像i/o错误这类问题导致的异常属于IOException。
5. RuntimeException定义的异常可以由用户选择性的进行异常处理
6. 常见的RuntimeException有：ClassCastExcptioon和NullPointExcption
7. 进程和线程
8. 进程: 任务管理器中的内容. 每个运行中的程序就是一个进程.

1)独立性: 每个进程都拥有自己私人的地址空间. 在没有经过进程本身运行的情况下是不能访问其中的内容的

2)动态性: 进程和程序的区别就是, 程序是静态的, 而进程是动态的. 有时间概念, 比如生命周期

3)并发行: 进程之间, 交替着执行

2. 线程: 一个顺序执行流. 一个进程可以有多个线程

1)进程之间不可以共享内存, 单线程之间共享内存非常容易

2)系统创建进程需要为该进程重新分配系统资源, 但是创建线程的代价很小. 所以多线程实现多任务并发比多进程实现并发的效率高

3) Java语言内置多线程功能支持, 而不是单纯的作为底层操作系统的调度方式

区别:

1. 地址空间和其他资源: 进程间相互独立, 同一进程的各线程间共享. 某进程内的线程在其他进程不可见
2. 通信: 进程间通信, 线程间可以直接读写进程数据段来进行通信, 需要进程同步和互斥手段的辅助, 以保证数据的一致性
3. 调度和切换: 线程上下文切换比进程上下文切换快得多