Java基础面试题

1. HashMap与HashTable的实现原理，线程安全性，hash冲突及处理算法?

解析：在1.8之前，HashMap是有数组加链表的形式实现的，在1.8及之后，HashMap由数组加链表和红黑树(当链表的长度超过8之后转化为红黑树)实，不是线程安全的；HashTable由数组加链表的形式实现，是线程相对安全的，以当前实例为锁，如果有两个线程，一个执行加入，一个执行get操作，则会阻塞，同时也会存在数据不一致性的问题，如果在外部没加锁，可能出现异常。hash冲突解决：hash值的计算是以key的hashCode为基准经过变换计算出的hash值，如果有两个key得到了相同的hash值，则会在数组中存放在相同的索引位置上，多个值则以链表或者红黑树的形式存在，如果两个key是equals或者是同一个对象会替换掉老的值。

2. ConcurrentHashMap

解析：1.8中是通过CAS实现同步的，1.7是通过锁分段的技术实现

3. 进程和线程的区别

解析：

1. 进程是操作系统分配资源的最小单位，线程是操作系统调度执行的最小单位；
2. 进程与进程之间的资源是相互独立，互不能访问的，必须通过进程间通信才能相互访问；线程是存在于进程中的一个执行线，线程之间共享同一个进程资源，包括内存、I/O、cpu等；
3. 进程分配资源大，切换时效率低；线程切换效率高

4. java的线程模型

解析：

首先，线程的实现有3种方式：使用操作系统内核线程实现、使用用户线程实现和使用用户线程加轻量级进程一起实现。

操作系统内核线程是指直接由操作系统内核支持的线程，这种线程由内核来完成线程切换，内核通过控制调度器对线程进行调度，并负责将线程的任务映射到各个处理器上。程序一般不直接使用内核线程，而是通过由系统提供的一种内核线程的高级接口--轻量级进程，就是我们通常意义上所讲的线程，由于每个轻量级进程都由一个内核线程支持，因此只有先支持内核线程才能有轻量级进程，这种1：1的关系称为一对一的线程模型；局限性：由于是基于内核线程实现的，所以各种线程操作，如创建、析构及同步，都需要进行系统调用，而系统调用的代价相对较高，需要在用户态和内核态来回切换；其次每个轻量级进程都需要一个内核线程支持，因此轻量级进程要消耗一定的内核资源。

用户线程：是指完全建立在用户空间的线程，系统内核不会感知到线程存在。用户线程的建立、同步、销毁和调度完全在用户空间中完成，不需要内核的帮助。局限性：由于没有系统内核的支援，所有的线程操作都需要由用户程序自己处理，线程的创建、切换和调度，以及阻塞如何处理、多处理系统如何映射到其他处理器上等这些异常困难的问题也要用户自己实现，所以用户线程的实现一般都是非常复杂的。这种模型也叫一对多线程模型。

用户线程加轻量级进程实现：用户线程还是完全建立在用户空间中，比如线程创建、切换、析构等操作依然在用户空间里面进行，这些操作依然廉价，并且可以支持大规模的用户线程并发；而轻量级进程的作用就是可以使用内核的线程调度和处理器映射的功能；这种模型也叫多对多模型。

java里面的线程模型是依赖于底层操作系统支持的，在window和linux上都是使用的一对一的线程模型实现的，一条java线程就映射到一条轻量级进程之中，因为window和linux提供的线程模型也是一对一的；在solaris平台中，由于提供了一对一和多对多的线程模型，因此solaris的jdk版本提供了对应的虚拟机参数让用户可以明确指定用哪种线程模型。

5. 线程池

创建线程池的优点：

1. 降低资源消耗：可以重复利用已经创建的线程降低线程创建和销毁造成的消耗；
2. 提高响应速度：当任务到达时，任务可以不需要等到线程创建就能立即执行；
3. 提高线程的可管理性：线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一分配、调优和监控。

Java中提供了Executors类让我们可以创建多种不同类型的线程池，包括有：

- newCachedThreadPool(): 返回的一个ThreadPoolExecutor实例，是一个corePoolSize为0，maxPoolSize为Max_Int，keepAliveTime为60s的线程池，工作队列为SynchronousQueue；
- newFixedThreadPool():返回的也是ThreadPoolExecutor实例，是一个corePoolSize=maxPoolSize的固定大小的线程池，keepAliveTime为0，工作对象为LinkedBlockingQueue；
- newScheduledThreadPool():返回的是ScheduledThreadPoolExecutor的实例，corePoolSize由参数决定，maxPoolSize为max_int，工作队列为DelayedWorkQueue
- newSingleThreadPool:返回的是ThreadPoolExecutor的实例，corePoolSize=maxPoolSize=1，工作队列为LinkedBlockingQueue；

Java默认提供的拒绝机制有：

- AbortPolicy：为Java线程池默认的阻塞策略，不执行此任务，而且直接抛出一个运行时异常，切记ThreadPoolExecutor.execute需要try catch，否则程序会直接退出；
- DiscardPolicy：直接抛弃，任务不执行，空方法
- DiscardOldestPolicy：从队列里面抛弃head的一个任务，并再次execute此task；
- CallerRunsPolicy：在调用execute的线程里面执行此command，会阻塞入口；
- 用户自定义拒绝策略（最常用），实现RejectedExecuionHandler接口

6. 数据一致性如何保证?

首先说一下为什么会出现数据不一致的原因？因为每个线程在工作时，都有自己独立的工作内存，当要访问一个数据时，必须经过这样两个操作：1. 从主内存执行读操作；2. 由工作内存执行load操作；同样保存一个数据时也必须经过两个操作：1. 工作内存执行store操作；2. 主内存执行write操作。所以可能当一个线程修改完一个数据之后，并不会马上反应到内存中，从而其他线程读取到的可能是不一样的数据。

保证：

锁

volatile修饰变量保证变量的修改对其他线程立即可见

final不可变变量

7. Java中实现多态的机制是什么?

多态是指通过接口或者抽象方法调用时，会根据具体的实现执行不一样的操作的这样一种现象。

java中实现多态的机制是继承+方法重写或者说实现。

8. 说说你对Java反射的理解?

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。通过对java字节码的控制来实现类实例的创建，属性访问和方法执行。java中获取字节码有如下方式：

- Class.forname(classname):通过全限定类名来访问java类的字节码；
- getClass():通过实例的getClass()方法可以访问对应类的字节码；
- .class:通过类的class属性可以访问类的字节码。

9. 同步的方法?

synchronized，volatile，lock（ReentrantLock），cas，ThreadLocal

10. Java中wait和sleep方法的区别?

1. wait是属于Object中的方法，sleep是属于Thread中的方法；
2. wait方法会释放当前线程占有的锁，而sleep方法不会；
3. wait，notify和notifyAll只能在同步控制方法或者同步控制代码块中使用，而sleep可以在任何地方使用；
4. wait方法通常用于线程间的交互，而sleep方法则用于暂停执行；

参考

11. Java中yield和join方法的区别?

1. 都是Thread里面的实例方法；
2. yield方法会让出当前线程占有的cpu调度时间，让同等优先级或者更高优先级的线程执行，如果没有这些线程，则当前线程会继续执行；
3. join方法表示在某一个线程的执行过程中等待另一个线程执行结束在继续执行当前线程，比如在main方法里面调用t.join方法，表示main线程会等待t线程执行结束才会继续执行t.join之后的代码。

12. synchronized和volatile关键字的作用?

1. 两个关键字都可以保证变量在多个线程间的可见性，即一个线程修改了某个变量的值，另一个线程能立即看到这个新值；
2. 禁止指令重排序；
3. synchronized通过加锁实现的原子性，当一个线程获取了锁之后，其他线程不能在获取这个锁直到线程执行完毕。
4. volatile本质是告诉jvm当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读取；synchronized则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞；
5. volatile仅能使用在变量级别；synchronized则可以使用在变量、方法和类级别的；
6. volatile仅能实现变量的修改可见性，并不能保证原子性；synchronized则可以保证变量的修改可见性和原子性；

7. volatile不会造成线程阻塞；synchronized则会造成线程阻塞；
8. volatile标记的变量不会被编译器优化，而synchronized可以被编译器优化。

13. ThreadLocal原理，实现及如何保证Local属性?

参考

14. 泛型的原理，举例说明?

在Java中，泛型并不是一种真正的泛型，泛型只会在源码中存在，被编译之后，类型会被擦除，字节码层面依然是通过类型转换实现的，泛型只不过是Java提供的一种语法糖。举例说明：在一个类中实现一个带泛型参数的重载方法，便会发现编译不通过，就知道编译代码被编译之后类型被擦除了，如下代码编译会失败：

```
public class GenericTypes {
    public static void method(List<String> list) {
        System.out.println("invoke method(List<String> list)");
    }

    public static void method(List<Integer> list) {
        System.out.println("invoke method(List<Integer> list)")
    }
}
```

15. java的异常体系?

顶级接口为Throwable，java中所有的异常都由它派生出来，其下分为Error和Exception两种类型的异常；其中Error表示错误，包括有虚拟机错误、系统错误等，并不属于我们程序代码的问题，我们不能捕获，捕获也无法处理；Exception表示程序异常，其下分为两个大类：运行时异常和检查异常，运行时异常表示异常是在运行过程中产生的逻辑问题，可以不捕获抛出，让程序员发现处理bug，检查异常是在编辑阶段就会产生的异常，必须要明确try catch捕获处理；

常见的运行时异常有：NullPointexception、IndexOutOfBoundException、FileNotFoundException、EOFException、NumberFormatException、ClassCastException、SQLException

检查异常有：IOException、ClassNotFoundException、Timeoutexception

16. 如何控制某个方法允许并发访问线程的个数?

使用信号量方法，在方法开始请求一个信号，方法结束时释放一个信号；如果信号被用完了则等待；这样可以通过信号量的个数来控制访问方法的线程个数；在Java中信号量通过Semaphore类实现。

17. 动态代理的区别，什么场景使用?

动态代理是Java里面最重要的一个技术，动态代理技术就是用来产生一个对象的代理对象的；有静态代理和动态代理两种方式，静态代理是在写代码编译时就知道被代理的类是哪个类了，而动态代理得等到运行是才能知道，在java中有两种动态代理：基于接口的动态代理和基于类的动态代理，其中基于接口的动态代理由JDK的提供的原生Proxy类实现，基于类的动态代理通过引入cglib实现。

18. 简述synchronized的Object Monitor机制?

synchronized关键字在经过编译之后，会在同步代码块的前后分别形成monitorenter和monitorexit两个指令，这两个字节码指令有一个refrence参数来指定要锁定和解锁的对象，在线程执行到monitorenter指令时，要尝试获取refrence参数的锁，如果这个对象没有被锁定，或者当前线程已经拥有了这个对象的锁，把锁的计数器加1，相应的执行到monitorexit指令时计数器减1，当计数器为0时，锁就被释放了。

19. 简述happen-before规则（先行发生原则）

先行发生是Java内存模型中定义的两项操作之间的偏序关系，如果说操作A先行发生于操作B，其实就是说在发生操作B之前，操作A产生的影响能被操作B观察到，“影响”包括修改了内存中共享变量的值、发送了消息、调用了方法等。Java内存模型下有一些天然的先行发生关系，这些先行发生关系无须任何同步器协助就已经存在，可以在编码中直接使用。如果两个操作之间的关系不在此列，并且无法从这些规则推导出来，他们就没有顺序性保障，虚拟机可以对他们随意得进行重排序，这些关系有：

- 程序次序规则：在一个线程内，按照程序代码顺序，书写在前面的操作先行发生于书写在后面的操作。准确的说，应该是控制流顺序而不是程序代码顺序；
- 管程锁定规则：一个unlock操作先行发生与后面对同一锁的lock操作。这里必须强调的是同一个锁，而后面是指时间上的先后顺序；
- volatile变量规则：对一个volatile变量的写操作先行发生于后面对这个变量的读操作，这里的后面同样是指时间上的先后顺序；
- 线程启动规则：Thread对象的start方法先行发生于此线程的每一个动作；
- 线程终止规则：线程中的所有操作都先行发生于对此线程的终止检测，我们可以通过Thread.join方法结束、Thread.isAlive方法的返回值等手段检测到线程已经终止执行了；
- 线程中断规则：对线程interrupt方法的调用先行发生于被中断线程的代码检测到中断事件的发生，可以通过Thread.interrupted方法检测到是有有中断发生；
- 对象终结规则：一个对象的初始化完成先行发生于它的finalize方法的开始；
- 传递性：如果操作A先行发生于操作B，操作B先行发生于操作C，那就可以得出操作A先行发生于操作C的结论。

20. JUC和Object的monitor机制区别是什么?

JUC指的是java.util.concurrent包里面的Lock，Object的monitor指的是synchronized；

1. JUC是jdk api层面的互斥锁实现，monitor机制是原生语法层面互斥锁的实现；
2. JUC提供了一些高级功能，比如：等待可中断、公平锁、多条件锁等
3. synchronized的锁是非公平的，ReentrantLock默认也是非公平锁，可以通过构造函数创建公平锁

21. 简述AQS的原理?

AQS是基于双向链表实现的FIFO等待队列，是用来实现同步器的同步框架，通俗点说是用来实现锁的工具；该队列用于存放需要等待的线程节点，AQS有个Node内部类用于保存等待线程的相关信息。

22. 简述DCL失效的原因，解决办法?

DCL意思是指双锁检测，主要用于在创建单例对象时使用这种技术。

失效原因：new操作并不是原子性的以及指令重排序引起的，new操作实际分为了三个步骤：1. 分配一块内存M；2. 在内存上初始化实例对象；3. 然后M的地址赋值给instance变量。这三个步骤经过指令重排序优化之后的执行路径却变成了这样：1. 分配一块内存M；2. 将M的地址赋值给instance变量；3. 最后在内存M上初始化实例对象；优化之后就可能导致线程获取到的instance是没有初始化过的，如果这个时候再访问instance的成员变量就可能触发空指针异常。

解决办法：将instance变量申明为volatile的，这样可以防止指令重排序

23. java中创建单例的方式?

参考

24. 简述nio原理?

25. jvm运行时数据区域有那几部组成，各自的作用?

- 方法区：与java堆一样，是各个线程共享的内存区域，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；别名：Non-Heap；
- 虚拟机栈：也是线程私有的，它的生命周期与线程相同；虚拟机栈描述的是Java方法执行的内存模型，每个方法在执行时都会创建一个栈帧用于存储局部变量表、操作数栈、动态链接、方法出口等信息。
- 本地方法栈：于虚拟机栈的作用相似，为虚拟机使用到的Native方法服务
- 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；此内存区域的目的是存放对象实例和数组；堆也是垃圾收集器管理的主要区域；可以细分为新生代和老年代，再细致一点的有Eden空间、From Survivor空间、To Survivor空间等。
- 程序计数器：一块较小的内存空间，它可以看作是当前线程所执行的字节码的行号指示器；因为是记录线程的字节码行号，所以每个线程都有一个独立的程序计数器，属于线程私有，独立存储，我们称这类内存区域为线程私有的内存；
- 运行时常量池：方法区的一部分，用于存放编译期生成的各种字面量和符号引用，这部分内容将在类加载后进入方法区的运行时常量池中存放。

25. GC算法有哪些？GC收集器有哪些?

GC算法：

1. 标记-清除算法：算法分为标记和清除两个阶段：首先标记出所需要回收的对象，在标记完成后统一回收所有被标记的对象。不足：一个效率，标记和清除两个过程的效率都不高；另一个是空间问题，标记清除之后会产生大量不连续的内存碎片，可能会导致分配大对象无法找到足够的连续内存而不得不提前触发另一次垃圾收集动作；
2. 复制算法：为了解决效率问题，复制算法将可用内存按容量大小分为大小相等的两块，每次只使用其中的一块，当一块内存用完了，就将还存活的对象复制到另外一块上面，然后再把已使用过的内存空间一次清理掉；缺点：对象存活率较高是要进行较多的复制操作；
3. 标记-整理算法：标记过程与标记清理算法一样，但后续步骤不是直接对可回收对象进行清理，而是让所有存活对象向一端移动，然后直接清理掉边界以外的内存；
4. 分代收集算法：当前商业虚拟机的垃圾收集都采用分代收集算法，根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，然后根据各个年代的特点采用最适当的收集算法；新生代采用复制算法，老年代采用标记清理或标记整理算法。

GC收集器：

1. Serial收集器：
2. ParNew收集器：
3. Parallel Scavenge收集器：
4. Serial Old收集器：
5. Parallel Old收集器：
6. CMS收集器：

7. G1收集器：

26. class加载各阶段过程?

- 加载：是类加载过程的一个阶段，主要完成以下三件事情：
    1. 通过一个类的全限定名来获取定义此类的二进制字节流；
    2. 将这个字节流所代表的静态存储结构转化为方法区的运行时数据结构；
    3. 在内存中生成一个代表这个类的java.lang.Class对象，作为方法区这个类的各种数据的访问入口。
- 验证：是连接阶段的第一步，这一阶段的目的是为了确保Class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全；验证阶段主要完成如下4个阶段的检验动作：
    1. 文件格式验证：是否以魔数0xCAFEBABE开头，java的主、次版本号是否在当前虚拟机处理范围之内，常量池中的常量是否有不被支持的类型等；
    2. 元数据验证：这个类是否有父类，这个类的父类是否继承了不允许被继承的类，是否实现了父类或接口之中要求实现的所有方法，字段、方法是否与父类产生了矛盾等；
    3. 字节码验证：确定程序语义是合法的、符合逻辑的，比如：任意时刻操作数栈的数据类型与指令代码序列都能配合工作，保证跳转指令不会跳转到方法体以外的字节码指令上；
    4. 符号引用验证：
- 准备：正式为类变量分配内存并设置类变量初始值（默认值，比如int是0之类的）的阶段，这些变量所使用的内存都将在方法区中进行分配；注意这个阶段进行内存分配的仅包括类变量，而不包括实例变量，实例变量将在对象实例化时随着对象一起分配在堆中；
- 解析：是虚拟机将常量池内的符号引用替换为直接引用的过程；
- 初始化：是类加载过程的最后一步，通过执行类构造器<clinit>()方法，对类变量和其他资源进行初始化（赋予程序员指定的值）

27. classLoader有哪些类型?

- BootstrapClassLoader（启动类加载器）：使用C++语言实现，是虚拟机自身的一部分；负责将放在JAVA_HOME/lib目录中的，或者被-Xbootclasspath参数所指定的路径中的，并且是虚拟机识别的（仅按照文件名识别，如rt.jar，名字不符合的类库即使放在lib目录中也不会被加载）类库加载到虚拟机内存中；
- ExtensionClassLoader（扩展类加载器）：由sum.misc.Launcher$ExtClassLoader实现，负责加载Java_home/lib/ext目录中的，或者被java.ext.dirs系统变量所指定的路径中的所有类库，开发者可以直接使用扩展类加载器；
- ApplicationClassLoader（应用程序类加载器）：由sum.misc.Launcher$AppClassLoader实现，是getSystemClassLoader()方法的返回值，所以也叫它系统类加载器；负责加载用户类路径（ClassPath）上所指定的类库，程序中默认的类加载器；

双亲委派模型：如果一个类加载器收到了类加载的请求，它首先不会自己去尝试加载这个类，而是把这个请求委派给父类加载器去完成，每一个层次的类加载器都是如此，因此所有的加载请求最终都应该传送到顶层的启动类加载器中，只有当父加载器反馈自己无法完成这个加载请求时，子加载器才会尝试自己去加载。

28. 常用的JDK命令行工具?

29. TCP的三次握手和四次挥手?

三次握手是发生在建立TCP连接时的动作，步骤如下：

1. 第一次：客户端发送一个带有SYN标志的数据包到服务端；
2. 第二次：服务端收到客户端发送的第一次握手信息，返回客户端带有SYN/ACK标志的数据包；
3. 第三次：客户端收到服务端返回的第二次握手信息后，发送带有ACK标志的数据包给服务端。

至此，三次握手结束，双方可以确保收发功能都正常，缺一不可。

四次挥手是发生在断开TCP连接时的动作，步骤如下：

1. 第一次：客户端发送一个带有FIN标志的数据包，告诉服务端要关闭客户端到服务端的数据传送；
2. 第二次：服务端收到这个FIN之后，返回给客户端一个ACK，表示确认收到请求；
3. 第三次：服务端发送一个FIN给客户端，关闭与客户端的连接；
4. 第四次：客户端发回ACK报文确认

参考

30. TCP与UDP的区别?

TCP特点：

1. 面向连接；
2. 可靠传输；
3. 以字节流的方式进行传输；
4. 传输效率慢，消耗资源多；
5. 首部字节20-60字节；
6. 适合要求通信数据可靠的场景，如：文件传输、邮件传输；

UDP特点：

1. 无连接；
2. 传输不可靠；
3. 以数据报文段的方式进行传输；
4. 传输效率快，所需资源少；
5. 首部自己只要8个字节；
6. 适合要求通信速度高的场景，比如：域名转换；

UDP 在传送数据之前不需要先建立连接，远地主机在收到 UDP 报文后，不需要给出任何确认。虽然 UDP 不提供可靠交付，但在某些情况下 UDP 确是一种最有效的工作方式（一般用于即时通信），比如： QQ 语音、 QQ 视频 、直播等等

TCP 提供面向连接的服务。在传送数据之前必须先建立连接，数据传送结束后要释放连接。 TCP 不提供广播或多播服务。由于 TCP 要提供可靠的，面向连接的运输服务（TCP的可靠体现在TCP在传递数据之前，会有三次握手来建立连接，而且在数据传递时，有确认、窗口、重传、拥塞控制机制，在数据传完后，还会断开连接用来节约系统资源），这一难以避免增加了许多开销，如确认，流量控制，计时器以及连接管理等。这不仅使协议数据单元的首部增大很多，还要占用许多处理机资源。TCP 一般用于文件传输、发送和接收邮件、远程登录等场景。

31. TCP协议如何保证可靠传输?

1. 应用数据被分割成TCP认为最适合发送的数据块；
2. TCP给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送给应用层；
3. 校验和：TCP将保持它首部和数据的校验和。这是一个端到端的校验和，目的是检测数据在传输过程中的任何变化。如果收到段的校验和有差错，TCP将丢弃这个报文段和不确认收到此报文段；
4. TCP的接收端会丢弃重复的数据；

5. 流量控制：TCP连接的每一方都有固定大小的缓冲空间，TCP的接收端只允许发送端发送接收端缓冲区能接纳的数据量大小。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP使用的流量控制协议是可变大小的滑动窗口协议。
6. 拥塞控制：当网络拥塞时，减少数据的发送；
7. 停止等待协议：也是为了实现可靠传输的，它的基本原理就是每发送完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组；
8. 超时重传：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

停止等待协议：

32. 在浏览器中输入url地址到显示页面的过程?

1. DNS解析；
2. TCP连接；
3. 发送HTTP请求；
4. 服务器处理请求并返回HTTP报文；
5. 浏览器解析渲染页面；
6. 连接结束。

33. HTTP的报文结构?

报文的组成部分：

1. 起始行：
    - 请求报文的起始行由：<方法> <请求资源地址> <http版本> 三个部分组成；
    - 响应报文的起始行由：<http版本> <响应状态码> <响应状态描述> 三个部分组成。
2. 首部块(请求/响应头)：每个首部都包含一个名字，后面跟着一个冒号，然后是一个可选的空格，接着是一个值，最后是一个CRLF。首部块是由一个空行CRLF结束的，表示首部块列表的结束和主体部分的开始；
3. 主体：包含一个由任意数据组成的数据块；并不是所有的报文都包含主体部分，有时，报文只是以一个CRLF结束。

34. HTTP中的请求方法有哪些？常用的有哪些?

1. GET：用于请求服务器发送某个资源；
2. HEAD：与GET方法的行为很类似，但服务器在响应中只返回首部，不会返回实体的主体部分；通过head请求客户端可以知道：
    - 在不获取资源的情况下了解资源的情况(比如判断其类型)；
    - 通过查看响应中的状态码，查看某个对象是否存在；
    - 通过查看首部，测试资源是否被修改。
3. PUT：向服务器写入文档，语义就是让服务器用请求的主体部分来创建一个由所请求的URL命名的新文档，或者是替换旧文档；
4. POST：起初是用来向服务器输入数据的，通常用它来支持HTML的表单；
5. TRACE：用于诊断；
6. OPTIONS：用于请求web服务器告知其支持的各种功能，可以询问服务器通常支持哪些方法，或者对某些特殊资源支持哪些方法；

7. DELETE：请求服务器删除请求URL所指定的资源。

35. HTTP与HTTPS的区别以及如何实现安全性?

区别：

1. HTTPS的服务器需要到CA申请证书，以证明自己服务器的用途；
2. HTTP信息是明文传输，HTTPS信息是密文传输；
3. HTTP与HTTPS的端口不同，一个是80端口，一个是443端口。

36. 死锁的四个必要条件和解决办法?

死锁：多个并发线程因争夺系统资源而产生相互等待的现象。

本质：系统资源有限；线程推进顺序不合理。

四个必要条件：

1. 互斥：某种资源一次只允许一个线程访问，即该资源一旦分配给某个进程，其他进程就不能再访问，直到该进程访问结束；
2. 占有且等待：一个进程本身占有资源（一种或多种），同时还有资源未得到满足，正在等待其他进程释放该资源；
3. 不可抢占：别人已经占有了某项资源，你不能因为自己也需要该资源，就去把别人的资源抢过来；
4. 循环等待：存在一个进程链，使得每个进程都占有下一个进程所需的至少一种资源。