HashMap 不是有序的，根据键得到值，具有很快的访问速度，遍历时，取得数据时完全随机的。不允许 key 重复，但是可以允许 value 重复，最多允许一条记录的 key 为 null，允许多条记录的值为 null。HashMap 不支持线程同步，是线程不安全的，Hashtable 是线程安全的，用 synchronized 关键字实现同步，效率低一些，不允许键值有 null。如果需要同步可以使用 Collections.syschronizedMap()方法实现同步，是内部锁的方式，只适合并发量小的情况。或者使用 ConcurrentHashMap。LinkedHashMap 和 TreeMap 是有序的，LinkedHashMap 按照读取顺序来排列，TreeMap 按照 key 大小升序排列。LinkedHashMap 内部除了 HashMap 结构，还维护了一个双向链表，用来保证元素顺序。TreeMap 内部有一个 Comparable 比较器，根据 key 值维护顺序，也可以自定义比较器，底层红黑树，复杂度 log(n)。红黑树通过红黑规则维护平衡。Hashtable 中 hash 数组的默认大小是 11，增加时为 old*2+1，HashMap 中的 hash 数组的默认大小是 16，增加时为 2 的指数，并发执行 put()时，多线程会导致 HashMap 的 Entry 形成环形结构，陷入死循环。

如果想要实现所有的线程一起等待某个时间发生，当某个事件发生时，所有线程一起执行，使用 CyclicBarrier。CyclicBarrier 底层由可重入锁 ReentrantLock 和 Condition 共同实现，还基于并发包的 AQS 同步器，ReentrantLock 是基于 AQS 实现的，AQS 的基础是 CAS(Compare And Swap)，AQS 基于 FIFO 队列实现。

垃圾回收机制，当一个对象没有引用指向的时候一个对象会被 GC，因为存在循环引用，所以根据根可达算法判断。

String 是不可变的，StringBuffer 和 StringBuilder 是可变的。后两者的字符内容可变，而前者创建后的内容不可变。String 不可变是因为 JDK 中类声明为 final 的。StringBuffer 是线程安全的，方法有 synchronized 关键字修饰，StringBuilder 是非线程安全的，方法没有 synchronized 修饰。线程安全会带来额外的系统开销，可能会发生阻塞现象，StringBuilder 的效率比 StringBuffer 高。如果对系统中的线程是否安全不知道，使用 StringBuffer，线程不安全地方加上 synchronized 关键字。

ArrayList 是以数组的形式存储在内存中，LinkedList 以链表的形式存储。ArrayList 适合查找，不适合指定位置的插入和删除操作，LinkedList 适合指定位置插入和删除操作，不适合查找。

Get 提交的数据最多 1024 字节，而 POST 没有限制。Get 提交的参数以及参数值会在地址栏显示，不安全，而 Post 不会在地址栏显示，是安全的。请求转发是一个请求，请求重定向是两个请求。

Session 应用服务器维护的一个服务器端的存储空间，Cookie 是客户端的存储空间，由浏览器维护。用户可以通过浏览器决定是否保存 Cookie，而不能决定是否保存 Session，因为 Session 是由服务器端维护的。Session 中保存的是对象，Cookie 中保存的是字符串。Session 和 Cookie 不能跨窗口使用，每次打开一个浏览器系统会赋予一个 SessionID，此时的

SessionID 不同，若要完成跨浏览器访问数据，可以使用 Application。Session、Cookie 都有失效时间，过期后会自动删除，减少系统开销。Servlet 的生命周期，init()负责初始化 Servlet 对象，service()负责响应客户端请求，destroy()是当 Servlet 对象退出的时候，负责释放占用资源。HTTP 报文包含内容：request line 是请求行，header line 是头部，blank line 是空白行，request body 是请求体。请求报文格式：<method> <request-url> <version> <headers><entity-body>。响应报文格式：<version> <status> <reason-phrase> <headers> <entity-body>。请求报文与响应报文只是起始行不同。常见状态码：200 请求成功 301 请求重定向 404 请求资源不存在 500 服务器错误。

PreparedStatement 支持动态设置参数，Statement 不支持。PreparedStatement 支持预编译，Statement 不支持，PreparedStatement 防止 SQL 注入，更加安全，Statement 不支持。SQL 注入就是通过 SQL 语句的拼接达到无参数查询数据库目的的方法。可以采用 PreparedStatement 避免 SQL 注入，在服务端接收参数数据后，进行验证，PreparedStatement 会自动检测，而 Statement 要手动检测。

Java 内存模型。一个对象、两个属性、四个方法实例化 100 次。由于 Java 中 new 出来的对象放在堆中，实例化 100 次，将在堆中产生 100 个对象，一般对象与其中的属性和方法都属于一个整体。如果属性和方法是静态的，用 static 关键字修饰，则属于类的属性和方法在内存中只有一份。

Java 内存模型规定的所有的变量都存储在主内存。每个线程还有自己的工作内存。线程的工作内存中保存了被该线程使用到的变量的主内存副本拷贝。线程对变量的所有操作都必须在工作内存中进行，而不是直接读写主内存中的变量。不同线程之间也无法直接访问对方工作内存中的变量，线程间的变量值传递需要通过主内存来完成。

ORM 模式时在单个组件中负责所有实体域对象的持久化，封装数据访问细节。Hibernate 是 ORM 的一个实现。Hibernate 提供了两级缓存。第一级缓存是 Session 的缓存。由于 Session 对象的生命周期通常对应一个数据库事务或者一个应用事务，因此它的缓存是事务范围的缓存。第一级缓存是必须的，不允许而且事实上无法被卸载。在第一级缓存中，持久化类的每个实例具有唯一的 ID。第二级缓存是一个可插拔的缓存插件，由 SessionFactory 负责管理，SessionFactory 对象的生命周期和应用程序的整个进程对应，因此第二级缓存是进程范围的缓存。缓存存放的对象的散装数据，第二级缓存是可选的，可以在每个类或者集合的粒度上配置二级缓存。

动态获取类的信息以及动态调用对象的方法的功能来自于 Java 的反射机制。反射机制提供了在运行的时候判断任意一个对象所属的类，在运行时构造任意一个类的对象，在运行的时候判断任意一个类所具有的成员变量和方法，在运行时调用任意一个一个对象的方法。在 JDK 中主要由 Class 类、Field 类、Method 类、Constructor 类、Array 类来实现 Java 的反射机制。

Spring 是处于 MVC 控制层的开源框架，具有面向切面编程的优势，提升了系统性能，通过控制反转，系统中用到的对象不是在系统加载时就全部实例化，而是在调用到这个类时才会实例化。其降低了组件之间的耦合性，实现了软件层次间的解耦。可以使用容易提供的众多服务，事务管理、消息服务、日志记录等。提供了 AOP 技术，很容易实现权限拦截，运行期监控等功能。AOP 中的是设计模式中的动态代理模式，提供 JDK 的动态代理接口 InvocationHandler，所有被代理对象的方法由 InvocationHandler 接管实际的处理任务。面向切面编程有切入点、切面、通知、织入概念。IOC 利用 Java 的反射机制实现，依赖注入即组件间的依赖关系由容器在运行期决定。依赖注入有构造函数注入和 set 方法注入两种。

Linux 下查看、监控 JVM。看总内存用 top，详细内存信息用 jmap，jstack 是 JDK 自带的工具，通过该工具可以查看 JVM 线程的运行状况，包括锁等待，线程是否在运行。

线程池的关键点是尽量减少线程切换和管理的开支，最大化利用 CPU。线程越少，线程切换和管理的开支越少。线程越多，CPU 的资源最大化利用。对于任务耗时短的情况，线程尽量少，如果线程太多有可能出现线程切换和管理的时间大于任务执行的时间，则降低了效率。对于任务耗时长的情况，如果是 CPU 类型的任务，线程不能太多，如果是 IO 类型的任务，线程多一些可以更充分利用 CPU。高并发低耗时，线程少一些，低并发高耗时，线程多一些。高并发高耗时，分析任务类型，增加排队，加大线程数。

Spring 的配置文件 bean 元素，scope 属性。Singleton，单例，表示只有一个对象，服务所有。Prototype 表示每次从容器中取出 bean，都会生成一个新实例，相当于 new 对象。Request 属性基于 web，表示每次接收一个请求生成一个新实例，Session 表示在每个 session 中该对象只有一个。GlobalSession！

线程的状态模型。（1）新建（new），new Thread。（2）就绪（Runnable），调用 Thread.start()方法，不一定马上执行，等待 CPU 时间。（3）运行（Running），获取 CPU 时间。（4）阻塞（Blocked），sleep 方法，等待锁。（5）死亡（Dead），正常结束和异常结束，Thread.stop()多线程下可能会造成数据不一致的问题。

原子性是指一个操作是不可中断的，即使在多个线程一起操作时，一旦一个操作开始，就不会被其他线程干扰。有序性是指程序实际执行的顺序并不一定和代码编写顺序相同，指令重排序现象。可见性是指当一个线程修改了一个共享变量的值，其他线程能够立即知道这个修改。

Happen-Before 规则，先行发生规则，保证多线程语义的一致性。程序顺序规则，一个线程内保证语义的串行性 a=1;b=a+1。Volatile 规则，volatile 变量的写优先发生于读，保证 volatile 变量的可见性。锁规则，解锁操作必然发生在随后的加锁之前。传递性，A 先于 B，B 先于 C，则 A 先于 C。线程的 start()方法先于其他所有操作。

CAS 操作是一条 CPU 指令，是原子操作，不会挂起，重试，无锁。包含三个参数。CAS(V,E,N)，V 表示要更新的变量，E 表示预期值，N 表示新值。仅当 V 值等于 E 值时，

才会将 E 值设置为 N，如果 V 值和 E 值不同，则说明已经有其他变量对其做了更新，则当前线程什么也不做。CAS 返回当前线程 V 的真实值。当多个线程同时使用 CAS 操作一个变量时，只会有一个胜出，并且成功更新，其余都失败。失败的线程不会挂起，只是通知失败，并且允许再次尝试，也允许失败的线程放弃操作。CAS 操作即使没有锁，也可以发现其他线程对当前线程的干扰，并进行处理。无锁的方式比阻塞的性能更好，线程不会挂起，只是不断的重试。

AtomicInteger 内部 private volatile int value 维护一个主要的变量值，使用 volatile 关键字保持可见性。还有 CAS 操作，并且死循环不断重试直到成功。

ReentrantLock 特点：（1）可重入，对于同一个线程，必须是可重入的，否则一个线程会把自己锁死，在内部维护一个可重入次数变量，如果不为 0，允许再次获得锁，如果为 0，说明释放了锁。（2）可中断，在加锁的同时可以去响应中断，如果发生了死锁，避免线程永远的等待，使用 lock.lockInterruptibly()。（3）可限时，tryLock()，在有限的时间内申请锁，申请不到就做其他事情。（4）公平锁和非公平锁，内部有两种实现，先来先得还是可以插队，公平锁的效率差。（5）一定要在 finally 手动释放。

ReentrantLock 内部实现：（1）CAS 操作，根据变量是否修改成功反推是否拿到了锁。（2）等待队列，如果没有拿到锁，线程进入等待队列。（3）park()，进入等待队列的线程通过 park()挂起。

Semaphore 是共享锁，锁只允许一个线程进入临界区，允许若干个线程进入临界区。ReadWriteLock，读不会修改数据，写需要修改数据，使用读写锁可以提高效率。读读之间不互斥，读写互斥，写写互斥。ConntDownLatch 只能使用一次，等待的是事件，基于 AQS 构建，内部维护一个变量 count。CyclicBarrier 是所有线程相互等待，可以重复使用，等待的是线程，维护一个 count 变量，是线程数，和一个 generation 变量，用来重复使用。

ConcurrentHashMap 是把一个大的 HashMap 分为若干个小的 HashMap，也就是 segment，每一个小的 segment 有一把锁，如果分为 16 个小的 HashMap 后，可以 16 个线程同时访问。其 key 和 value 都不能为空，rehash()是超过了 threshold，再次 hash，把空间翻倍，比较耗时。

BlockingQuene 是阻塞队列，线程安全的，效率不是很高，多个线程共享数据的容器，ArrayBlockingQuene 内部用数组实现，LinkedBlockingQueue 用链表实现，内部有 ReentrantLock 作为线程安全的作用，Condition 作为条件通知的作用，可以用来实现生产者消费者模型。高性能队列可用 ConcurrentLinkedQueue 实现。

线程池，线程的闯将和销毁代价比较高，如果频繁的创建和销毁线程，占据的 CPU 资源比较多。给业务逻辑的 CPU 资源会比较少，线程池实现对线程的复用。单个任务处理时间短，处理任务量大的时候使用线程池。负载过量后通过拒绝策略处理，workQueue 用来保存等待被执行的任务的阻塞队列，任务必须实现 Runnable 接口。

ForkJoin 把大任务分为小任务，并且将子结果整合为最终的结果。Future 模式的核心思想是异步调用，构造数据要花时间比较多，没有得到数据立即返回，可以在将来的时候得到数据，一段耗时的程序不需要立即得到返回结果。

死锁产生的四个条件，互斥，占有且等待，非抢占，循环等待。避免死锁的算法，银行家算法。

传统 IO 为每一个客户端使用一个线程，如果客户端出现延迟等异常，线程可能会被占用很长时间，因为数据的准备和读取都在这个线程中，如果客户端数量多，将会消耗大量的系统资源。NIO 是非阻塞的 IO，数据准备好了再工作。一个线程对应一个 Selector，一个 Selector 对应多个客户端。少量线程监控大量客户端，即使阻塞，数量也比较少。

锁优化：（1）减少锁的持有时间，不需要同步的代码不用放在同步的区间。（2）减少锁的粒度，分段锁，即 ConcurrentHashMap。（3）锁分离，ReadWriteLock 读写锁，LinkedBlockingQueue 的 take 和 put 在两端进行。（4）锁粗化，做其他不需要的同步工作，但是能够很快执行完毕，则整合成一次锁请求。（5）锁消除，编译器级别的优化。在即时编译时，如果发现不能被共享的对象，即编译器任务变量不会逃逸出去，则消除这些对象的锁操作。

虚拟机内部对锁的优化：（1）偏向锁，锁会偏向当前已经占有锁的线程，将对象头 mark 标记为偏向，当有其他线程请求锁时，偏向模式结束，在竞争激烈的场合会加重系统负担，因为不停的中断。（2）轻量级锁，通过 CAS 操作请求锁，如果成功，获得锁，失败则升级为重量级锁。竞争激烈场合会加重系统负担。（3）自旋锁，当竞争存在时，如果线程可以很快获得锁，不在 OS 层挂起线程，让线程做几个空操作。虚拟机内置锁的获取顺序：偏向锁—轻量级锁—自旋锁。

ThreadLocal 是完全可以替换掉锁的策略。当使用 ThreadLocal 维护变量时，会为每个使用该变量的线程提供独立的变量副本，所以每一个线程都可以独立的改变自己的副本，而不会影响其他线程所对应的副本。在 ThreadLocal 类中有一个 Map，用于存储每一个线程的变量副本，Map 中元素的键为线程对象，值为对应线程的变量副本。

哈希表是一种数据结构，可以提供快速的插入和查找操作，插入和删除只需要接近常量的时间，O(1)时间。缺点是基于数组的，数组的创建不好扩展，某些哈希表被填满后性能下降十分严重，没有一种简便的方法可以以任何一种顺序遍历表中的数据项。

解决哈希冲突的方法：（1）线性探测；（2）再哈希法，stepSize=constant*(key%constant)；（3）链地址法（HashMap 中使用的方法）

解决循环队列的队空和队满的方法：设置一个 flag 变量，初始值为 0，每次入队操作成功后 flag=1，出队操作成功后 flag=0，队空的条件为 front=rear&&flag=0，队满的条件时 front=rear&&flag=1。

堆是完全二叉树，除了最后一层节点不是满的，其他的每一层从左到右都完全是满的，

常常使用一个数组实现，每个父节点都大于子节点。堆可以用来实现优先级队列。用数组表示一个堆，数组中节点的索引为 X，则父节点的下标为(X-1)/2，左子节点下标为 2*x+1，右子节点下标为 2*X+2。

树可以实现在数组中查找数据和有序数组一样快，插入和删除和链表一样快。二叉搜索树，一个节点的左子节点的关键字小于这个节点，右子节点的关键字大于或者等于这个父节点。