# 需要掌握的知识点

## 对于项目

1. 明确你的项目到底是做什么的，有哪些功能
2. 明确你的项目的整体架构，在面试的时候能够清楚地画给面试官看并且清楚地指出从哪里调用到哪里、使用什么方式调用
3. 明确你的模块在整个项目中所处的位置及作用
4. 明确你的模块用到了哪些技术，更好一些的可以再了解一下整个项目用到了哪些技术

## 专业技能

1. 基本语法：这包括 static、final、transient 等关键字的作用，foreach 循环的原理（语法糖，其实还是使用了 Iterator 遍历）等等。今天面试我问你 static 关键字有哪些作用，如果你答出 static 修饰变量、修饰方法我会认为你合格，答出静态块，我会认为你不错，答出静态内部类我会认为你很好，答出静态导包我会对你很满意，因为能看出你非常热衷研究技术。
2. 集合：基本上就是 List、Map、Set，问的是各种实现类的底层实现原理，实现类的优缺点。集合要掌握的是 ArrayList、LinkedList、Hashtable、HashMap、ConcurrentHashMap、HashSet 的实现原理，能流利作答，当然能掌握 CopyOnWrite 容器和 Queue 是再好不过的了。另外多说一句，ConcurrentHashMap 的问题在面试中问得特别多，大概是因为这个类可以衍生出非常多的问题，关于 ConcurrentHashMap，我给网友朋友们提供三点回答或者是研究方向：（1）ConcurrentHashMap 的锁分段技术（2）ConcurrentHashMap 的读是否要加锁，为什么（get 操作不需要锁。第一步是访问 count 变量，这是一个 volatile 变量，由于所有的修改操作在进行结构修改时都会在最后一步写 count 变量，通过这种机制保证 get 操作能够得到几乎最新的结构更新。对于非结构更新，也就是结点值的改变，由于 HashEntry 的 value 变量是 volatile 的，也能保证读取到最新的值。）（3）ConcurrentHashMap 的迭代器是强一致性的迭代器还是弱一致性的迭代器（弱一致性）。
3. 设计模式：面试中关于设计模式的问答主要是三个方向：（1）你的项目中用到了哪些设计模式，如何使用（2）知道常用设计模式的优缺点（3）能画出常用设计模式的 UML 图
4. 多线程：线程池也是比较常问的一块，常用的线程池有几种？这几种线程池之间有什么区别和联系？线程池的实现原理是怎么样的？实际一些的，会给你一些具体的场景，让你回答这种场景该使用什么样的线程池比较合。多线程同步、锁这块也是重点。synchronized 和 ReentrantLock 的区别、synchronized 锁普通方法和锁静态方法、死锁的原理及排查方法等等。
5. JDK 源码：比较重要的源码：（1）List、Map、Set 实现类的源代码（2）ReentrantLock、AQS 的源代码（3）AtomicInteger 的实现原理，主要能说清楚 CAS 机制并且 AtomicInteger 是如何利用 CAS 机制实现的（4）线程池的实现原理（5）Object 类中的方法以及每个方法的作用
6. 框架：一些基本的应用。

7. 数据库：一些基本的像 union 和 union all 的区别、left join、几种索引及其区别就不谈了，比较重要的就是数据库性能的优化。
8. 数据结构与算法：数组、链表是基础，栈和队列深入一些但也不难，树挺重要的，比较重要的树 AVL 树、红黑树，可以不了解它们的具体实现，但是要知道什么是二叉查找树、什么是平衡树，AVL 树和红黑树的区别。
9. Java 虚拟机：Java 虚拟机中比较重要的内容：（1）Java 虚拟机的内存布局（2）GC 算法及几种垃圾收集器（3）类加载机制，也就是双亲委派模型（4）Java 内存模型（5）happens-before 规则（6）volatile 关键字使用规则
10. Web 方面的一些问题：谈谈分布式 Session 的几种实现方式；讲一下 Session 和 Cookie 的区别和联系以及 Session 的实现原理。web.xml 里面的内容是重点，Filter、Servlet、Listener，不说对它们的实现原理一清二楚吧，至少能对它们的使用知根知底。另外，一些细节的方面比如 get/post 的区别、forward/重定向的区别、HTTPS 的实现原理也都可能会被考察到。

# Java 基础

1. 如何实现在 main()方法执行之前输出东西？ 使用静态代码块。静态代码块在类加载时就会调用。
2. Java 程序初始化的顺序？父类静态变量--父类静态代码块--子类静态变量--子类静态代码块--父类非静态变量--父类非静态代码--父类构造函数--子类非静态变量--子类非静态代码--子类构造函数。
3. 普通方法可以与构造函数方法名一样
4. 为什么 Java 中有些接口没有任何方法？这些叫做标识接口，仅仅充当标识的作用，类似于汽车的标签。比如 cloneable 接口以及 serializable 接口。
5. Java 中 clone 方法的作用？返回一个对象的复制。这个复制对象是一个新的对象而不是原来对象的引用。如果用“=”得到的是这个对象的引用。只有基本数据类型是值传递。
6. 使用 clone（）方法？首先需要继承 cloneable 接口，在 clone 方法中调用 super.clone()，然后在 clone 方法中把浅复制的引用指向原型对象新的克隆体。
7. 浅复制和深复制的区别？浅复制只考虑所复制的对象，不复制它所引用的对象。
8. 反射机制？反射机制能够实现在运行时对类进行装载。反射能使得代码编写更加灵活。常用于一些框架的开发。内省 API（通过反射的方式操作 JavaBean 的属性）。
9. Java 创建对象的方式？new、反射、clone()、反序列化。
10. 面向对象的三大特征？继承、多态、封装。（继承破坏封装性，怎么看？）
11. 组合和继承的不同？继承是 is a 的关系，而组合是 has a 的关系。能用组合就不用继承。
12. 多态的实现机制：方法的重载（编译时多态性），方法的覆盖（运行时多态）。
13. 重载和覆盖的区别？覆盖是子类与父类之间的关系，重载是一个类中方法之间的关系。覆盖要求参数列表一样，重载要求参数列表不一样。
14. 抽象类与接口？
15. 内部类？静态内部类（static，不依赖于外部类就可以实例化）、成员内部类

（不可以定义静态的属性和方法）、局部内部类（不能被 public、private、protected、static 修饰，只能访问 final 类型的局部变量）、匿名内部类（没有名称的内部类，没有构造函数，必须继承其他类或者其他接口，不能定义静态成员以及方法，只能创建一个实例）。

16. 如何获取父类的类名？不能用 super.getClass().getName()，要使用 this.getClass().getSuperclass().getName()。（getClass 是 object 的 final 方法，不能被覆盖，返回的是此 object 运行时的类。）
17. This 与 super 的区别？
18. Static 关键字的作用？为某个特定的数据类型或对象分配单一的存储空间；实现某个方法或者属性与类关联而不是与对象关联。
19. Static 实现单例？
20. Static 与 final 结合使用表示该方法不可覆盖而且可以通过类名直接访问。
21. Volatite 是设计被用来修饰不同线程访问和修改的是同一个变量。
22. Strictfp 是精确浮点，用来确保浮点运算的准确性。
23. 什么是不可变类？不可变类是指当创建这个类的实例之后，就不允许修改他的值了。所有基本类型的包装类（Integer、Float 等，string）。创建不可变类要遵循的原则（类中的成员变量都被 private 修饰，类中没有修改成员变量的方法，确保所有方法不会被子类覆盖）
24. 不可变类的优缺点？优点：使用简单，线程安全，节省内存。缺点：会因为值的不同而产生新的对象。
25. 在 Java 中，原始数据类型在参数传递时是按值传递，而包装类型是按引用传递的。
26. Math 类中的 round，ceil，floor？
27. Char 类型变量是否可以用来存储一个中文汉字？可以
28. 判断是否存在中文字符？str.length()==str.getBytes().length
29. 实现国际化的应用常见的手段是？利用 ResourceBundle 类。
30. New String(“abc”)创建了几个对象？一个或者两个。
31. “==”、equals、hashCode 有什么不同？“==”是比较两个变量的值是否相等。在内存中所存储的值是否相等。即引用类型的引用和基本数据类型，比较的是引用。Equals 如果没有被重写那么也是比较引用，和“==”一样。但是像 String 被覆盖之后就是比较字符串的值了。hashCode 是鉴定两个对象是否相等。Equals 相等则 hashCode 也必须相等，equals 不相等，hashCode 可以不相等也可以相等。hashCode 不相等则 equals 也不相等，hashCode 相等，则 equals 可以相等也可以不相等。
32. 当一个字符串需要经常被修改时，使用 stringBuffer 比使用 string 要好很多。StringBuffer 是线程安全的，StringBuider 不是线程安全的。执行效率方面，StringBuilder 最高，StringBuffer 次之，string 最低。
33. 字符串 String 修改实现原理？
34. Java 中数组是不是对象？是对象，每个数组都有其对应的类型，可以通过 instanceof 来判断。（a instanceof int[]）
35. Length 属性与 length()方法？length 是数组的属性用于获取数组的长度，而 length()是字符串的方法用于计算字符串的长度。
36. Finally 块中的代码什么时候执行？finally 块里的代码是在 return 之前执行的，如果有 return 语句会覆盖别处的 return 语句。对于基本数据类型，在 finally

块中改变 return 的值对返回值没有任何影响，而对引用类型的数据会有影响。

37. Finally 代码块也是不一定被执行的。在进入 try 语句之前已经报了异常；程序执行时遇到强制退出。
38. 运行时异常与普通异常？异常类（error 和 exception）；exception 又包括检查异常（IO 异常、SQL 异常）和运行时异常（空指针异常、类型转换异常、数组越界异常、数组存储异常、算术异常、缓冲区溢出异常）。
39. 异常处理时需要注意的地方？先捕获子类异常，然后在捕获基类异常；尽早抛出异常并进行处理；自定义异常；异常能处理就处理
40. JVM 类加载机制？隐式加载和显示加载两种。Bootstrap loader 加载系统类——ExtClassLoader 加载扩展类——AppClassLoader 加载应用类。
41. 类加载的主要步骤：装载——链接（检查；准备；解析）——初始化
42. 什么是垃圾回收（GC）？回收程序中不在使用的内存。垃圾回收器使用有向图来进行记录和管理内存中的所有对象，识别对象是否是“可达的”，有引用变量引用的就是可达对象。不可达的对象都会被垃圾回收器回收。
43. 垃圾回收算法？引用计数算法（引用计数器，被引用就加 1，释放引用时减 1。无法解决相互引用的问题）；追踪回收算法（利用 JVM 的对象有向图，标记遍历的对象，回收没有被标记的对象）；压缩回收算法（把堆中活动的对象移动到堆的一端，留出一大块空闲区域，相当于进行了碎片处理）；复制回收算法（将堆分成两个一样的区域，只有其中的一个区域被使用，直到这个区域被消耗完，程序会将所有活动的对象复制到另外一个区域，如此循环）；按代回收算法（把堆分成很多个子堆，每个子堆就视为一代。优先搜集年幼的对象，如果多次搜集任然存活，就把该对象转移到高一级堆中，减少对其的扫描）
44. Java 是否存在内存泄露（OOM，OutOfMemory）问题？存在。内存泄露的两种情况：堆中申请的空间没有被释放；对象已经不再被使用但还是在内存中保留。
45. Java 中引起内存泄露的原因，举几个例子？静态集合类（Vector,HashMap）；各种连接（数据库连接，IO 连接）；监听器；变量不合理的作用域（变量生命周期与程序的生命周期一致）；单例模式可能造成 OOM。
46. Java 中堆和栈的区别？栈内存主要用于存放基本数据类型以及引用变量，堆内存用来存放运行时创建的对象（一般来说 new 处理的对象）。常量池是存放字符串常量以及基本数据类型常量。线程之间共享堆内存，所有多线程要对数据进行同步。从功能上看，堆是主要用来存放对象的，栈主要用来执行程序的。
47. 容器数据结构：列表，队列，集合，栈，映射表（list,queue,set,stack,map）
48. 当一个集合被作为参数传递给一个函数时，如何才可以确保函数不能修改它？在作为参数传递之前，我们可以使用 Collections.unmodifiableCollection(Collection c)方法创建一个只读集合，这将确保改变集合的任何操作都会抛出 UnsupportedOperationException。
49. .大写的 O 是什么？举几个例子？
    大写的 O 描述的是，就数据结构中的一系列元素而言，一个算法的性能。Collection 类就是实际的数据结构，我们通常基于时间、内存和性能，使用大写的 O 来选择集合实现。比如：例子 1：ArrayList 的 get(index i)是一个常量时间操作，它不依赖 list 中元素的数量。所以它的性能是 O(1)。例子 2：一个

对于数组或列表的线性搜索的性能是 O(n)，因为我们需要遍历所有的元素来查找需要的元素。

50. Hashmap 是基于散列表实现的，而 Treemap 是基于红黑树实现的。
51. Iterator 与 ListIterator 的区别？Iterator 只能正向遍历集合，适合于获取移除元素，ListIterator 可以支持元素的修改。
52. 线程安全的容器？vector、hashtable
53. Linklist 用于随机访问的效率较低，主要用于对数据指定位置的插入或是删除操作。主要操作为索引时应该使用 vector 或 arrylist。
54. Hashmap 允许空键值，但最多允许一条记录的键值为空。而 hashtable 不允许。Hashtable 使用的是 Enumeration，hashmap 使用的是 Iterator。
55. Hashtable 的同步指的是什么？同步意味着在一个时间点只能有一个线程可以修改 hash 表，任何线程在执行 hashtable 的更新操作前都需要获取对象锁，其他线程则等待锁的释放。
56. 如何实现 hashmap 的同步？通过 Collections 的 synchizedMap()方法。
57. Hashmap 中使用链地址法来解决 hash 冲突（不同的 key 值得到相同的 hash 值）。
58. 在使用自定义的类作为 hashmap 的 key 时需要注意：一般需要重写 equals 和 hashCode 方法；最好把这个类设计成不可变类。
59. 与 Java 集合框架相关的有哪些最好的实践？

    （1）根据需要选择正确的集合类型。比如，如果指定了大小，我们会选用 Array 而非 ArrayList。如果我们想根据插入顺序遍历一个 Map，我们需要使用 TreeMap。如果我们不想重复，我们应该使用 Set。

    （2）一些集合类允许指定初始容量，所以如果我们能够估计到存储元素的数量，我们可以使用它，就避免了重新哈希或大小调整。

    （3）基于接口编程，而非基于实现编程，它允许我们后来轻易地改变实现。

    （4）总是使用类型安全的泛型，避免在运行时出现 ClassCastException。

    （5）使用 JDK 提供的不可变类作为 Map 的 key，可以避免自己实现 hashCode() 和 equals()。

    （6）尽可能使用 Collections 工具类，或者获取只读、同步或空的集合，而非编写自己的实现。它将会提供代码重用性，它有着更好的稳定性和可维护性。
60. 线程和进程？进程是指一段正在执行的程序（应用程序执行的最小单元）；线程是指程序代码的一个执行单元。线程拥有自己的栈，但是共享堆。
61. 多线程的优势？减少程序的响应时间；线程的创建和切换开销更小；提高 CPU 的利用率；简化程序的结构。
62. 同步和异步的区别？多线程同时对同一数据进行操作时，同步机制是指只能有一个线程对其进行读写，其他线程进入等待状态，直到线程结束对该资源的使用。异步是指每个线程都包含运行时自身所需要的数据和方法，不必关心其他线程的状态或行为。简单来说，同步就是我喊你吃饭，如果你没听到就一直喊你直到你和我一起去吃饭；而异步就是我喊你去吃饭但是不管你有没有回应我自己都去吃饭了。
63. 实现同步的方式：利用同步代码块；利用同步方法来实现同步。
64. 什么是死锁(deadlock)？两个进程都在等待对方执行完毕才能继续往下执行的时候就发生了死锁。结果就是两个进程都陷入了无限的等待中。
65. 如何确保 N 个线程可以访问 N 个资源同时又不导致死锁？使用多线程的

时候，一种非常简单的避免死锁的方式就是：指定获取锁的顺序，并强制线程按照指定的顺序获取锁。因此，如果所有的线程都是以同样的顺序加锁和释放锁，就不会出现死锁了。

66. .创建线程有几种不同的方式？你喜欢哪一种？为什么？有三种方式可以用来创建线程：继承 Thread 类；实现 Runnable 接口；应用程序可以使用 Executor 框架来创建线程池实现 Runnable 接口这种方式更受欢迎，因为这不需要继承 Thread 类。在应用设计中已经继承了别的对象的情况下，这需要多继承（而 Java 不支持多继承），只能实现接口。同时，线程池也是非常高效的，很容易实现和使用。
67. 一个类是否可以同时继承 Thread 与实现 Runable 接口？可以。
68. Run 方法和 start 方法有什么区别？run 只是跑一次方法，start 才是正在启动一个线程。
69. 多线程同步实现方法有哪些？synchronized 关键字（synchronized 方法和 synchronized 块）；wait 和 notify 方法；lock（lock()：以阻塞的方式获取锁。tryLock()：以非阻塞方式返回锁。）
70. Sleep 不会释放锁，而 wait 会释放锁。
71. Synchronized 与 lock 有什么不同？synchronized 使用 object 对象本身的 notify、wait、notifyAll 调度机制。而 lock 可以使用 condition 进行线程之间的调度，完成 synchronized 实现的所有功能。
72. 当一个线程进入一个对象的一个 synchronized()方法之后，其他线程是否可以进入该对象的其他方法？同步静态方法也是可以调用的。静态方法同步锁是当前类的字节码，非静态方法的同步锁是 this。如果这个方法的内部调用了 wait()方法，那么其他线程就可以访问同一对象的其他 synchronized 方法。如果没有 wait()方法并且其他方法都是 synchronized 的方法，那么就不能访问这个对象的其他方法。
73. 什么是守护进程？是指在程序运行时在后台提供一种通用服务的线程。守护线程一个例子就是垃圾回收器。
74. Join()方法的作用是让调用该方法的线程在执行完 run()方法后在执行 join 后面的代码。
75. Jdbc 处理事务采用的方法？commit()和 rollback()。
76. 脏读、不可重复读、虚读？
77. Class.forName()的作用是把类加载到 JVM 中。Statement 用于执行不带参数的简单 SQL，preparedStament 表示预编译的 SQL 语句的对象用于执行带参数的预编译 SQL 语句。CallableStatement 则表示提供用来调用数据库存储过程的接口。
78. JDO 是什么？JDO 是 Java 数据对象，一个用于存取某种数据仓库中的对象的标准化 API，它使开发人员能够间接地访问数据库。
79. JDBC 与 hiernate 以及 mybatis 有什么区别？Hibernate 以及 mybatis 是 JDBC 的封装，对数据库的访问还是通过 JDBC 完成的。
80. Class.forName()和 loadClass 的区别？loadClass 加载类时可以选择是否对类进行解析和初始化，而 forName 加载的时候会将 Class 进行解析和初始化。
81. JVM 使用哪种字符表示？Unicode characters
82. 什么时候用 assert？assert 是断言，断言是一个包含布尔表达式的语句，一般用于调试目的。

83. (A<5)?10.9:9　输出 9.0　　char x='x';int i=10;false?i:x;false?10:x 输出 120，x
84. 对于基本类型变量，Java 是传值的副本，对于一切对象型变量，传递的是引用的副本。
85. 序列化的用途：远程方法调用（RMI）；对 javabean 来说，bean 的信息通常在设计时就已经保存好了，供程序启动时使用。
86. Java 是如何管理内存的？其实就是对象的分配和释放问题。通过 new 为每个对象在堆中申请一块内存空间。对象的释放是由 GC 来做决定和执行的。
87. 什么是 Java 中的内存泄露？对象是可达的，但是对象对于程序来说确实无用的。这些对象不会被 GC 回收，任然占用着内存，这就造成了内存泄露。
88. 内存泄露主要由什么原因引起？保留下来却永远都不再使用的对象引用。
89. 如何确定内存泄露的位置？使用 JVM 检测工具，堆 dump。参数：-XX：-HeapDumpOutofMemoryError。
90. 在各种 List 中，最好的做法是以 ArrayList 作为默认选择，当插入，删除频繁时，使用 LinkedList；vector 总是比 ArrayList 慢。在各种 map 中，HashMap 用于快速查找。当元素个数固定时

# Java web

1. 页面请求的工作流程是怎么样的？首先用户通过浏览器输入链接地址来请求所需资源，浏览器接受用户请求，并把用户请求发送给服务器端，客户端与服务器端通过 HTTP 来完成交互，请求流中包含一些信息。服务器接收到客户端的请求后查找用户所需要的资源，并把该资源返回给客户端（特定的消息格式），浏览器对 HTML 进行解析然后展示结果给客户。
2. GET 与 POST 方法的区别？
3. 什么是 servlet？servlet 是采用 Java 语言编写的服务端程序，它运行于 Web 服务器中的 servlet 容器中，主要功能是提供请求/响应的 web 服务模式，动态生成 web 内容。
4. Servlet 的优势？较好的移植性；执行效率高；使用方便；可扩展性强。
5. 容器收到请求，创建 HttpServletResponse 以及 HttpServletRequest，找到对应的 servlet，针对请求创建一个单独的线程，将上面两个对象传入，容器调用 servlet 的 service 方法，service 方法会调用 doPost 或 doGet 方法完成具体的任务，同时返回给容器，容器把响应消息组装成 HTTP 格式返回给客户端，此时线程运行结束，同时删除两个创建的对象。
6. Servlet 的生命周期？Servlet 被服务器实例化后，容器运行其 init 方法，请求到达时运行其 service 方法，service 方法自动派遣运行与请求对应的 doXXX 方法（doGet，doPost）等，当服务器决定将实例销毁的时候调用其 destroy 方法。web 容器加载 servlet，生命周期开始。通过调用 servlet 的 init()方法进行 servlet 的初始化。通过调用 service()方法实现，根据请求的不同调用不同的 do***()方法。结束服务，web 容器调用 servlet 的 destroy()方法。简单来说就是 5 个步骤：加载，创建，初始化，处理客户请求，卸载。
7. 通常情况下，服务器只会创建一个 Servlet 实例对象，也就是说 servlet 实例一旦创建，它就会驻留在内存中，为后续的请求服务，直至 web 容器退出，才会被销毁。

8. JSP 与 Servlet 的异同？jsp 可以看做是一个特殊的 servlet，只不过是对 servlet 的扩展。Servlet 实现的方式是在 Java 中嵌套 HTML 代码，而 jsp 是在 HTML 中嵌套 Java 代码。Servlet 没有内置对象，jsp 中的内置对象都是必须通过 HttpServletResponse 以及 HttpServletRequest，httpServlet 对象得到的。
9. MVC 模式？
10. Servlet 中的 forward 和 redirect 有什么不同？forward 是转发，地址栏不变，一次请求；redirect 是重定向，地址栏变化，2 次请求。
11. Servlet 线程安全问题解决方法？servlet 实现 singlemodel 接口（标记接口）
12. Jsp 的 9 大内置对象？pageContext，request，session，application，response，config，out，exception，page。page（pagecontext 域） request（request 域） session（session 域） application（servletcontext 域）
13. ServletContext 与 servletConfig？servletConfig 获取初始化参数。ServletContext 对象通常也称为 context 域对象（应用程序范围）服务器启动时创建，每个 web 应用都有一个 servletContext 对象。ServletContext 域：1.这是一个容器 2.说明这个容器的作用范围
14. Request 对象的方法？
15. Jsp 有哪些动作？jsp:include：在页面被请求的时候引入一个文件。jsp:forward：把请求转到一个新的页面。jsp:useBean：寻找或者实例化一个 JavaBean。jsp:setProperty：设置 JavaBean 的属性。jsp:getProperty：输出某个 JavaBean 的属性。jsp:plugin：根据浏览器类型为 Java 插件生成 OBJECT 或 EMBED 标记
16. Jsp 中的 include 指令和 include 动作有什么区别？include 指令：<%@ include file="" %> 编译阶段的指令，在编译时就替换了最终形成文件，只有第一次请求时执行。是静态导入。Include 动作：<jsp:include page="" flush="">。这是运行时的语法，主页面被请求时才会被包含进来，类似于方法调用，每次请求都会执行。是动态导入。
17. 会话跟踪技术？page（pagecontext 域）request（request 域） session（session 域） application（servletcontext 域）
18. 保存会话数据的技术？cookie 和 session。cookie 是一种浏览器端的缓存技术，而 Session 是一种服务器端的缓存技术(依赖 cookie)
19. 登陆功能是用 Session 实现的,就是向 Session 对象中保存当前用户的对象。自动登录的功能用 Cookie 实现，就是登陆时将用户的信息保存为持久化 Cookie 下次访问时，读取请求中如果有用户信息的 Cookie 就可以自动登陆。
20. 如何防止表单重复提交？使用 session 技术。添加一个隐藏域，生成一个随机值，存入 session 中，处理请求时取出值进行比较，相等说明不是重复提交，不相等则说明是重复提交。
21. 什么是 AJAX？AJAX 技术是客户端技术，是一种支持异步请求的技术。
22. 什么是 JavaEE？JavaEE 是一个行业标准，主要通过 Java 开发服务器端应用提供一个独立的，可移植的，多用户的企业级平台，从而简化程序的开发和部署。
23. JavaEE 常用的术语？Web 服务器（IIS 和 Apache），web 容器（Tomcat，Jboss 等），EJB 容器，Applet 容器，appilcation client 容器，JDNI（Java 命名和目录接口），JMS（Java 消息服务），JTA（Java 事务服务，用于分布式事务等），JAF（JavaBean 激活框架，用于数据处理），RMI（远程方法调用）等。

24. 什么是 webservice？是一种基于网络的分布式模块化组件。基于以下一些协议来实现：XML，WSDL，UDDI，SOAP。
25. SOAP 与 REST 的区别？SOAP 数据使用 XML 数据格式，定义了一整套复杂的标签。REST 是面向资源的。

# JVM

1. Java 字节码文件的格式
2. 内部类的存储方式
3. 垃圾回收器的分类及优缺点
4. 类在虚拟机中的加载过程
5. 即时编译器的前后端优化方法
6. CMS 垃圾回收器的工作过程
7. CAS 指令以及其他线程安全的方法
8. 各种内存溢出的情况，包括 JNI 调用

1. Java 的内存区域？Java 运行时数据区主要有程序计数器（一块较小的内存区域，线程私有，执行 Java 方法计数器记录的是正在执行的虚拟机字节码指令的地址；执行 native 方法，这个计数器为空（undefined）。这个内存区域是唯一一个在 Java 中没有定义 OOM 的区域）；虚拟机栈（线程私有，生命周期与线程相同，为虚拟机执行 Java 方法或是字节码服务的，存放基本数据类型以及引用变量和 returnAddress 类型：指向了一条字节码指令的地址。这个区域存在两种异常：线程请求的栈深度大于虚拟机所提供的深度抛出 StackOverflowError；动态扩展大于虚拟机栈固定长度则抛出 OutOfMemoryError 异常）；本地方法栈（为虚拟机使用到的 native 方法服务。）；堆（线程共享，在虚拟机启动时创建，存放对象实例以及数组。也被成为 GC 堆，现在的垃圾回收算法是分代收集算法，Java 堆还可以分为新生代和老生代。如果在堆中没有内存完成实例分配，并且堆无法扩展，就会抛出 OOM 异常）；方法区（线程共享，用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码数据等。当方法区无法满足内存分配时就会抛出 OOM 异常）；常量池（是方法区的一部分）
2. 虚拟机中对象创建的过程（new 指令）？①检查指令的参数是否能在常量池中定位到一个类的符号引用，检查这个类是否已被加载，解析和初始化，如果没有就要进行类加载。②类加载检查③为新生对象分配内存（指针碰撞、空闲列表）④内存空间初始化零值⑤设置对象⑥执行 init 方法。
3. 对象的内存布局？对象在内存中存储的布局可以分为 3 块区域：对象头（自身运行数据，指针类型）、实例数据（真正的有效信息）、对齐填充（占位符）。
4. 对象的访问定位？Java 程序通过栈上的 reference 数据来操作栈上的具体对象。主流的访问方式有句柄和直接指针。
5. 应对解决 OutOfMermoryError(OOM)Java 堆内存问题？先通过内存映像分析工具对 dump 出来的堆转储快照进行分析，确认是内存泄露还是内存溢出。如果是内存泄露，进一步通过工具查看泄露对象到 GC Roots 的引用链，就可

以知道对象是怎样通过与 GC Roots 关联而导致 GC 不能自动回收的，这样就可以准确的定位出泄露的位置了。如果不存在泄露，那就是说内存中对象却是还是活着，那就要检查虚拟机的堆参数，与机器物理内存对比看看是否还可以调大，从代码上检查是否存在某些对象生命周期过长，持有状态时间过长的情况，尝试减少程序运行期间的内存消耗。

6. 栈内存溢出？栈空间无法继续分配。内存太小（StackOverflowError）；已使用的栈空间太大（OutOfMemoryError）。在单线程下，无论是栈容量太小还是栈帧太大，都是抛出 StackOverflowError。
7. 方法区和运行时常量池溢出？PermGen space　　intern()方法。
8. 本机直接内存溢出？
9. 判读对象已死？引用计数算法（好处：实现简单，判定效率也高。缺点：不能解决对象之间相互循环引用的问题），但是一般不推荐引用计数法。推荐使用可达性分析算法，通过一系列的 GC Roots 的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链，当一个对象到 GC Roots 没有引用链相连时则说明这个对象是不可用的。
10. 哪些对象可以作为 GC Roots？虚拟机栈中引用的对象，方法区中类静态属性引用对象，方法区中常量引用的对象，本地方法栈中 JNI 引用的对象。
11. 对象生存还是死亡，要经过两次标记？GC Roots 引用链一次标记——finalize()——F-Queue 队列——Finalizer 线程——GC 二次标记——回收，任何一个对象的 finalize 方法只会被系统自动调用一次，如果对象面临下一次回收，就不会再调用 finalize 了。
12. 无用的类？该类的所有实例都已经被回收；加载该类的 classloader 已经被回收；该类对应的 java.lang.Class 对象已经没有任何引用，无法再任何地方进行反射访问该类。
13. 垃圾收集算法？标记-清除算法（最基础的收集算法，但是存在两个问题：效率问题，空间问题（内存碎片））。复制算法（将内存按容量划分为大小相等的两块，每次使用一块，对整个半区进行回收并且复制到另一块上，解决了效率问题以及碎片问题，但是存在的问题是将内存缩小了原来的一半，代价较高。现在一般都是使用复制算法来回收新生代，但是进行了改进，提出了将内存分为一块较大的 Eden 空间和两块较小的 Survivor 空间，每次使用其中的 Eden 和其中一块 Survivor，还有就是需要老生代进行分配担保）。标记-整理算法（前期和标记-清理算法一样，后续不是直接进行对象的清理而是让所有存活的对象都向一端移动，直接清理端边界以外的内存）。现在虚拟机一般都采用分代收集算法（老生代：标记-清除算法或者标记-整理算法；新生代：复制算法）
14. HotSpot 虚拟机垃圾收集算法实现？枚举根节点（GC Roots 的枚举，使用一组 OopMap 的数据结构。）；安全点（特定的位置记录 OopMap，只有在到达安全点的时候才能暂停。抢断式中断（当 GC 时，把所有线程都中断，如果发现线程中断的地方不在安全点上，就恢复线程让它跑到安全点上）和主动式中断（当 GC 需要中断时，不直接操作线程，仅仅设置一个标志位，各线程执行时主动轮询这个标志位，发现为真时就自己中断挂起。））
15. 垃圾收集器？serial 收集器（单线程，暂停其他所有工作线程，用于运行在 client 模式下的虚拟机）、ParNew 收集器（其实就是 serial 收集器的多线程版本，适合于运行在 server 模式下的虚拟机，目前只有它能与 CMS 收集器（老

生代收集器）配合工作）、Parallel Scavenge 收集器（新生代收集器，使用复制算法，并行的多线程收集器。可以控制吞吐量）、Serial Old 收集器（是 serial 收集器的老年代版本，单线程收集器，使用标记-整理算法，client 模式下虚拟机使用）、parallel old 收集器（parallel scavenger 收集器的老年代版本，使用多线程和标记-整理算法）、CMS 收集器（以停顿时间最小化为目标，基于标记-清除算法，老生代收集器，并发收集，低停顿，但是不足之处是：对 CPU 资源非常敏感；无法处理浮动垃圾；产生很多碎片。）、**G1 收集器（面向服务端应用的新生代+老生代收集器，主要特点：并发与并行；分代收集；空间整合；可预测的停顿，JDK7 引入）**

16. CMS 收集器的整个过程？初始标记（仅标记一下 GC Roots 直接关联到的对象）——并发标记（进行 GC Roots Tracing 的过程，并发实现）——重新标记（修正并发标记期间因用户程序继续运转而引起的标记变动的那部分标记记录）——并发清除
17. G1 收集器运行的过程？初始标记——并发标记——最终标记——筛选回收
18. 理解 GC 日志？阅读 GC 日志是处理 Java 虚拟机内存问题的基础技能。如果有“Full”表示这次 GC 发生了 Stop-The-World 的。
19. 内存分配与回收策略？Java 体系中的自动内存管理主要解决了给对象分配内存以及回收分配给对象的内存的问题。内存分配策略：对象优先在 Eden 分配（当 Eden 没有足够空间进行分配时，虚拟机将进行一次 minor GC）;大对象直接进入老年代（最典型的大对象就是那种很长的字符串以及数组，可以通过设置参数来实现）；长期存活的对象进入老年代（给每个对象定义一个对象年龄计数器，当年龄达到多少时就将该对象晋升到老生代，可以通过设置参数来设置阈值）；动态对象年龄判定（如果在 Suvivor 空间中相同年龄所有对象大小的总和大于 Survivor 空间的一半，那么年龄大于该年龄的对象就可以直接进入老年代，无需等到 MaxTenuringThreshold 中要求的年龄）；空间分配担保。
20. Minor GC 与 Major GC/Full GC？minor GC 是指发生在新生代的垃圾回收动作，非常频繁而且回收速度比较快；Full GC 是指发生在老年代的 GC 出现了 Major GC 一般至少会一次 Minor GC。但是速度比较慢。
21. JDK 监控以及故障处理工具？

    jps（显示指定系统内的虚拟机进程）:虚拟机进程状况工具。可以列出正在运行的虚拟机进程。

    Jstat：虚拟机统计信息监视工具，用于监视虚拟机各种运行状态信息的命令行工具。显示本地或远程虚拟机进程中的类装载、内存、垃圾收集、JIT 编译等运行参数。

    Jinfo：Java 配置信息工具，实时地查看和调整虚拟机各项参数。

    Jmap：Java 内存映像工具，用于生成堆转储快照（heapdump 或 dump）。还可以查询 finalize 执行队列、Java 堆和永久代到的详细信息。

    Jhat：虚拟机转储快照分析工具。与 jmap 搭配使用来分析 jmap 生成的堆转储快照。

    Jstack：Java 堆跟踪工具。用于生成虚拟机当前时刻的线程快照（threaddump 或者 Javacore 文件）。主要目的是定位线程出现长时间停顿的原因，如线程死锁，死循环，请求外部资源等待过长等。
22. HSDIS 是一个虚拟机 JIT 编译代码的反汇编插件。

23. JDK 的可视化工具？JConsole 和 VisualVM。
24. 高性能硬件上的程序部署问题？默认使用吞吐量优先的收集器，一次 Full GC 回收堆的停顿时间较长，导致网站失去响应。可以使用若干集群的方式，无 session 复制的亲合式集群（即 web 应用服务器集群，负载均衡器，均衡器按照算法（sessionID 分配）将一个用户请求永远分配到固定的一个集群节点进行处理。），可以将垃圾回收器改为 CMS 收集器。
25. 集群间同步导致内存溢出？同步导致集群间网络交互非常频繁，当网络情况不能满足传输要求时，重发数据就会在内存中不断堆积，很快就会产生内存溢出。
26. 堆外内存导致到的溢出错误？操作系统对每个进程的内存有限制导致的错误。虽然设置 Java 堆内存很大，但是还要看操作系统实际分配多少。除了 Java 堆和永久代之外，我们还应该注意以下这些区域还会占用较多的内存：Direct Memory；线程堆栈；Socket 缓存区；JNI 代码；虚拟机和 GC。
27. 外部命令导致系统缓慢？
28. 服务器 JVM 进程崩溃？集成时会遇到集成方有许多接口不能用，导致积累了许多 web 服务没有调用完成，进而导致等待的线程和 socket 连接越来越多，最终使得 JVM 崩溃。可以使用生产者/消费者模式（生产者—缓冲池—消费者）
29. 不恰当数据结构导致内存占用过大？例如 HashMap<Long,Long>
30. 由 Windows 虚拟内存导致的长时间停顿？设置参数 -Dsun.awt.keepWorkingSetOnMinimize=true。
31. Eclipse 运行速度调优？eclipse.ini 文件中编辑。编译时间和类加载时间的优化。可以禁止掉字节码验证部分过程（-Xverify:none）；调整内存设置控制垃圾收集频率（Xms 与 Xmx 设置的值要恰当）；选择收集器降低延迟
32. 实现语言无关性的基础仍然是虚拟机和字节码存储格式。Java 虚拟机只和 Class 文件有所关联。
33. Class 文件是一组以 8 位字节为基础单位的二进制流，中间没有分隔符，几乎全是程序运行时必要的数据。Class 文件格式采用一种类似于 C 语言结构体的伪结构来存储数据，只有两种类型：无符号数（基本数据类型）和表（复合数据类型）。
34. 每个 class 文件的头 4 个字节被称为魔数，它的作用是确定这个文件是否为一个能被虚拟机接受的 class 文件。
35. Class 文件结构：首先是魔数，然后是文件版本号（版本号+次版本号+主版本号），接下来就是常量池入口，常量池主要存放两大类常量（字面量（字符串，声明为 final 的常量值）和符号引用（类和接口的全限定名，字段的名称和描述符，方法的名称和描述符）），常量池结束之后紧接着是两个字节代表访问标志（access_flags，2 个字节），这个标志用于识别一些类或者接口层次的访问信息。然后是类索引和父类索引以及接口索引集合，由这三项数据来确定这个类的继承关系。字段表集合（字段访问标志，名称索引，描述符索引，属性表集合），用于描述接口或类中声明的变量。方法表集合。属性表集合（Java 虚拟机运行时会忽略掉它不认识的属性）。
36. Code 属性？Java 程序方法体中的代码经过 javac 编译器处理后，最终变为字节码指令存储在 Code 属性中。Code 属性出现在方法表的属性集合之中。
37. This 关键字的实现机制？通过 javac 编译器编译的时候把对关键字 this 的访

问转变成对一个普通方法参数的访问，然后在虚拟机调用实例方法时自动传入此参数。

38. 编译器使用异常表而不是简单的跳转命令来实现 Java 异常以及 finally 处理机制。
39. Exceptions 属性，是在方法表中与 code 属性平级的属性。不同于上面的异常表。作用是列举出方法中可能抛出的受检查异常，也就是方法后面 throws 后面列举的异常。
40. LineNumberTable 属性：描述 Java 源码行号与字节码行号之间的对应关系。（不是必须的）
41. LocalVariableTable 属性：描述栈帧中局部变量表中的变量与 Java 源码中定义变量之间的关系。也不是必需的。
42. ConstantValue 属性：通知虚拟机自动为静态变量赋值。只有被 static 关键字修饰的变量才可以使用这个属性。
43. InnerClass 属性用于记录内部类与宿主类之间的关联。
44. Statckmaptable 属性用于代替基于数据分析的类型推导验证器。Code 属性中最多只能允许一个 statckmaptable 属性。
45. Signature 属性记录泛型签名信息。因为在 Java 中泛型的实现是擦除法的伪泛型，在字节码中，泛型信息编译后通通擦除了。实现擦除法的好处是实现简单，非常容易实现 backport，运行期也能节省一些类型所占的内存。
46. 字节码指令：采用面向操作数栈而不是寄存器的架构。大多数指令不包含操作数，只包含一个操作码。
47. 虚拟机的类加载机制？类加载的时机：加载-连接（验证-准备-解析）-初始化-使用-卸载。什么时候开始类加载过程中的加载阶段这个由虚拟机具体实现来把握。但是对于初始化阶段只有 5 种情况必须立即进行初始化（遇到 new、getstatic、putstatic、invokestatic（使用到的 Java 场景：使用 new 实例化对象，读取或设置静态字段，调用一个静态方法）；对类进行反射调用时；初始化一个类时发现父类还没有进行初始化；虚拟机启动时用户需要制定一个执行的主类；JDK1.7 动态语言支持），注意：通过子类引用父类的静态字段，不会导致子类初始化！通过数组定义引用类，不会触发此类的初始化！常量在编译阶段就会存入调用类的常量池中，本质上并没有直接引用到定义常量的类，因此不会触发定义常量的类的初始化！
48. 接口中不能使用“static{}”语句块。
49. 类加载的方式？本地编译好的 class 中直接加载；网络加载：java.net.URLClassLoader 可以加载 url 指定的类；从 jar、zip 等等压缩文件加载类，自动解析 jar 文件找到 class 文件去加载 util 类；从 java 源代码文件动态编译成为 class 文件
50. 类加载的过程？加载，验证，准备，解析，初始化。加载是类加载过程的一个阶段。
51. 加载阶段的工作？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 java 堆中生成一个代表这个类的 java.lang.Class 对象，做为方法区这个类的各种数据的访问入口。
52. 加载阶段完成之后二进制字节流就按照虚拟机所需的格式存储在方法区去中。

53. 类加载验证的目的？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。
54. 验证阶段的检验动作？
    ① 文件格式验证：验证字节流是否符合 Class 文件格式的规范，并且能被当前版本的虚拟机处理。
    ② 元数据验证：对字节码描述的信息进行语义分析，以确保其描述的信息符合 java 语言规范的要求。
    ③ 字节码验证：这个阶段的主要工作是进行数据流和控制流的分析。任务是确保被验证类的方法在运行时不会做出危害虚拟机安全的行为。
    ④ 符号引用验证：这一阶段发生在虚拟机将符号引用转换为直接引用的时候（解析阶段），主要是对类自身以外的信息进行匹配性的校验。目的是确保解析动作能够正常执行。
55. 准备阶段是正式为变量（被 static 修饰的变量，不包括实例变量）分配内存并设置初始值（通常情况下是指数据类型的零值，如果存在 ConstantValue 属性，那就在准备阶段就直接赋值初始化了），这些内存都将在方法区中进行分配，这里的变量仅包括类标量不包括实例变量。
56. 解析是虚拟机将常量池的符号引用替换为直接引用的过程。
    ① 什么是符号引用和直接引用？符号引用：符号引用以一组符号来描述所引用的目标，符号可以是任意形式的字面量，只要使用时能无歧义地定位到目标即可。符号引用与虚拟机实现的内存布局无关，引用的目标并不一定已经加载到内存中。直接引用：直接引用可以是直接指向目标的指针，相对偏移量或是一个能间接定位到目标的句柄。直接饮用是与内存布局相关的。
    ② 字段解析：类本身→接口→父类→返回引用权限验证，但在编译器中如果有一个同名字段同时出现在类的接口和父类中，或者同时在自己或者父类的多个接口中出现，那么编译器可能会拒绝执行。具体见 P233。
    ③ 类方法解析：类本身→父类→接口（如果匹配这说明是抽象类，抛出异常）→返回引用权限验证或宣告失败抛出异常（IllegalAccessError）
    ④ 接口方法解析：接口本身→父接口（不存在访问权限验证）
57. 类初始化阶段是类加载过程的最后一步，前面过程中基本都是由虚拟机主导和控制的。到了初始化阶段才真正开始执行自定义的 Java 代码或者说是字节码。初始化是根据程序员制定的主观计划区初始化变量和其他资源，或者可以从另外一个角度来表达：初始化阶段是执行类构造器<clinit>()方法的过程。
58. <clinit>()方法是由编译器自动收集类中所有类变量的赋值动作和静态代码块（static{}）中的语句合并产生的。静态代码块中只能访问到定义在静态代码块语句之前的变量，定义在它之后的变量，可以赋值但不能访问。（例子 P225）
59. <clinit>()方法不需要显式调用父类构造器，虚拟机会保证在子类的<clinit>()方法前已经执行了父类的<clinit>()方法。虚拟机中第一个被执行的<clinit>()方法一定是 Object。
60. 类与类加载器一同确立了类在 Java 虚拟机中的唯一性，每个类加载器，都拥有一个独立的类名称空间。
61. 不同类加载器对 instanceof 关键字运算结果会有影响。P229
62. JVM 三种预定义类型类加载器，当一个 JVM 启动的时候，Java 缺省开始使用如下三种类型类装入器：

① 启动（Bootstrap）类加载器：引导类装入器是用本地代码实现的类装入器，它负责将 <Java_Runtime_Home>/lib 下面的类库加载到内存中。由于引导类加载器涉及到虚拟机本地实现细节，开发者无法直接获取到启动类加载器的引用，所以不允许直接通过引用进行操作。

② 标准扩展（Extension）类加载器：扩展类加载器是由 Sun 的 ExtClassLoader（sun.misc.Launcher$ExtClassLoader）实现的。它负责将< Java_Runtime_Home >/lib/ext 或者由系统变量 java.ext.dir 指定位置中的类库加载到内存中。开发者可以直接使用标准扩展类加载器。

③ 系统（System）类加载器：系统类加载器是由 Sun 的 AppClassLoader（sun.misc.Launcher$AppClassLoader）实现的。它负责将系统类路径（CLASSPATH）中指定的类库加载到内存中。开发者可以直接使用系统类加载器。

63. 类加载器之间的这种层次关系称为累加器的双亲委派模型。双亲委派机制通俗的讲，就是某个特定的类加载器在接到加载类的请求时，首先将加载任务委托给父类加载器，依次递归，如果父类加载器可以完成类加载任务，就成功返回；只有父类加载器无法完成此加载任务时，才自己去加载。

64. 


65. 委托机制的意义？防止内存中出现多份同样的字节码

比如两个类 A 和类 B 都要加载 System 类：如果不用委托而是自己加载自己的，那么类 A 就会加载一份 System 字节码，然后类 B 又会加载一份 System 字节码，这样内存中就出现了两份 System 字节码。如果使用委托机制，会递归的向父类查找，也就是首选用 Bootstrap 尝试加载，如果找不到再向下。这里的 System 就能在 Bootstrap 中找到然后加载，如果此时类 B 也要加载 System，也从 Bootstrap 开始，此时 Bootstrap 发现已经加载过了 System 那么直接返回内存中的 System 即可而不需要重新加载，这样内存中就只有一份 System 的字节码了。

66. 能不能自己写个类叫 java.lang.System？通常不可以，但可以采取另类方法达到这个需求。为了不让我们写 System 类，类加载采用委托机制，这样可以保证爸爸们优先，爸爸们能找到的类，儿子就没有机会加载。而 System 类是 Bootstrap 加载器加载的，就算自己重写，也总是使用 Java 系统提供的 System，自己写的 System 类根本没有机会得到加载。

67. 破坏双亲委派机制的三次？兼容 JDK；基础类回调用户代码的情况，使用线程上下文类加载器，典型的例子就是 JNDI；代码热替换，OSGI。

68. 使用线程上下文类加载器，可以在执行线程中，抛弃双亲委派加载链模式，使用线程上下文里的类加载器加载类。典型的例子有，通过线程上下文来加载第三方库 jndi 实现，而不依赖于双亲委派。大部分 java app 服务器 (jboss, tomcat..)也是采用 contextClassLoader 来处理 web 服务。还有一些采用 hotswap 特性的框架，也使用了线程上下文类加载器，比如 seasar

(full stack framework in japenese)。

69. 早期（编译期）优化：许多新生的 Java 语法特性都是靠编译器的语法糖来实现的，而不是依赖虚拟机底层改进来实现的。
70. 编译过程？解析与填充符号表过程→注解处理→语义分析与字节码生成
71. Java 编译动作的入口是 com.sun.tools.javac.main.JavaCompiler 类。
72. 解析与填充符号表：词法与语法分析，填充符号表。
73. 语义分析与字节码生成：标注检查，数据及控制流分析，解语法糖（指的是计算机语言中添加的某种语法对语言功能没有影响，但是更方便程序员使用，解语法糖的过程由 desugar()方法触发），字节码生成
74. Java 语言中的泛型实现方法称为类型擦除，基于这种方法实现的泛型称为伪泛型。注意泛型的重载！P313
75. 条件编译（条件为常量的 if 语句）:if 语句在编译阶段就会被运行，生成的字节码之中只包括条件成立的部分，不会包含 if 语句以及另一部分的子句。
76. 查看及分析即时编译结果？参数-XX:PrintCompilation 和-XX:PrintInlining。
77. 编译优化技术？一般来说，以编译方式执行本地代码比解释执行更快，因此，即时编译器产生的本地代码会比 javac 产生的字节码更加优秀。公共子表达式消除（语言无关）、数组范围检查消除（语言相关）、方法内联、逃逸技术。
78. Java 内存模型：主内存和工作内存。

# SQL

1. 触发器？特殊的存储过程；通过事件触发执行；强化约束；维护数据完整性和一致性。
2. 存储过程？预编译的 SQL 语句；允许模块化设计；需要执行多次 SQL 时，只需要创建一个存储过程，就可以在程序中多次调用，而且执行速度更快。可以用一个命令来调用存储过程。
3. 索引？特殊的查询表；相当于生活中书的目录，不需要查询表的内容就可以找到想要的数据，提高了查询的效率，加速了对数据的索引。但缺点是减慢了录入数据的速度，增加了数据库的大小。
4. 维护数据库的完整性和一致性，你喜欢用触发器还是自写业务逻辑？为什么？我是这样做的，尽可能使用约束，如 check,主键，外键，非空字段等来约束，这样做效率最高，也最方便。其次是使用触发器，这种方法可以保证，无论什么业务系统访问数据库都可以保证数据的完整新和一致性。最后考虑的是自写业务逻辑，但这样做麻烦，编程复杂，效率低下。
5. 什么叫视图？游标是什么？视图是一种虚拟的表，具有和物理表相同的功能。可以对视图进行增，改，查，操作，试图通常是有一个表或者多个表的行或列的子集。对视图的修改不影响基本表。它使得我们获取数据更容易，相比多表查询。游标：是对查询出来的结果集作为一个单元来有效的处理。游标可以定在该单元中的特定行，从结果集的当前行检索一行或多行。可以对结果集当前行做修改。一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。
6. 索引默认创建的都是 BTREE 索引。SQL 索引有两种，聚集索引和非聚集索引。聚集索引

存储记录是物理上连续存在，而非聚集索引是逻辑上的连续，物理存储并不连续。字典目录的例子：拼音查询法就是聚集索引，部首查询就是一个非聚集索引。

7. 创建索引的语法？CREATE INDEX index_name ON table(index_col_name)
8. 索引设计的原则？最适合索引的列是出现在 WHERE 子句中到的列，而不是 select 后面的列；使用短索引以及唯一索引；不要过度索引。
9. 为什么数据库对于 varchar 最大值设置为 8000,而不是 10000 呢？是由于数据页大小最大为 8K。
10. BTREE 索引与 HASH 索引？MEMORY 存储引擎的表。HASH 索引只能用于=和⇔操作符的等式比较。不能使用 hash 索引来加速 order by 操作。当对字段进行范围查询时，只有 BTREE 索引可以通过索引访问。
11. 创建视图？CREATE OR REPLACE VIEW view_name AS select 语句
12. 存储过程与函数的区别？
13. 事件调度器？可以将数据库按照自定义的时间周期触发某种操作，可以理解为时间触发器。默认是关闭的，通过 SET GLOBAL event_schelduler=1 打开。主要用于处理一些定时任务，比如定期清理日志表。
14. 触发器只能创建在永久表上，触发器的触发时间可以是 before 或者 after，触发事件可以是 INSERT、UPDATE、DELETE。对同一个表相同触发时间的相同触发事件只能定义一个触发器。
15. DELIMITER $$？
16. 触发器的创建？CREATE TRIGGER trigger_name trigger_time trigger_event ON tb FOR EACH ROW trigger_statement
17. 对于有重复记录的 INSERT，触发顺序是 BEFORE INSERT、BEFORE UPDATE、AFTER INSERT；对于没有重复记录的 INSERT，触发顺序只是 BEFORE INSERT、AFTER INSERT。
18. 触发器使用限制？不能调用将数据返回客户端的存储程序，不能采用 CALL 语句动态调用 SQL 语句。不能在触发器中开始或者结束事务。
19. MySQL 事务隔离级别？Read Uncommitted（读取未提交内容），Read Committed（读取提交内容），Repeatable Read（可重复读），Serializable（可串行化）。默认隔离级别是 Repeatable Read。
20. LOCK TABLE 和 UNLOCK TABLES？
21. 事务管理？SET AUTOCOMMIT、START TRANSACTION、COMMIT、ROLLBACK 等语句。MySQL 默认情况下是自动提交的。
22. 在锁表期间，用 start transaction 命令开始一个新的事务，会造成一个隐含的 unlock tables 被执行。对 lock 方式加的表锁，不能通过 rollback 的方式回滚。
23. 所有的 DDL 语句都是不能回滚的。事务可以通过定义 SAVEPOINT 指定回滚事务的一部分，但是不能指定事务提交一部分。
24. 分布式事务只支持 InnoDB 存储引擎。
25. 分布式事务的原理？使用分布式事务的应用程序设计一个或多个资源管理器以及一个事务管理器。资源管理器（RM）用于提供向事务资源的途径。事务管理器（TM）用于协调作为一个分布式事务一部分的事务。分为两个阶段，首先所有分支被 TM 告知要准备提交了；然后 TM 告知 RMs 是否要提交或者回滚。
26. SQL 安全问题？SQL 注入，利用某些数据库的外部接口将用户数据插入到实际的数据库操作语言中，从而达到入侵数据库乃至操作系统的目的。
27. 应用开发中采用的应对 SQL 安全的措施？使用 PrepareStatement+Bind-Variable 防止 SQL 注入；使用应用程序提供的转换函数；自定义函数进行校验。

28. SQL Mode：完成数据校验，保障数据的准确性；设置 SQL Mode 为 ANSI 模式，满足数据库之间迁移。可以设置 mode 为 STRICT_AS_CONTACT（严格模式）实现数据的严格校验。校验日期数据合法性（TRADITIONAL，会直接提示日期非法，拒绝插入）；启用 NO_BACKSLASH_ESCAPES 模式，使反斜线成为普通字符。启用 PIPES_AS_CONCAT，将“||”作为字符串连接符。SET SESSION sql_mode=’’；
29. 什么是 MySQL 分区？分区是指按照一定的规则，数据库把一个表分解为多个更小，更容易管理的部分。分区通过“分而治之”的方法管理数据表，提高了数据处理的并行度从而提升性能。
30. MySQL 分区的优点？可以存储更多的数据；优化查询；对于已经过期或者不需要的数据可以快速的删除；跨多个磁盘来分散数据查询，获得更大的吞吐量。
31. MySQL 的分区适用于一个表的所有数据和索引，MySQL 分区表上创建的索引一定是本地 LOCAL 索引。
32. MySQL 分区类型：RANGE 分区（基于一个给定连续区间范围，把数据分配到不同的分区）；LIST 分区（基于枚举出的值列表分区）；HASH 分区（基于给定的分区个数，把数据分配到不同的分区）；KEY 分区（类似于 HASH 分区）
33. 分区语句？
    RANGE：PARTITION BY RANGE COLUMNS (id)(PARTITION p0 VALUES LESS THAN (‘1997-01-12’))；NULL 值被当做最小值来处理
    LIST：PARTITION BY LIST (CATEGORY)(PARTITION p0 VALUES IN (‘food’,’vegetable’))；NULL 必需出现在枚举列表中，否则不被接受
    HASH：PARTITION BY HASH (id) PARTITIONS 4；NULL 被当做零值处理
    KEY：PARTITION BY KEY (id) PARTITIONS 4；NULL 被当做零值处理

# 项目

1. 什么是单点登录？以及单点登录实现的原理？
   单点登录是一种在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统的技术。
2. SAML 是什么？
   SAML 是基于 XML 的标准，叫做安全断言标记语言，一般用于不同安全域之间的身份认证和数据交换。
3. 项目难点？统一身份认证系统项目主要是基于开源软件 WSO2 Identity Server，项目难点在于我们要支持多协议的单点登录功能，包括 SAML2.0，OAuth2.0 以及 openconnect，sts。我们要了解每个协议或标准的整个登录流程，因为现在国内再用的单点登录系统一般都是 CAS，所以 WSO2 的资料比较少，在了解各个协议的过程中，我们必须阅读源码，了解它的实现，并针对它的实现进行登录样例的开发。因为下面一个宁波市组织架构管理系统需要用到该项目的接口，所以我们还需要整理出它提供的各种接口，包括 WSDL 文件以及部分 REST 的接口。针对其提供的统一登录界面，我们需要根据客户的需求进行更改，但我们不能修改其实现的业务逻辑，所以在这方面也花了较长时间。

4. 项目的亮点？项目的亮点是 WSO2 开源系统本来是一个非常臃肿的，重量级的单点登录系统，我们对其进行了精简，提供了一个能满足用户需求的统一认证平台，通过一段时间的研究，形成了比较完整的二次开发中文文档。
5. 组织架构项目的不足？没有考虑高并发的情况，但是这也基本上能满足宁波市政府人员的需求了，如果接下来他们要对大众开放的话，我们还需要对其进行优化。还有就是因为需要调用 WSO2 的接口，所以会导致系统依赖于网络环境以及统一认证服务器。数据库可扩展性方面也有所欠缺。
6. 组织架构项目的优点？对外提供了比较完善的 REST 接口，方便宁波市各个部门对数据的调用。
7. 理财系统？

# 多线程

1. 当并发执行累加操作不超过百万次时，速度会比串行执行累加操作的要慢。这是因为线程有创建和上下文切换的开销。
2. 测试上下文切换次数和时长？使用 Lmbench3 可以测量上下文切换的时长；使用 vmstat 测量上下文切换的次数。
3. 如何减少上下文切换？无锁并发编程（可以使用一些方法来避免使用锁，比如将数据的 ID 按照 Hash 算法取模分段，不同的线程处理不同段的数据）、CAS 算法（Java 的 Atomic 包使用 CAS 算法来更新数据，不需要加锁）、使用最少线程（避免创建不需要的线程）、使用协程（在单线程里实现多任务调度，并在单线程里维持多个任务间的切换）。
4. 减少线程上下文具体做法？①用 jstack 命令 dump 线程信息，看看进程里的线程都在做什么；②统计线程分别处于什么状态；③打开 dump 文件查看具体的线程在做什么；④进行下一步优化工作
5. 死锁？线程之间互相等待对方释放锁。
6. 介绍避免死锁的几个常见方法？避免一个线程同时获取多个锁；避免一个线程在锁内同时占用多个资源，尽量保证每个锁只占用一个资源；尝试使用定时锁，使用 lock.tryLock 来代替使用内部锁机制；对于数据库锁，加锁和解锁必须在一个数据库里。

# 设计模式

1. 设计模式的目的是为了代码的重用，避免程序大量修改，使得代码更加易于理解。
2. 常见的设计模式：工厂模式，单例模式，适配器模式，享元模式以及观察者模式。
3. 单例模式：确保某一个类只有一个实例，而且自行实例化，并在整个应用程序的生命周期中只存在一个实例。
4. 单例模式和全局变量的区别？全局变量是对一个对象的静态引用，虽然可以提供全局访问的功能，但是它并不能保证应用程序只有一个实例。代码规范中指明要少用全局变量，最后全局变量不能实现继承。
5. 单例模式需要注意的：首先构造函数私有；然后提供一个全局访问点。注意多线程下单

例的问题。

6. 工厂模式：专门负责实例化大量公共接口的类。动态决定将哪一个类实例化。工厂模式的几种形态：简单工厂模式；工厂方法模式；抽象工厂模式。
7. 简单工厂模式：工厂类根据提供给他的参数返回几个可能产品中的一个类的实例，通常返回类都有一个公共的父类或者公共的方法。优点：客户端不需要修改代码。缺点：当需要增加新的产品类的时候，不仅需新加产品类，还要修改工厂类，违反了开闭原则。
8. 工厂方法模式：这个和简单工厂有区别，简单工厂模式只有一个工厂，工厂方法模式对每一个产品都有相应的工厂。优点：首先完全实现‘开－闭 原则’，实现了可扩展。其次更复杂的层次结构，可以应用于产品结果复杂的场景。
9. 抽象工厂模式：抽象工厂模式是所有形态的工厂模式中最为抽象和最具一般性的一种形态。
10. 工厂方法和抽象工厂的比较？

    工厂方法模式：一个抽象产品类，可以派生出多个具体产品类。
    一个抽象工厂类，可以派生出多个具体工厂类。
    每个具体工厂类只能创建一个具体产品类的实例。

    抽象工厂模式：多个抽象产品类，每个抽象产品类可以派生出多个具体产品类。
    一个抽象工厂类，可以派生出多个具体工厂类。
    每个具体工厂类可以创建多个具体产品类的实例。

    区别：工厂方法模式只有一个抽象产品类，而抽象工厂模式有多个。
    工厂方法模式的具体工厂类只能创建一个具体产品类的实例，而抽象工厂模式可以创建多个。
11. 适配器模式：把一个类的接口转换成客户端所期待的另一个接口。类适配器；对象适配器。
12. 类适配器： Adapter 类既继承了 Adaptee （被适配类），也实现了 Target 接口（因为 Java 不支持多继承，所以这样来实现），在 Client 类中我们可以根据需要选择并创建任一种符合需求的子类，来实现具体功能。继承了被适配类，同时实现目标接口。
13. 对象适配器：它不是使用多继承或继承再实现的方式，而是使用直接关联，或者称为委托的方式。组合被适配类，同时实现目标接口。
14. 观察者模式（发布/订阅模式）：优点：观察者模式解除了主题和具体观察者的耦合，让耦合的双方都依赖于抽象，而不是依赖具体。从而使得各自的变化都不会影响另一边的变化。缺点：依赖关系并未完全解除，抽象通知者依旧依赖抽象的观察者。
15. 上述几种设计模式的类图以及代码要熟悉。
16. 生产者-消费者问题的 Java 实现。（1）wait() / notify()方法（基类 Object 的两个方法）（2）await() / signal()方法（3） BlockingQueue 阻塞队列方法（4） PipedInputStream / PipedOutputStream

# Linux 命令

1. 查找文件（find，grep，ls，head，tail）
2. 查看一个程序是否运行（ps -ef|grep tomcat） ps
3. 终止线程（kill -9）

4. 创建文件（mkdir）；删除文件（rmdir;rm）
5. 复制文件（cp）;远程拷贝（scp）
6. 移动文件（mv）
7. 修改文件权限（chmod 777）
8. 压缩文件（tar -czf）;解压文件（tar -xvzf）
9. 查看端口占用情况（netstat）
10. 文件下载（curl，wget）

# Mybatis

1. #{...} 和 ${...} 的区别？MyBatis 将 #{…} 解释为 JDBC prepared statement 的一个参数标记。而将 ${…} 解释为字符串替换。#方式能够很大程度防止 sql 注入。一般能用#的就别用$。MyBatis 排序时使用 order by 动态参数时需要注意，用$而不是#。
2. 如何使用 LIKE？like "%"#{name}"%"；like '%'||#{name}||'%'；like '%${name}%'。建议使用第一种，经过预编译 SQL。
3. 如何获取自动生成的(主)键值？insert 方法总是返回一个 int 值 - 这个值代表的是插入的行数。而自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。
4. 在 mapper 中如何传递多个参数？①#{0}代表接收的是 dao 层中的第一个参数，#{1}代表 dao 层中第二参数，更多参数一致往后加即可。②采用 map 传参③使用@param 注解。
5. 什么是 MyBatis 的接口绑定,有什么好处？接口映射就是在 IBatis 中任意定义接口,然后把接口里面的方法和 SQL 语句绑定。
6. Mybatis 如何实现接口绑定？接口绑定有两种实现方式，一种是通过注解绑定,就是在接口的方法上面加上@Select@Update 等注解里面包含 Sql 语句来绑定，另外一种就是通过 xml 里面写 SQL 来绑定，在这种情况下,要指定 xml 映射文件里面的 namespace 必须为接口的全路径名。
7. MyBatis 实现一对一有几种方式？具体怎么操作的？有联合查询和嵌套查询，联合查询是几个表联合查询，只查询一次，通过在 resultMap 里面配置 association 节点配置一对一的类就可以完成；嵌套查询是先查一个表，根据这个表里面的结果的外键 id，去再另外一个表里面查询数据,也是通过 association 配置,但另外一个表的查询通过 select 属性配置。
8. MyBatis 实现一对多有几种方式？怎么操作的？有联合查询和嵌套查询,联合查询是几个表联合查询,只查询一次,通过在 resultMap 里面配置 collection 节点配置一对多的类就可以完成;嵌套查询是先查一个表,根据这个表里面的结果的外键 id,去再另外一个表里面查询数据,也是通过配置 collection,但另外一个表的查询通过 select 节点配置。
9. MyBatis 里面的动态 Sql 是怎么设定的？用什么语法？MyBatis 里面的动态 Sql 一般是通过 if 节点来实现,通过 OGNL 语法来实现,但是如果要写的完整,必须配合 where,trim 节点,where 节点是判断包含节点有内容就插入 where,否则不插入,trim 节点是用来判断如果动态语句是以 and 或 or 开始,那么会自动把这个 and 或者 or 取掉。
10. MyBatis 里面的核心处理类叫做 SqlSession。
11. 讲下 MyBatis 的缓存？MyBatis 的缓存分为一级缓存和二级缓存。一级缓存放在 session 里面，默认就有，二级缓存放在它的命名空间里，默认是打开的；使用二级缓存属性类

需要实现 Serializable 序列化接口(可用来保存对象的状态)，可在它的映射文件中配置<cache/>。

12. .MyBatis(IBatis)的好处是什么？mybatis 把 sql 语句从 Java 源程序中独立出来，放在单独的 XML 文件中编写，给程序的维护带来了很大便利。mybatis 封装了底层 JDBC API 的调用细节，并能自动将结果集转换成 Java Bean 对象，大大简化了 Java 数据库编程的重复工作。灵活控制 sql 语句。

# 数据结构与算法

1. 实现单链表的增删操作？
   增加操作（插入数据到 $a_{i-1}$~$a_i$ 之间）：首先要找到 $a_{i-1}$ 的引用 p;生成一个数据域为 x 的新节点 s；设置 p.next=s,s.next=a。
   删除操作（第 i 个节点删除）：首先要找到 $a_{i-1}$ 的引用 p；让 p 直接指向后继节点 $a_{i+1}$ 即 p.next= $a_{i+1}$。
2. 如何从链表中删除重复数据？第一种方法是遍历链表，把遍历的值存储到 hashtable 中，如果存在重复，则说明可以删除。但这种方法时间复杂度较低，遍历过程中需要额外的空间来存储已经遍历过的值。第二种方法是对链表进行双重遍历，这种方法不需要额外的存储空间，但是时间复杂度要高。外循环正常遍历链表，假设外循环当前遍历的节点是 cur，内循环就从 cur 开始遍历，若碰到相同的节点值则删除这个节点。
3. 如何找出单链表中的倒数第 K 个元素？遍历求出链表长度，正向遍历 n-k；高效做法：两个指针，让其中一个指针比另一个指针先前移 k-1 步，然后两个指针同时往前移，直到先行的指针为 null。
4. 如何实现链表的反转？需要调整指针的指向，需要记录下链表断掉的那个节点，因为遍历不到了，接下来还要找到反转后链表的头节点，也就是原始链表的尾节点，即 next 为空指针的节点。
5. 如何从头到尾输出单链表？①指针反转②栈实现③递归实现（每访问到一个节点，先递归输出它后面的节点，再输出该节点自身）
6. 如何寻找单链表中的中间节点？两个指针的方法，快指针一次走两步，慢指针一次走一步，快指针到达链表尾而慢指针恰好到达链表中部。
7. 如何检测一个链表是否有环？还是采用双指针的方法，快慢指针结合，如果快慢指针在快指针到达尾部之前出现快指针等于慢指针则说明这是一个带环的链表。
8. 如何判断两个链表是否相交？分别遍历两个链表，记录它们的尾节点，如果尾节点相同这说明两个链表相交。
9. 栈与队列？栈是后进先出。队列是先进先出。可以采用数组和链表来实现栈。
10. 求栈中的最小元素？使用空间换时间，使用两个栈结构，一个栈用于存储数据，一个栈用于存储栈中的最小元素。
11. 如何实现队列？队列也可以使用数组和链表两种方式实现。
12. 如何使用两个栈来模拟队列的操作？使用栈 A 和栈 B 模拟队列 Q，A 为插入栈，B 为弹出栈。栈 A 提供入队列的功能，栈 B 提供出队列的功能。要入队列直接入栈 A 即可。出队列要考虑栈 B 是否为空，若为空则依次弹出栈 A 的数据，放入栈 B 中，再弹出栈 B；如果栈 B 不为空，则直接弹出栈 B 的数据。
13. 如何使用两个队列来实现栈？使用队列 Q1 和队列 Q2 模拟栈 S，Q1 为入队列，Q2 为出队列。Q1 提供压栈的功能，队列 Q2 提供弹栈的功能。要压栈直接入队列 Q1，要弹栈

需要考虑队列 Q1 是否只有一个元素，若只有一个元素则让 Q1 中的元素出队列并输出即可，不然则队列 Q1 中的所有元素出队列，入队列 Q2，最后一个元素不要入队列 Q2，输出该元素，然后将队列 Q2 所有元素入队列 Q1。

14. 如何进行选择排序？对于给定的一组记录，经过第一轮比较得到最小的记录，将其与第一个记录进行交换，接着对不包含第一个记录的其他记录进行比较，得到最小的记录并与第二个记录交换，重复该过程，直到比较的记录只有一个为止。时间复杂度均为 $O(n^2)$，n 较小时使用较好。
15. 如何进行插入排序？对于给定的一组记录，初始时按照第一个记录自成一个有序序列，其余记录为无序序列。接着从第二个记录开始，按照记录的大小依次将当前处理的记录插入到之前的有序序列中，直至最后一个记录。时间复杂度最好时为 O(n)，最坏 $O(n^2)$，大部分有序时使用。
16. 如何进行冒泡排序？对于给定的记录，从第一个记录开始依次对相邻的两个记录进行比较，当前面的记录大于后面的就交换位置，进行一轮比较和换位后，记录中最大的会位于最末尾，然后对前面的记录进行第二轮比较，重复该过程。时间复杂度最好时为 O(n)，最坏 $O(n^2)$
17. 如何进行归并排序？对于给定的记录，首先将每两个相邻为 1 的子序列进行归并，得到 n/2 个有序子序列，再将其两两归并，反复执行此过程，直到得到一个有序序列。时间复杂度为 O(nlogn)
18. 如何进行快速排序？对于给定一组记录，通过一趟排序后，将原来得到序列分为两部分，其中前一部分的所有记录均比后一部分的所有记录小，然后再次对前后两部分的记录进行快速排序，递归该过程，直到序列中的所有元素都有序为止。时间复杂度最好时为 O(nlogn)。
19. 快速排序基准关键字的选取：①三者取中。将首、尾、中间值进行比较，取中值作为基准。②取随机值。
20. 快排与归并排序的区别？分组的策略不同，归并排序前一半元素为一组，后一半为一组，而快速排序是将小于基准值的为一组，大于基准值的为一组。两者都是采用分而治之的思想。但是分组简单的合并的时候就复杂。
21. 如何进行希尔排序？先将整个待排序的记录序列分割成若干个子序列分别进行直接插入排序，待整个序列中的记录基本有序时，再对全体记录进行一次直接插入排序。
22. 如何进行堆排序？一是构建堆，二是交换堆顶元素与最后一个元素的位置。时间复杂度为 O(nlogn)
23. 红黑树有以下属性

    1.节点是红色或黑色。

    2.根是黑色。

    3.所有叶子（外部节点）都是黑色。

    4.每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

    5.从每个叶子到根的所有路径都包含相同数目的黑色节点。

# 大型技术架构

1. 网站架构模式：分层（单一职责，MVC 分层，控制层-服务层-数据层），分割（不同功能与服务分割），分布式（服务调用需要通过网络，带来了网络问题；分布式数据一致性问题），集群，缓存，异步，冗余（冷备份，热备份），自动化，安全。
2. 常用的分布式方案：分布式应用与服务；分布式静态资源（动静分离 JS,CSS,图片等，减轻应用服务器的负载压力）；分布式数据与存储（读写分离；分库分表；缓存优化；传统数据库分布式部署；使用 NoSQL）；分布式计算。
3. 缓存：CDN（内容分发，部署在离用户最近的网络服务商），反向代理（部署在网站的数据中心），本地缓存（缓存在本机内存中，但是不适合大量的数据），分布式缓存（分布式缓存集群）。热点数据缓存，注意缓存的时间，避免出现数据脏读，影响数据一致性。
4. 异步：单一服务器可以通过多线程共享内存队列的方式实现；分布式系统中可以通过分布式消息队列实现。典型的模型就是生产者-消费者模式，两者之间不存在直接调用，只是保持数据结构不变。作用：提高网站可用性；加快网站响应速度；消除并发访问高峰。
5. 性能问题，网站响应速度慢，优化措施：通过浏览器缓存，页面压缩等；使用 CDN，动静分离，部署反向代理服务器，缓存热点文件；使用本地缓存与分布式缓存；使用消息队列，异步处理请求；代码层面使用多线程，内存管理等进行优化；数据库方面使用索引，缓存，优化 SQL，读写分离等。
6. 可用性：网站高可用的主要手段就是冗余，通过负载均衡服务器统一一个集群对外提供服务，有效的负载均衡策略；数据服务器进行实时备份，宕机时进行数据转移并恢复。
7. 网站可扩展性的主要手段是事件驱动架构和分布式服务。事件驱动通常将请求构造成消息发布到消息队列之中，消息处理者通过消息队列中获取消息进行处理。分布式服务是将业务与基础服务分离开来。
8. 性能优化策略：首先要进行性能数据的搜集，然后针对性能报告进行性能分析，检查请求处理的各个环节的日志信息，分析是哪个环节响应时间较长，检查监控数据，分析影响性能的是硬件设施（内存，磁盘，CPU，网络）还是代码问题还是架构设计不合理，亦或是系统资源不足等。找到问题的原因后再针对不同的问题进行相应的优化。
9. 浏览器访问优化：减少 http 请求（合并 CSS,JS 文件，图片等），http 协议是无状态的应用层协议，意味着每次 http 请求都需要建立通信链路，进行数据传输。使用浏览器缓存（设置 http 的头字段）。启用压缩（对 html，css，js 文件进行 GZIP 压缩）
10. 网站优化第一定律：优先考虑使用缓存优化性能。
11. 缓存的基本原理：缓存指的是将数据存储在相对较高访问速度的存储介质中，减少数据访问的时间。缓存的本质是一个内存 Hash 表，数据缓存以一对 Key,Value 的形式存储在内存 Hash 表中。Hash 表数据读写的时间复杂度为 O（1）。缓存中主要用来存放读写比较高，很少变化的数据，应用程序先读写缓存，缓存中没有或者数据失效再去数据库中查询，并将查询到的数据写入缓存。缓存要考虑数据一致性问题与脏读。可以设置策略是数据更新后马上更新缓存。缓存雪崩问题通过分布式缓存解决。
12. 分布式缓存：缓存部署在多个服务器组成的集群上。两种缓存架构方式：JBoss Cache 为代表的需要更新同步的分布式缓存；以 Memcached 为代表的不互相通信的分布式缓存。
13. JBoss Cache 的分布式缓存在集群中的所有服务器上都保存相同的缓存数据，当某台服务器缓存更新时，会通知集群中所有的机器进行缓存更新或清除缓存。一般会将 JBoss Cache 与应用程序部署在同一服务器上。

14. Memcached 采用的是集中式的缓存管理，缓存与应用分离部署，缓存系统部署在专门的集群上，应用程序通过一致性 hash 等路由算法选择缓存服务器远程访问缓存数据，缓存服务器之间不通信。这样缓存集群可以很简单的实现扩容，具备良好的可伸缩性。采用的是 TCP 协议（UDP 也支持）通信，序列化协议是通过基于文本的自定义协议。服务端和客户端，采用 memcached 协议交互。
15. ①负载均衡+session 复制，将 session 同步至每个应用服务器，保证服务的状态。②session 绑定，利用负载均衡的源地址 hash 算法实现将来源于同一 IP 的请求始终分发到同一台应用服务器上。③利用浏览器的 cookie 记录下 session 以及 sessionID，将 session 以及 sessionID 发送给负载均衡器，负载均衡服务器根据 sessionID 将请求转发至相应的应用服务器。④设置专门的 session 服务器统一管理 session，应用程序每次读写 session 都通过 session 服务器。
16. 负载均衡算法：轮询：请求依次分发到每台应用服务器上；加权轮询：根据应用服务器的性能进行加权重新分配；随机；最少连接：记录每个应用服务器正在处理的请求连接数，将新的请求分配到连接数最少的服务器上；源地址散列。

# 综合问题

1. Runnable 接口和 Callable 接口的区别？
   Runnable 接口中的 run()方法的返回值是 void，它做的事情只是纯粹地去执行 run()方法中的代码而已；Callable 接口中的 call()方法是有返回值的，是一个泛型，和 Future、FutureTask 配合可以用来获取异步执行的结果。
2. CyclicBarrier 和 CountDownLatch 的区别？
   （1）CyclicBarrier 的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch 则不是，某线程运行到某个点上之后，只是给某个数值-1 而已，该线程继续运行
   （2）CyclicBarrier 只能唤起一个任务，CountDownLatch 可以唤起多个任务
   （3）CyclicBarrier 可重用，CountDownLatch 不可重用，计数值为 0 该 CountDownLatch 就不可再用了
3. volatile 关键字的作用？保证多线程数据的可见性。每次读到 volatile 修饰的数据的值都是最新的。禁止了语义重排序。还有一个作用就是和 CAS 一起保证了原子性。
4. 什么是线程安全？如果你的代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么你的代码就是线程安全的。线程安全还分为几个级别：不可变；绝对线程安全；相对线程安全（我们所说的线程安全一般指的是这个）；线程非安全。
5. Java 中如何获取到线程 dump 文件？可以设置虚拟机参数；可以使用 jstack 命令；还可以使用 Thread 类的 getStackTrace()方法。
6. 生产者消费者模型的作用是什么？提升系统的运行效率；解耦。
7. 怎么检测一个线程是否持有对象监视器？Thread 类提供了一个 holdsLock(Object obj)方法，当且仅当对象 obj 的监视器被某条线程持有的时候才会返回 true，注意这是一个 static 方法，这意味着“某条线程”指的是当前线程。
8. ConcurrentHashMap 的并发度是什么？ConcurrentHashMap 的并发度就是 segment 的大小，默认为 16，这意味着最多同时可以有 16 条线程操作 ConcurrentHashMap，这也是 ConcurrentHashMap 对 Hashtable 的最大优势。
9. FutureTask 是什么？FutureTask 表示一个异步运算的任务。FutureTask 里面可以传入一个

Callable 的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于 FutureTask 也是 Runnable 接口的实现类，所以 FutureTask 也可以放入线程池中。

10. Linux 环境下如何查找哪个线程使用 CPU 最长？（1）获取项目的 pid，jps 或者 ps -ef | grep java；（2）top -H -p pid，顺序不能改变。
11. Java 中用到的线程调度算法是什么？抢占式。一个线程用完 CPU 之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。
12. Thread.sleep(0)的作用是什么？Thread.sleep(0)手动触发一次操作系统分配时间片的操作，这也是平衡 CPU 控制权的一种操作。
13. 什么是 Java 内存模型？Java 内存模型将内存分为了主内存和工作内存。类的状态，也就是类之间共享的变量，是存储在主内存中的，每次 Java 线程用到这些主内存中的变量的时候，会读一次主内存中的变量，并让这些内存在自己的工作内存中有一份拷贝，运行自己线程代码的时候，用到这些变量，操作的都是自己工作内存中的那一份。在线程代码执行完毕之后，会将最新的值更新到主内存中去。
14. happens-before，即先行发生原则，定义了操作 A 必然先行发生于操作 B 的一些规则，比如在同一个线程内控制流前面的代码一定先行发生于控制流后面的代码、一个释放锁 unlock 的动作一定先行发生于后面对于同一个锁进行锁定 lock 的动作等等，只要符合这些规则，则不需要额外做同步措施，如果某段代码不符合所有的 happens-before 规则，则这段代码一定是线程非安全的。
15. 什么是 CAS？假设有三个操作数：内存值 V、旧的预期值 A、要修改的值 B，当且仅当预期值 A 和内存值 V 相同时，才会将内存值修改为 B 并返回 true，否则什么都不做并返回 false。当然 CAS 一定要 volatile 变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值 A 对某条线程来说，永远是一个不会变的值 A，只要某次 CAS 操作失败，永远都不可能成功。
16. 什么是 AQS？AQS 全称为 AbstractQueuedSychronizer，翻译过来应该是抽象队列同步器。如果说 java.util.concurrent 的基础是 CAS 的话，那么 AQS 就是整个 Java 并发包的核心了，ReentrantLock、CountDownLatch、Semaphore 等等都用到了它。AQS 实际上以双向队列的形式连接所有的 Entry，比方说 ReentrantLock，所有等待的线程都被放在一个 Entry 中并连成双向队列，前面一个线程使用 ReentrantLock 好了，则双向队列实际上的第一个 Entry 开始运行。
17. Hashtable 的 size()方法中明明只有一条语句”return count”，为什么还要做同步？（1）同一时间只能有一条线程执行固定类的同步方法，但是对于类的非同步方法，可以多条线程同时访问。所以，这样就有问题了，可能线程 A 在执行 Hashtable 的 put 方法添加数据，线程 B 则可以正常调用 size()方法读取 Hashtable 中当前元素的个数，那读取到的值可能不是最新的，可能线程 A 添加了完了数据，但是没有对 size++，线程 B 就已经读取 size 了，那么对于线程 B 来说读取到的 size 一定是不准确的。而给 size()方法加了同步之后，意味着线程 B 调用 size()方法只有在线程 A 调用 put 方法完毕之后才可以调用，这样就保证了线程安全性。（2）CPU 执行代码，执行的不是 Java 代码，这点很关键，一定得记住。Java 代码最终是被翻译成汇编代码执行的，汇编代码才是真正可以和硬件电路交互的代码。即使你看到 Java 代码只有一行，甚至你看到 Java 代码编译之后生成的字节码也只有一行，也不意味着对于底层来说这句语句的操作只有一个。一句”return count”假设被翻译成了三句汇编语句执行，完全可能执行完第一句，线程就切换了。
18. 线程类的构造方法、静态块是被哪个线程调用的？线程类的构造方法、静态块是被 new

这个线程类所在的线程所调用的，而 run 方法里面的代码才是被线程自身所调用的。

19. 高并发、任务执行时间短的业务怎样使用线程池？并发不高、任务执行时间长的业务怎样使用线程池？并发高、业务执行时间长的业务怎样使用线程池？

（1）高并发、任务执行时间短的业务，线程池线程数可以设置为 CPU 核数+1，减少线程上下文的切换

（2）并发不高、任务执行时间长的业务要区分开看：a）假如是业务时间长集中在 IO 操作上，也就是 IO 密集型的任务，因为 IO 操作并不占用 CPU，所以不要让所有的 CPU 闲下来，可以加大线程池中的线程数目，让 CPU 处理更多的业务 b）假如是业务时间长集中在计算操作上，也就是计算密集型任务，这个就没办法了，和（1）一样吧，线程池中的线程数设置得少一些，减少线程上下文的切换

（3）并发高、业务执行时间长，解决这种类型任务的关键不在于线程池而在于整体架构的设计，看看这些业务里面某些数据是否能做缓存是第一步，增加服务器是第二步，至于线程池的设置，设置参考（2）。最后，业务执行时间长的问题，也可能需要分析一下，看看能不能使用中间件对任务进行拆分和解耦。

20. 数据库瓶颈？分表分库；读写分离；主从库；SQL 优化；分布式缓存等。
21. JDBC 实现事务？setAutoCommit();commit();rollback()
22. 偏向锁，轻量级锁，重量级锁？

偏向锁：对象头 Mark word 储存线程 ID，线程进入和退出不需要 CAS 操作以及加锁和解锁。一般用于单线程。

轻量级锁：创建锁记录存储空间，复制对象头 mark word 到锁记录，线程要加锁阻塞时，会自旋以等待同步代码解锁。自旋时间及次数可以设置，一般适用于同步代码执行迅速的情况。存在锁竞争时，会膨胀成重量级锁。通过自旋，线程不阻塞，响应速度较快，但是自旋消耗 CPU。

重量级锁：线程阻塞，响应时间较慢。

23. loadClass()和 Class.forName()区别：Class.forName()加载的类会初始化，而 loadClass 加载的不会初始化。
24. JDBC 核心接口：DataSource，Connection，Statement，ResultSet。
25. ArryList 实现 list，RandomAccess，Cloneable，Serializable 接口。RandomAccess 是一个标记接口，用来表示其支持快速访问。用数组来实现。定义的数组使用 transient 关键字，不需要序列化与反序列化，而是自身实现了底层的存储。主要是因为数组定义是定长的，如果没有全部使用，那么序列化就会浪费内存容量。默认数组大小 10，每次扩容 1.5 倍。扩容会导致数组拷贝，非常浪费性能。写会扩容，删不会缩容。ArrayList 适用于元素查找与更新，不适用于元素插入与删除。LinkedList 基于双向循环列表，适用于元素的插入与删除。
26. 同步？异步？阻塞？非阻塞？线程安全？线程不安全？同步和异步关注的是消息通信机制；阻塞和非阻塞关注的是程序在等待调用结果（消息，返回值）时的状态。

举个通俗的例子：

你打电话问书店老板有没有《分布式系统》这本书，如果是同步通信机制，书店老板会说，你稍等，”我查一下"，然后开始查啊查，等查好了（可能是 5 秒，也可能是一天）告诉你结果（返回结果）。

而异步通信机制，书店老板直接告诉你我查一下啊，查好了打电话给你，然后直接挂电话了（不返回结果）。然后查好了，他会主动打电话给你。在这里老板通过“回电”这种方式来回调。

你打电话问书店老板有没有《分布式系统》这本书，你如果是阻塞式调用，你会一

直把自己“挂起”，直到得到这本书有没有的结果；

如果是非阻塞式调用，你不管老板有没有告诉你，你自己先一边去玩了，当然你也要偶尔过几分钟 check 一下老板有没有返回结果。

在这里阻塞与非阻塞与是否同步异步无关。跟老板通过什么方式回答你结果无关。