1.接口和抽象类的区别

①抽象类里可以有构造方法，而接口内不能有构造方法。

②抽象类中可以有普通成员变量，而接口中不能有普通成员变量。

③抽象类中可以包含非抽象的普通方法，而接口中所有的方法必须是抽象的，不能有非抽象的普通方法。

④抽象类中的抽象方法的访问类型可以是 public ，protected 和默认类型，但接口中的抽象方法只有 public 和默认类型。

⑤ 抽象类中可以包含静态方法，接口内不能包含静态方法。

⑥抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static 类型，并且默认为 public static 类型。

⑦一个类可以实现多个接口，但只能继承一个抽象类。

⑧接口更多的是在系统框架设计方法发挥作用，主要定义模块之间的通信，而抽象类在代码实现方面发挥作用，可以实现代码的重用。

2.Java 虚拟机的运行时数据区有几块？线程私有和线程共享区域有哪些？

①程序计数器：线程私有，当前县城执行的字节码的行号指示器。

②虚拟机栈：线程私有，存放基本数据类型、对象引用和 returnAddress 类型。

③本地方法栈：为虚拟机使用到的 Native 方法服务。

④Java 堆：线程共享，存放对象的实例，也是 GC 回收器管理的主要区域。

⑤方法区：线程共享，存放已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码等数据。

⑥运行时常量池：方法区的一部分，存放编译期生成的各种字面量和符号引用。

⑦直接内存：不是虚拟机运行时数据区的一部分，也不是 Java 虚拟机规范中定义的内存区域，容易引起 OOM 异常，NIO 会调用，不受 Java 堆大小的限制。

3.HashMap 和 HashTable 区别？

①Hashtable 是基于陈旧的 Dictionary 类的，HashMap 是 Java 1.2 引进的 Map 接口的一个实现。

②Hashtable 的方法是同步的，而 HashMap 的方法不是，因此

HashTable 是线程安全的，但是代码的执行效率上要慢于 HashMap。

③HashMap 允许空值和空键，但是 HashTable 不可以。

④HashMap 非同步实现 Map 接口，是一个 “链表数组” 的数据结构，最大承载量是 16，可以自动变长，由 Entry[]控制（key，value，next），hashCode() 判断 key 是否重复。

⑤建议需要做同步，使用 ConcurrentHashMap，降低了锁的粒度。在 hashMap 的基础上，ConcurrentHashMap 将数据分为多个 segment，默认 16 个（concurrency level），然后每次操作对一个 segment 加锁，避免多线程锁得几率，提高并发效率。这里在并发读取时，除了 key 对应的 value 为 null 之外，并没有使用锁。

4.ArrayList 和 LinkedList 区别？

ArrayList 基于数组实现，LinkedList 基于链表实现，ArrayList 增加和删除比 LinkedList 慢，但是 LinkedList 在查找的时需要递归查找，效率比 ArrayList 慢。关于多线程方面，如果要求线程安全的，有一个 Vector，不过比较多的使用的是 CopyOnWriteArrayList 替代 ArrayList，CopyOnWriteArrayList 适合使用在读操作远远大于写操作的场景里，比如缓存。发生修改时候做 copy，新老版本分离，保证读的高性能，适用于以读为主的情况。

5.Set 接口

①HashSet 是 Set 接口的典型实现，HashSet 按 hash 算法来存储元素，因此具有很好的存取和查找性能。特点：不能保证元素的排列顺序，顺序有可能发生变化；HashSet 是异步的；集合元素值可以是 null；当向 HashSet 集合中存入一个元素时，HashSet 会调用该对象的 hashCode（）方法来得到该对象的 hashCode 值，然后根据该 HashCode 值来确定该对象在 HashSet 中存储的位置。HashSet 还有一个子类 LinkedHashSet，其集合也是根据元素 hashCode 值来决定元素的存储位置，但它同时用链表来维护元素的次序，这样使得元素看起来是以插入的顺序保存的，也就是说，当遍历 LinkedHashSet 集合元素时，它将会按元素的添加顺序来访问集合里的元素。所以 LinkedHashSet 的性能略低于 HashSet，但在迭代访问全部元素时将有很好的性能，因为它以链表来维护内部顺序。

②TreeSet 是 SortSet 接口的唯一实现，TreeSet 可以确保集合元素处于排序状态。TreeSet 不是根据元素插入顺序进行排序的，而是根据元素的值来排序的。TreeSet 支持两种排序方法：自然排序和定制排序。

③EnumSet 中所有值都必须是指定枚举类型的值，它的元素也是有序的，以枚举值在枚举类的定义顺序来决定集合元素的顺序。EnumSet 集合不允许加入 null 元素，否则会抛出 NullPointerException 异常。EnumSet 类没有暴露任何构造器来创建该类的实例，程序应该通过它

提供的 static 方法来创建 EnumSet 对象。

④总结：

A、HashSet 的性能比 Treeset 好，因为 TreeSet 需要额外的红黑树算法来维护集合元素的次序，只有当需要一个保持排序的 Set 时，才会用 TreeSet。

B、EnumSet 是性能最好的，但它只能保存枚举值。

C、它们都是线程不安全的。

注：Set 是一种不包含重复的元素的 Collection，即任意的两个元素 e1 和 e2 都有 e1.equals(e2)=false，Set 最多有一个 null 元素。

关于 HashSet，条目数和容量之和来讲，迭代是线性的。因此，如果迭代性能很重要，那就应该慎重选择一个适当的初始容量。容量选得太大，既浪费空间，也浪费时间。默认的初试容量是 101，一般来讲，它比你所需要的要多。可以使用 int 构造函数来指定初始容量。要分配 HashSet 的初始容量为 17: Set s=new HashSet(17);

HashSet 另有一个称作装载因数（load factor）的"调整参数（tuning parameter）"。

区别: 1. HashSet 是通过 HashMap 实现的,TreeSet 是通过 TreeMap 实现的,只不过 Set 用的只是 Map 的 key。 2. Map 的 key 和 Set 都有一个共同的特性就是集合的唯一性.TreeMap 更是多了一个排序的功能. 3. hashCode 和 equal()是 HashMap 用的, 因为无需排序所以只需

要关注定位和唯一性即可.

a. hashCode 是用来计算 hash 值的,hash 值是用来确定 hash 表索引的.

b. hash 表中的一个索引处存放的是一张链表, 所以还要通过 equal 方法循环比较链上的每一个对象 才可以真正定位到键值对应的 Entry.

c. put 时,如果 hash 表中没定位到,就在链表前加一个 Entry,如果定位到了,则更换 Entry 中的 value,并返回旧 value

4. 由于 TreeMap 需要排序,所以需要一个 Comparator 为键值进行大小比较.当然也是用 Comparator 定位的.

a. Comparator 可以在创建 TreeMap 时指定

b. 如果创建时没有确定,那么就会使用 key.compareTo()方法,这就要求 key 必须实现 Comparable 接口.

TreeMap 是使用 Tree 数据结构实现的,所以使用 compare 接口就可以完成定位了.

6.Cookie Session 区别

具体来说 cookie 机制采用的是在客户端保持状态的方案,而 session 机制采用的是在服务器端保持状态的方案.同时我们也看到,由于采用服务器端保持状态的方案在客户端也需要保存一个标识,所以 session 机制可能需要借助于 cookie 机制来达到保存标识的目的,但实际上它

还有其他选择.

cookie 机制.正统的 cookie 分发是通过扩展 HTTP 协议来实现的,服务器通过在 HTTP 的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的 cookie.然而纯粹的客户端脚本如 JavaScript 或者 VBScript 也可以生成 cookie.而 cookie 的使用是由浏览器按照一定的原则在后台自动发送给服务器的.浏览器检查所有存储的 cookie,如果某个 cookie 所声明的作用范围大于等于将要请求的资源所在的位置,则把该 cookie 附在请求资源的 HTTP 请求头上发送给服务器.

cookie 的内容主要包括：名字,值,过期时间,路径和域.路径与域一起构成 cookie 的作用范围.若不设置过期时间,则表示这个 cookie 的生命期为浏览器会话期间,关闭浏览器窗口,cookie 就消失.这种生命期为浏览器会话期的 cookie 被称为会话 cookie.会话 cookie 一般不存储在硬盘上而是保存在内存里,当然这种行为并不是规范规定的.若设置了过期时间,浏览器就会把 cookie 保存到硬盘上,关闭后再次打开浏览器,这些 cookie 仍然有效直到超过设定的过期时间.存储在硬盘上的 cookie 可以在不同的浏览器进程间共享,比如两个 IE 窗口.而对于保存在内存里的 cookie,不同的浏览器有不同的处理方式

session 机制.session 机制是一种服务器端的机制,服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息.

当程序需要为某个客户端的请求创建一个 session 时,服务器首先检查这个客户端的请求里是否已包含了一个 session 标识（称为 session id）,如果已包含则说明以前已经为此客户端创建过 session,服务器就

按照 session id 把这个 session 检索出来使用(检索不到,会新建一个),如果客户端请求不包含 sessionid,则为此客户端创建一个 session 并且生成一个与此 session 相关联的 session id,session id 的值应该是一个既不会重复,又不容易被找到规律以仿造的字符串,这个 session id 将被在本次响应中返回给客户端保存.

保存这个 sessionid 的方式可以采用 cookie,这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器.一般这个 cookie 的名字都是类似于 SEEESIONID.但 cookie 可以被人为的禁止,则必须有其他机制以便在 cookie 被禁止时仍然能够把 session id 传递回服务器.

经常被使用的一种技术叫做 URL 重写,就是把 session id 直接附加在 URL 路径的后面.还有一种技术叫做表单隐藏字段.就是服务器会自动修改表单,添加一个隐藏字段,以便在表单提交时能够把 session id 传递回服务器.比如：实际上这种技术可以简单的用对 action 应用 URL 重写来代替.

7、Java IO 和 NIO 区别

①NIO 操作直接缓存区，直接与 OS 交互，Selector IO 复用机制。

IO	NIO
面向流	面向缓冲
阻塞 IO	非阻塞 IO
无	选择器

Selector：Java NIO 的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来 “选择” 通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

②NIO 与 Netty：A、NIO 的类库和 API 复杂，使用麻烦，需要熟练使用 Selector、ServerSocketChannel、SOcketChannel、ByteBuffer 等。B、NIO 涉及到 Reactor 模式，需要了解 Java 多线程和网络编程。C、JDKNIO Bug-epoll bug 容易导致 Selector 空轮询，最终导致 CPU100% 占用，虽然 JDK1.6 update18 修复了这个问题，但是直到 JDK1.7 问题依然存在，只是降低了发生的概率。

③Netty 的优点：A、API 简单，开发门槛低；B、功能强大，预置了多种解码功能，支持多种主流协议；C、可以通过 ChannelHandler 对通信框架进行灵活的扩展；D、性能高，Netty 的综合性能是最好的；E、Netty 修复了一经发现了所有的 JDKNIO BUG，成熟，稳定。

同步和异步的概念描述的是用户线程与内核的交互方式：同步是指用户线程发起 IO 请求后需要等待或者轮询内核 IO 操作完成后才能继续执行；而异步是指用户线程发起 IO 请求后仍继续执行，当内核 IO 操作完成后会通知用户线程，或者调用用户线程注册的回调函数。

引申：

Java 中 IO 的种类和应用场景：

A、同步阻塞式：BIO。用于连接数目较小且固定的架构，对服务器资

源占用高。

B、伪异步 IO 变成：线程池和任务队列。

C、NIO 编程：a、缓冲徐 ByteBuffer；b、通道 channel 全双工，同时用于读写；c、多路复用器 selector。用于连接数目多且较短的架构，如聊天服务器等，但是编程复杂，存在 epoll bug，导致 Selector 空轮询，直至 CPU 占用达到 100%，虽然在 JDK1.6 update18 中有对这个 bug 的修复，但是在 JDK1.7 中依然可能会出现这个问题，只是降低了 bug 出现的概率。

D、AIO 编程：用于连接数目多且较长的架构，如相册服务器等，充分调用 OS 参与并发操作，基于 JDK1.7。

阻塞和非阻塞的概念描述的是用户线程调用内核 IO 操作的方式：阻塞是指 IO 操作需要彻底完成后才返回到用户空间；而非阻塞是指 IO 操作被调用后立即返回给用户一个状态值，无需等到 IO 操作彻底完成。

8、Java 锁机制

①synchronized：把代码块声明为 synchronized，有两个重要后果，通常是指该代码具有　原子性和可见性。作用：A、当两个并发线程访问同一个对象 object 中的这个 synchronized(this)同步代码块时，一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。B、当一个线程访问 object 的一个 synchronized(this)同步代码块时，另一个线程仍然可以访问该

object 中的非 synchronized(this)同步代码块。C、尤其关键的是，当一个线程访问 object 的一个 synchronized(this)同步代码块时，其他线程对 object 中所有其它 synchronized(this)同步代码块的访问将被阻塞。

A、原子性：原子性意味着个时刻，只有一个线程能够执行一段代码，这段代码通过一个 monitor object 保护。从而防止多个线程在更新共享状态时相互冲突。

B、可见性：可见性则更为微妙，它要对付内存缓存和编译器优化的各种反常行为。它必须确保释放锁之前对共享数据做出的更改对于随后获得该锁的另一个线程是可见的。

C、volatile 只保证可见性和禁止重排序，不保证原子性。

②synchronized 限制：

A.它无法中断一个正在等候获得锁的线程；

B.也无法通过投票得到锁，如果不想等下去，也就没法得到锁；

C.同步还要求锁的释放只能在与获得锁所在的堆栈帧相同的堆栈帧中进行，多数情况下，这没问题（而且与异常处理交互得很好），但是，确实存在一些非块结构的锁定更合适的情况。

③java.util.concurrent.lock：

ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义，但是添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。此外，它还提供了在激烈争用情况下更佳的性能。

用 sychronized 修饰的方法或者语句块在代码执行完之后锁自动释放，

而是用 Lock 需要我们手动释放锁，所以为了保证锁最终被释放(发生异常情况)，要把互斥区放在 try 内，释放锁放在 finally 内。

④ReentrantWriteReadLock 中的 ReadLock 和 WWriteLock，在全为读时实现并发读，并发读写或并发写时候加锁。

总结：synchronized 是 Java 原语，阻塞的，竞争锁机制；新锁更加面向对象，并且支持中断和支持公平锁。

9、设计模式的理解和原则

A、开闭原则（Open Close Principle）

开闭原则就是说对扩展开放，对修改关闭。在程序需要进行拓展的时候，不能去修改原有的代码，实现一个热插拔的效果。所以一句话概括就是：为了使程序的扩展性好，易于维护和升级。想要达到这样的效果，我们需要使用接口和抽象类，后面的具体设计中我们会提到这点。

B、里氏代换原则（Liskov Substitution Principle）

里氏代换原则(Liskov Substitution Principle LSP)面向对象设计的基本原则之一。 里氏代换原则中说，任何基类可以出现的地方，子类一定可以出现。 LSP 是继承复用的基石，只有当衍生类可以替换掉基类，软件单位的功能不受到影响时，基类才能真正被复用，而衍生类也能够在基类的基础上增加新的行为。里氏代换原则是对 “开-闭” 原则的补充。实现 “开-闭” 原则的关键步骤就是抽象化。而基类与

子类的继承关系就是抽象化的具体实现，所以里氏代换原则是对实现抽象化的具体步骤的规范。—— From Baidu 百科

C、依赖倒转原则（Dependence Inversion Principle）

这个是开闭原则的基础，具体内容：真对接口编程，依赖于抽象而不依赖于具体。

D、接口隔离原则（Interface Segregation Principle）

这个原则的意思是：使用多个隔离的接口，比使用单个接口要好。还是一个降低类之间的耦合度的意思，从这儿我们看出，其实设计模式就是一个软件的设计思想，从大型软件架构出发，为了升级和维护方便。所以上文中多次出现：降低依赖，降低耦合。

F、迪米特法则（最少知道原则）（Demeter Principle）

为什么叫最少知道原则，就是说：一个实体应当尽量少的与其他实体之间发生相互作用，使得系统功能模块相对独立。

F、合成复用原则（Composite Reuse Principle）

原则是尽量使用合成/聚合的方式，而不是使用继承。

10、Java 反射

反射机制指的是程序在运行时能够获取自身的信息。

为什么要用反射机制？直接创建对象不就可以了吗，这就涉及到了动态与静态的概念，

静态编译：在编译时确定类型，绑定对象,即通过。

动态编译：运行时确定类型，绑定对象。动态编译最大限度发挥了 java 的灵活性，体现了多态的应用，有以降低类之间的藕合性。

一句话，反射机制的优点就是可以实现动态创建对象和编译，体现出很大的灵活性，特别是在 J2EE 的开发中
它的灵活性就表现的十分明显。

作用：①首先得根据传入的类的全名来创建 Class 对象。 ②获得类方法的方法。③　获得类中属性的方法。

缺点：①性能第一：反射包括了一些动态类型，所以 JVM 无法对这些代码进行优化。因此，反射操作的效率要比那些非反射操作低得多。我们应该避免在经常被 执行的代码或对性能要求很高的程序中使用反射。②安全限制：使用反射技术要求程序必须在一个没有安全限制的环境中运行。如果一个程序必须在有安全限制的环境中运行，如 Applet。③内部暴露：由于反射允许代码执行一些在正常情况下不被允许的操作（比如访问私有的属性和方法），所以使用反射可能会导致意料之外的副作用——代码有功能上的错误，降低可移植性。反射代码破坏了抽象性，因此当平台发生改变的时候，代码的行为就有可能也随着变化。

11、线程、线程池：

①创建线程有 4 方式：继承 Thread+实现 Runnable+实现 Callable+线程池获得

②wait 和 sleep 比较：sleep 方法有：sleep(long millis)，sleep(long millis, long nanos)，调用 sleep 方法后，当前线程进入休眠期，暂停执行，但该线程继续拥有监视资源的所有权。到达休眠时间后线程将继续执行，直到完成。若在休眠期另一线程中断该线程，则该线程退出。等待有其它的线程调用 notify()或 notifyAll()进入调度状态，与其它线程共同争夺监视。

③线程池：多线程技术主要解决处理器单元内多个线程执行的问题，它可以显著减少处理器单元的闲置时间，增加处理器单元的吞吐能力。

一个线程池包括以下四个基本组成部分：

A、线程池管理器（ThreadPool）：用于创建并管理线程池，包括创建线程池，销毁线程池，添加新任务；

B、工作线程（PoolWorker）：线程池中线程，在没有任务时处于等待状态，可以循环的执行任务；

C、任务接口（Task）：每个任务必须实现的接口，以供工作线程调度任务的执行，它主要规定了任务的入口，任务执行完后的收尾工作，任务的执行状态等；

D、任务队列（taskQueue）：用于存放没有处理的任务。提供一种缓冲机制。

④线程池分类：

A、newFixedThreadPool　创建一个指定工作线程数量的线程池。

每当提交一个任务就创建一个工作线程，如果工作线程数量达到线程池初始的最大数，则将提交的任务存入到池队列中。

B、newCachedThreadPool 创建一个可缓存的线程池。

这种类型的线程池特点是：

1).工作线程的创建数量几乎没有限制(其实也有限制的,数目为 Interger.MAX_VALUE), 这样可灵活的往线程池中添加线程。

2).如果长时间没有往线程池中提交任务，即如果工作线程空闲了指定的时间(默认为 1 分钟)，则该工作线程将自动终止。终止后，如果你又提交了新的任务，则线程池重新创建一个工作线程。

C、newSingleThreadExecutor 创建一个单线程化的 Executor，即只创建唯一的工作者线程来执行任务，如果这个线程异常结束，会有另一个取代它，保证顺序执行(我觉得这点是它的特色)。

单工作线程最大的特点是可保证顺序地执行各个任务，并且在任意给定的时间不会有多个线程是活动的。

D、newScheduleThreadPool 创建一个定长的线程池，而且支持定时的以及周期性的任务执行，类似于 Timer。

⑤Executors 类，提供了一系列静态工厂方法用于创先线程池，返回的线程池都实现了 ExecutorService 接口。

⑥线程池参数：

A、corePoolSize（线程池的基本大小）

B、runnableTaskQueue（任务队列）：用于保存等待执行的任务的阻塞队列。

1）LinkedBlockingQueue：一个基于链表结构的阻塞队列，此队列按 FIFO（先进先出）排序元素，吞吐量通常要高于 ArrayBlockingQueue。

静态工厂方法 Executors.newFixedThreadPool()使用了这个队列。

2）SynchronousQueue：一个不存储元素的阻塞队列。每个插入操作必须等到另一个线程调用移除操作，否则插入操作一直处于阻塞状态，吞吐量通常要高于 LinkedBlockingQueue，静态工厂方法 Executors.newCachedThreadPool 使用了这个队列。

3）PriorityBlockingQueue：一个具有优先级的无限阻塞队列。

C、maximumPoolSize（线程池最大大小）：线程池允许创建的最大线程数。

D、ThreadFactory：用于设置创建线程的工厂，可以通过线程工厂给每个创建出来的线程设置更有意义的名字。

E、RejectedExecutionHandler（饱和策略）：当队列和线程池都满了，说明线程池处于饱和状态，那么必须采取一种策略处理提交的新任务。这个策略默认情况下是 AbortPolicy，表示无法处理新任务时抛出异常。以下是 JDK1.5 提供的四种策略：

1）AbortPolicy：直接抛出异常。

2）CallerRunsPolicy：只用调用者所在线程来运行任务。

3)DiscardOldestPolicy：丢弃队列里最近的一个任务并执行当前任务。

4）DiscardPolicy：不处理，丢弃掉。

5)当然也可以根据应用场景需要来实现 RejectedExecutionHandler 接口自定义策略。如记录日志或持久化不能处理的任务。

F、keepAliveTime（线程活动保持时间）：线程池的工作线程空闲后，保持存活的时间。所以如果任务很多，并且每个任务执行的时间比较

短，可以调大这个时间，提高线程的利用率。

G、TimeUnit（线程活动保持时间的单位）：可选的单位有天（DAYS），小时（HOURS），分钟（MINUTES），毫秒(MILLISECONDS)，微秒(MICROSECONDS, 千分之一毫秒)和毫微秒(NANOSECONDS, 千分之一微秒)。

12、J2EE 是什么？谈谈你的理解

原来 sun 公司 13 中技术规格的综合，分别是：

(1)、JDBC（java Database Connectivity）：

JDBC API 为访问不同的数据库提供了一种统一的途径，就像 ODBC 一样，JDBC 对开发者屏蔽了一些细节问题，同时，JDBC 对数据库的访问也具有平台无关性。

(2)、JNDI(Java Name and Directory Interface):

JNDI API 被用于执行名字和目录服务。它提供了一致的模型用来存取和操作企业级的资源如 DNS 和 LDAP，本地文件系统，或应用服务器中的对象。

(3)、EJB（Enterprise JavaBean）：

J2ee 技术之所以赢得全体广泛重视的原因之一就是 EJB，他们提供了一个框架开发和实施分布式商务逻辑，由此很显著简化了具有可伸缩性和高度复杂的企业级应用开发。EJB 规范定义了 EJB 组件何时如何与他们的容器继续拧交互作用。容器负责提供公用的服务，例如目录

服务、事务管理、安全性、资源缓冲池以及容错性。但是注意的是，EJB 并不是 J2EE 的唯一途径。正是由于 EJB 的开放性，使得有的厂商能够以一种和 EJB 平行的方式来达到同样的目的。

(4)、RMI（RemoteMethod Invoke）：remote（遥远的） invoke（调用）：

正如其名字所表示的那样，RMI 协议调用远程对象上方法。它使用了序列化方式在客户端和服务器端传递数据。RMI 是一种被 EJB 使用的更底层的协议。

(5)、Java IDL（接口定义语言）/CORBA：公共对象请求代理结构（Common Object Request Breaker Architecture）：

在 java IDL 的支持下，开发人员可以将 Java 和 CORBA 集成在一起。他们可以创建 Java 对象并使之可以在 CORBA ORB 中展开，或者他们还可以创建 Java 类并做为和其他 ORB 一起展开的 CORBA 对象客户。后一种方法提供了另外一种途径，通过它可以被用于你的新的应用和旧系统相集成。

(6)、JSP(Java Server Pages):

Jsp 页面由 html 代码和嵌入其中的 Java 新代码所组成。服务器在页面被客户端所请求以后对这些 java 代码进行处理，然后将生成的 html 页面返回给客户端的浏览器。

(7)、Java Servlet：

servlet 是一种小型的 java 程序，它扩展了 web 服务器的功能。作为一种服务器端的应用，当被请求时开始执行，这和 CGI Perl 脚本很相

似。Servlet 提供的功能大多和 jsp 类似，不过实现方式不同。JSP 通过大多数的 html 代码中嵌入少量的 java 代码，而 servlet 全部由 java 写成并生成相应的 html。

(8)、XML（Extensible Markup Language）：

XML 是一种可以用来定义其他标记语言的语言。它被用来在不同的商务过程中共享数据。XML 的发展和 Java 是互相独立的，但是，它和 java 具有相同目标正是平台独立。通过 java 和 xml 的组合，我们可以得到一个完美的具有平台独立性的解决方案。

(9)、JMS（Java Message Service）：

Ms 是用于和面向消息的中间件相互通信的应用程序接口（API）。它既支持点对点的域，有支持发布/订阅类型的域，并且提供对下列类型的支持：经认可的消息传递，事务性消息传递，一致性消息和具有持久性的订阅者的支持。JMS 还提供了另一种方式对您的应用与旧的后台系统相集成。

(10)、JTA（Java Transaction Architecture）：

JTA 定义了一种标准 API，应用系统由此可以访问各种事务监控。

(11)、JTS（Java　Transaction Service）:

JTS 是 CORBA OTS 事务监控的基本实现。JTS 规定了事务管理器的实现方式。该事务管理器是在高层支持 Java Transaction API（JTA）规范，并且在较底层实现 OMG OTS specification 的 java 映像。JTS 事务管理器为应用服务器、资源管理器、独立的应用以及通信资源管理器提供了事务服务。

(12)、JavaMail：

JavaMail 是用于存取邮件服务的 API，它提供了一套邮件服务器的抽象类。不仅支持 SMTP 服务器，也支持 IMAP 服务器。

(13)、JAF（JavaBeans Activation Framework）：

JavaMail 利用 JAF 来处理 MIME 编码的邮件附件。MIME 的字节流可以被转换成 java 对象，或者转换自 Java 对象。大多数应用都可以不需要直接使用 JAF。

13、nginx 和 apache 的对比理解

①nginx 相对于 apache 的优点：

轻量级，同样起 web 服务，比 apache 占用更少的内存及资源，抗并发，nginx 处理请求是异步非阻塞的，而 apache 则是阻塞型的，在高并发下 nginx 能保持低资源低消耗高性能，高度模块化的设计，编写模块相对简单。

apache 相对于 nginx 的优点：A.rewrite，比 nginx 的 rewrite 强大；B.动态页面，模块超多，基本想到的都可以找到；C.少 bug，nginx 的 bug 相对较多；D.超稳定.

一般来说，需要性能的 web 服务，用 nginx 。如果不需要性能只求稳定，那就 apache.

②作为 Web 服务器：相比 Apache，Nginx 使用更少的资源，支持更多的并发连接，体现更高的效率。Nginx 采用 C 进行编写, 不论是系统资源开销还是 CPU 使用效率都比 Perlbal 要好很多.

③Nginx 配置简洁,Apache 复杂。Nginx 静态处理性能比 Apache 高 3 倍以上，Apache 对 PHP 支持比较简单，Nginx 需要配合其他后端用。Apache 的组件比 Nginx 多，现在 Nginx 才是 Web 服务器的首选。

④最核心的区别在于 apache 是同步多进程模型，一个连接对应一个进程；nginx 是异步的，多个连接（万级别）可以对应一个进程。

⑤nginx 处理静态文件好,耗费内存少.但无疑 apache 仍然是目前的主流,有很多丰富的特性.所以还需要搭配着来.当然如果能确定 nginx 就适合需求,那么使用 nginx 会是更经济的方式。

⑥nginx 处理动态请求是鸡肋，一般动态请求要 apache 去做，nginx 只适合静态和反向。

⑦Nginx 优于 apache 的主要两点：A.Nginx 本身就是一个反向代理服务器 B.Nginx 支持 7 层负载均衡；其他的当然，Nginx 可能会比 apache 支持更高的并发。

14、Struts2

①工作原理

A、在 Struts2 框架中的处理大概分为以下几个步骤：

1）客户端初始化一个指向 Servlet 容器（例如 Tomcat）的请求

2）这个请求经过一系列的过滤器（Filter）（这些过滤器中有一个叫做 ActionContextCleanUp 的可选过滤器，这个过滤器对于 Struts2 和其

他框架的集成很有帮助，例如：SiteMesh Plugin）

3）接着 FilterDispatcher 被调用，FilterDispatcher 询问 ActionMapper 来决定这个请是否需要调用某个 Action

4）如果 ActionMapper 决定需要调用某个 Action，FilterDispatcher 把请求的处理交给 ActionProxy

5）ActionProxy 通过 Configuration Manager 询问框架的配置文件，找到需要调用的 Action 类

6）ActionProxy 创建一个 ActionInvocation 的实例。

7）ActionInvocation 实例使用命名模式来调用，在调用 Action 的过程前后，涉及到相关拦截器（Intercepter）的调用。

8）一旦 Action 执行完毕，ActionInvocation 负责根据 struts.xml 中的配置找到对应的返回结果。返回结果通常是（但不总是，也可 能是另外的一个 Action 链）一个需要被表示的 JSP 或者 FreeMarker 的模版。在表示的过程中可以使用 Struts2 框架中继承的标签。在这个过程中需要涉及到 ActionMapper。

②工作流程：

1）客户端在浏览器中输入一个 url 地址。

2）这个 url 请求通过 http 协议发送给 tomcat。

3）tomcat 根据 url 找到对应项目里面的 web.xml 文件。

4）在 web.xml 里面会发现有 struts2 的配置。

5）然后会找到 struts2 对应的 struts.xml 配置文件。

6）根据 url 解析 struts.xml 配置文件就会找到对应的 class。

7）调用完 class 返回一个字 String，根据 struts.xml 返回到对应的 jsp。

15、memcached 和 Redis 区别，NOSQL 的理解

①Memcached 出现于 2003 年，key 支持 250bytes，value 支持 1MB；redis 出现于 2009 年，key 和 value 都是支持 512MB 的。

持久化方面，memcached 过期失效，redis 可以缓存回收（6 种回收机制），有存储，可算为 NOSQL，有优化，支持 190 多种命令。

集群方面，memcached 不支持集群，基于两次哈希，第一次哈希找到服务器节点，第二次哈希找到存储的值。

1）Redis 不仅仅支持简单的 k/v 类型的数据，同时还提供 list，set，hash 等数据结构的存储。

2）Redis 支持数据的备份，即 master-slave 模式的数据备份。

3）Redis 支持数据的持久化，可以将内存中的数据保持在磁盘中，重启的时候可以再次加载进行使用。

Redis 只会缓存所有的 key 的信息，如果 Redis 发现内存的使用量超过了某一个阀值，将触发 swap 的操作，Redis 根据 “swappability= age*log(size_in_memory)” 计算出哪些 key 对应的 value 需要 swap 到磁盘。然后再将这些 key 对应的 value 持久化到磁盘中，同时在内存中清除。这种特性使得 Redis 可以保持超过其机器本身内存大小的数据。当然，机器本身的内存必须要能够保持所有的 key，毕竟这些

数据是不会进行 swap 操作的。

②memcached、redis 和 MongoDB 三者之间的对比：

1）性能

都比较高，性能对我们来说应该都不是瓶颈体来讲，TPS（每秒事务处理量）方面 redis 和 memcache 差不多，要大于 mongod

2）操作的便利性

memcache 数据结构单一

redis 丰富一些，数据操作方面，redis 更好一些，较少的网络 IO 次数

mongodb 支持丰富的数据表达，索引，最类似关系型数据库，支持的查询语言非常丰富

3）内存空间的大小和数据量的大小

redis 在 2.0 版本后增加了自己的 VM 特性，突破物理内存的限制；可以对 key value 设置过期时间（类似 memcache）

memcache 可以修改最大可用内存,采用 LRU 算法

mongoDB 适合大数据量的存储，依赖操作系统 VM 做内存管理，吃内存也比较厉害，服务不要和别的服务在一起

4）可用性（单点问题）

对于单点问题，

redis，依赖客户端来实现分布式读写；主从复制时，每次从节点重新连接主节点都要依赖整个快照,无增量复制，因性能和效率问题，

所以单点问题比较复杂；不支持自动 sharding,需要依赖程序设定一致 hash 机制。

一种替代方案是，不用 redis 本身的复制机制，采用自己做主动复制（多份存储），或者改成增量复制的方式（需要自己实现），一致性问题和性能的权衡

Memcache 本身没有数据冗余机制，也没必要；对于故障预防，采用依赖成熟的 hash 或者环状的算法，解决单点故障引起的抖动问题。

mongoDB 支持 master-slave,replicaset（内部采用 paxos 选举算法，自动故障恢复）,auto sharding 机制，对客户端屏蔽了故障转移和切分机制。

5）可靠性（持久化）

对于数据持久化和数据恢复，

redis 支持（快照、AOF）：依赖快照进行持久化，aof 增强了可靠性的同时，对性能有所影响；memcache 不支持，通常用在做缓存,提升性能；MongoDB 从 1.8 版本开始采用 binlog 方式支持持久化的可靠性。

6）数据一致性（事务支持）

Memcache 在并发场景下，用 cas 保证一致性。

redis 事务支持比较弱，只能保证事务中的每个操作连续执行。

mongoDB 不支持事务

7）数据分析

mongoDB 内置了数据分析的功能(mapreduce),其他不支持。

8）应用场景

redis：数据量较小的更性能操作和运算上。

memcache：用于在动态系统中减少数据库负载，提升性能;做缓存，提高性能（适合读多写少，对于数据量比较大，可以采用 sharding）。

MongoDB:主要解决海量数据的访问效率问题。