J2SE 基础

1. 九种基本数据类型的大小，以及他们的封装类。

byte char boolean short int long float double

Byte Character Boolean Short Integer Long Float Double

2. Switch 能否用 string 做参数？

char int byte short 可以 java7 后可以使用 string 做参数

3. equals 与==的区别。

“==”是一个运算符，而 equals 是 object 里面的一个方法。

对于基本类型，在比较大小的时候可以使用“==”，看两个元素是否相等。Equals 没有用武之地。

对于对象之间的比较，“==”是比较两个对象的内存地址是否相等， equals 在 object 也是使用 “==”运算符实现的，所以不重写 equals 方法的类使用是一样的，但是我们一般会在子类中重写他，比较两个对象的内容是否一样。

4. Object 有哪些公用方法？

Object clone()

finalize()

equals(object o)

getClass(): return the run time class of this object

int hashCode():Returns a hash code value for the object.

String toString():object 的实现 getClass().getName() + "@" + Integer.toHexString(hashCode());

Notify() notifyAll()

Wait() wait(long n) wait(long l, int nami)//更精确的控制等待时间

5. Java 的四种引用，强弱软虚，用到的场景。

强

软 解决 oom

弱 解决 oom

虚：主要用来跟踪对象被垃圾回收的活动。虚引用必须和引用队列关联使用，当垃圾回收器准备回收一个对象时，如果发现它还有虚引用，就会把这个虚引用加入到与之 关联的引用队列中。程序可以通过判断引用队列中是否已经加入了虚引用，来了解被引用的对象是否将要被垃圾回收。如果程序发现某个虚引用已经被加入到引用队列，那么就可以在所引用的对象的内存被回收之前采取必要的行动。

6. Hashcode 的作用。

hashCode 的存在主要是用于查找的快捷性，如 Hashtable，HashMap 等，hashCode 是用来在散列存储结构中确定对象的存储地址的；

一般来说 两个 equals 的对象应该具有一样的 hashcode

7. ArrayList、LinkedList、Vector 的区别。

ArrayList 和 Vector 是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要设计到数组元素移动等内存操作，所以索引数据快插入数据慢，Vector 由于使用了 synchronized 方法（线程安全）所以性能上比 ArrayList 要差，LinkedList 使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快！

8. String、StringBuffer 与 StringBuilder 的区别。

String 字符串常量

StringBuffer 字符串变量（线程安全）字符串缓冲区对象

StringBuilder 字符串变量（非线程安全）

9. Map、Set、List、Queue、Stack 的特点与用法。

详细

Map：一组成对的“键值对”对象

Set：没有顺序 不重复（数学中的集合）

List：有顺序 可重复

Queue：PriorityQueue 这个是优先级队列

Stack 栈

10. HashMap 和 HashTable 的区别。

1.HashTable 的方法是同步的，HashMap 未经同步，所以在多线程场合要手动同步 HashMap 这个区别就像 Vector 和 ArrayList 一样。

2.HashTable 不允许 null 值(key 和 value 都不可以),HashMap 允许 null 值(key 和 value 都可以)。

3.HashTable 有一个 contains(Object value)，功能和 containsValue(Object value)功能一样。

4.HashTable 使用 Enumeration，HashMap 使用 Iterator。

5.HashTable 中 hash 数组默认大小是 11，增加的方式是 old*2+1。HashMap 中 hash 数组的默认大小是 16，而且一定是 2 的指数。

6.哈希值的使用不同，HashTable 直接使用对象的 hashCode，代码是这样的：

```
int hash = key.hashCode();
int index = (hash & 0x7FFFFFFF) % tab.length;
```

而 HashMap 重新计算 hash 值，而且用与代替求模：

```
int hash = hash(k);
int i = indexFor(hash, table.length);
```

11. HashMap 和 ConcurrentHashMap 的区别，HashMap 的底层源码。

HashMap 是线程不安全的

ConcurrentHashMap 是线程安全的

hashTable 也是线程安全的

区别：

从类图中可以看出来在存储结构中 ConcurrentHashMap 比 HashMap 多出了一个类 Segment，而 Segment 是一个可重入锁。

ConcurrentHashMap 是使用了锁分段技术技术来保证线程安全的。

锁分段技术：首先将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。

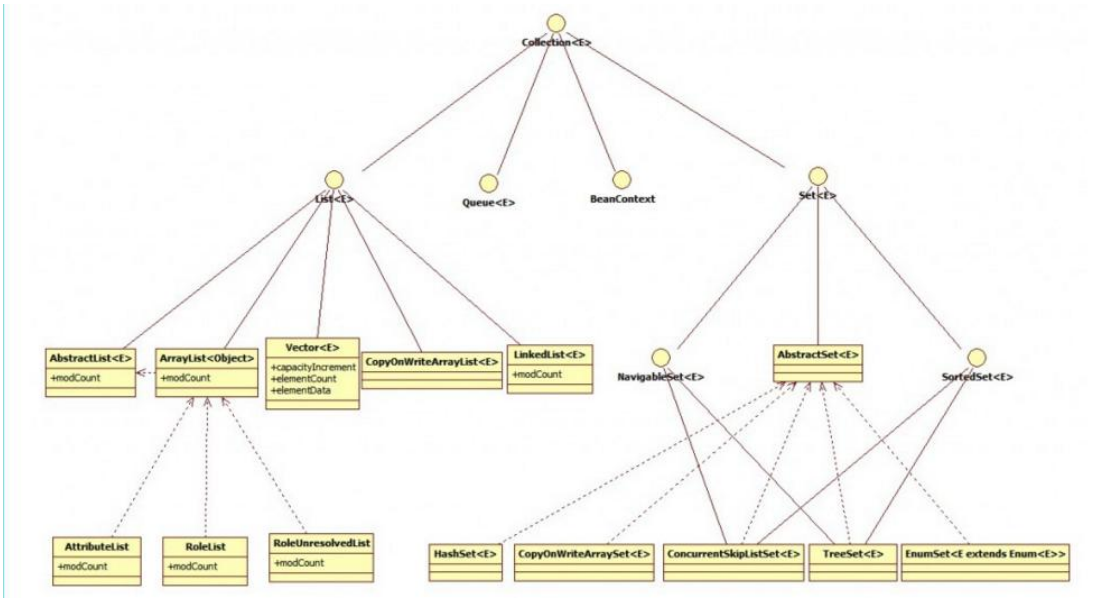
12. TreeMap、HashMap、LindedHashMap 的区别。

Hashmap 是一个最常用的 Map,它根据键的 HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。HashMap 最多只允许一条记录的键为 Null;允许多条记录的值为 Null;HashMap 不支持线程的同步，即任一时刻可以有多个线程同时写 HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections 的 synchronizedMap 方法使 HashMap 具有同步的能力，或者使用 ConcurrentHashMap。

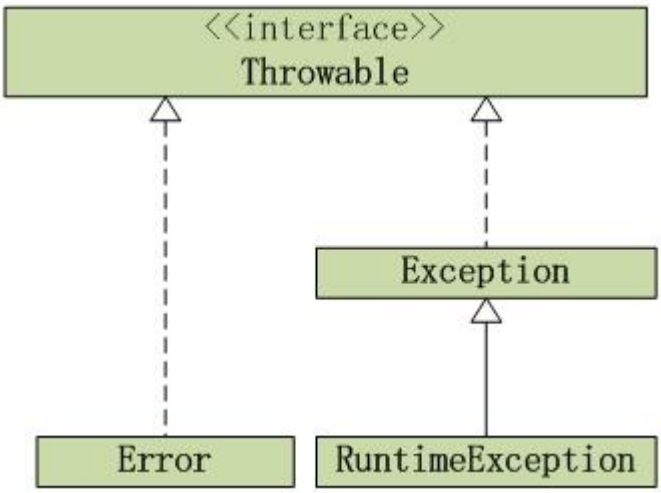
Hashtable 与 HashMap 类似,它继承自 Dictionary 类，不同的是:它不允许记录的键或者值为空;它支持线程的同步，即任一时刻只有一个线程能写 Hashtable,因此也导致了 Hashtable 在写入时会比较慢。

LinkedHashMap 保存了记录的插入顺序，在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比 HashMap 慢，不过有种情况例外，当 HashMap 容量很大，实际数据较少时，遍历起来可能

会比 LinkedHashMap 慢，因为 LinkedHashMap 的遍历速度只和实际数据有关，和容量无关，而 HashMap 的遍历速度和他的容量有关。
TreeMap 实现 SortMap 接口，能够把它保存的记录根据键排序,默认是按键值的升序排序，也可以指定排序的比较器，当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。
一般情况下，我们用的最多的是 HashMap,HashMap 里面存入的键值对在取出的时候是随机的,它根据键的 HashCode 值存储数据,根据键可以直接获取它的值，具有很快的访问速度。在 Map 中插入、删除和定位元素，HashMap 是最好的选择。
TreeMap 取出来的是排序后的键值对。但如果您要按自然顺序或自定义顺序遍历键，那么 TreeMap 会更好。
LinkedHashMap 是 HashMap 的一个子类，如果需要输出的顺序和输入的相同,那么用 LinkedHashMap 可以实现,它还可以按读取顺序来排列，像连接池中可以应用。
13. Collection 包结构，与 Collections 的区别。



Collection 是集合类的上级接口，子接口主要有 Set 和 List、Map。
Collections 是针对集合类的一个帮助类，提供了操作集合的工具方法：一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。
14. try catch finally，try 里有 return，finally 还执行么？执行
15. Excption 与 Error 包结构。OOM 你遇到过哪些情况，SOF 你遇到过哪些情况。



Thorwable 类所有异常和错误的超类，有两个子类 Error 和 Exception，分别表示错误和异常。其中异常类 Exception 又分为运行时异常(RuntimeException)和非运行时异常，这两种异常有很大的区别，也称之为不检查异常（Unchecked Exception）和检查异常（Checked Exception）。从责任这个角度看 Error 属于 JVM 需要负担的责任、RuntimeException 是程序应该负担的责任、checked exception 是具体应用负担的责任。

2.1 Error

Error 是程序无法处理的错误，比如 OutOfMemoryError、ThreadDeath 等，Java 虚拟机抛出一个 Error 对象，应用程序不捕获或抛出 Errors 对象，你可能永远不会遇到需要实例化 Error 的应用，这些异常发生时，Java 虚拟机（JVM）一般会选择线程终止。

2.2 Exception

Exception 是程序本身可以处理的异常，这种异常分两大类运行时异常和非运行时异常。程序中应当尽可能去处理这些异常。

2.2.1 Checked exception

这类异常都是 Exception 的子类。从程序语法角度讲是必须进行处理的异常，如果不处理，程序就不能编译通过，如 IOException、SQLException 等以及用户自定义的 Exception 异常，一般情况下不自定义检查异常。

一个 checked exception 强迫它的客户端可以抛出并捕获它，一旦客户端不能有效地处理这些被抛出的异常就会给程序的执行带来不期望的负担。

Checked exception 还可能带来封装泄漏，看下面的代码：

```
public List getAllAccounts() throws FileNotFoundException, SQLException{
...
}
```

上边的方法抛出两个异常，客户端必须显示的对这两种异常进行捕获和处理，即使是在完全不知道这种异常到底是因为文件还是数据库操作引起的情况下。因此，此时的异常处理将导致方法和调用者之间不合适的耦合。

2.2.2 Unchecked exception

这类异常都是 RuntimeException 的子类，虽然 RuntimeException 同样也是 Exception 的子类，但是它们是特殊的，它们不能通过 client code 来试图解决，所以称为 Unchecked exception，如 NullPointerException、IndexOutOfBoundsException 等，这些异常是不检查异常，程序中可以选择捕获处理，也可以不处理。这些异常一般是由程序逻辑错误引起的，程序应该从逻辑角度尽可能避免这类异常的发生。

16. Java 面向对象的三个特征与含义。

封装：即把对象的全部属性和全部服务结合在一起， 形成一个不可分割的独立单位；
尽可能隐藏对象的内部结构。

继承：在面向对象世界里面，常常要创建某对象（如：一个职员对象） ，然后需要一个该基本对象的更专业化的版本， 比如， 可能需要一个经理的对象。显然经理实际上是一个职员，经理和职员具有 is a 的关系，经理只是一个带有附加特征的职员（经理也是一个职员）。因此，需要有一种办法从现有对象来创建一个新对象。这个方式就是继承。

多态：父类引用指向子类对象 并且在运行时决定调用哪个方法

17. Override 和 Overload 的含义去区别。

Overload：顾名思义，就是 Over(重新)——load（加载），所以中文名称是重载。它可以表现类的多态性，可以是函数里面可以有相同的函数名但是参数名、返回值、类型不能相同；或者说可以改变参数、类型、返回值但是函数名字依然不变。

Override：就是 ride(重写)的意思，在子类继承父类的时候子类中可以定义某方法与其父类有相同的名称和参数，当子类在调用这一函数时自动调用子类的方法，而父类相当于被覆盖（重写）了。

18. Interface 与 abstract 类的区别。

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以有普通成员变量，接口中没有普通成员变量

3.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，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,虽然 eclipse 下不报错，但应该也不行），但接口中的抽象方法只能是 public 类型的，并且默认即为 public abstract 类型。

5. 抽象类中可以包含静态方法，接口中不能包含静态方法

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 final 类型，并且默认即为 public static final 类型。

7. 一个类可以实现多个接口，但只能继承一个抽象类。

接口更多的是在系统架构设计方法发挥作用，主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用

19. Static class 与 non static class 的区别。

static 应当（注意是应当）使用类名来引用。而 non-static 必须（是必须）使用对象实例名来引用。

因为 static、non-static 的数据相关性，static 只能引用类的 static 数据成员；而 non-static 既可以引用类的 static 数据成员，也可以引用对象自身的数据。

static 与 non-static method 在 overload 方面是一样的。

static 方法是与类相关的，不是通过 this 引用的，所以它不能被 override。其引用在编译期就得确定。而 non-static 方法才有可能被 override。

static 与 abstract，它们不能同时用于修饰一个方法。因为 abstract 的语义就是说这个方法是多态方法，需要 subclass 的实现。而 static 方法则是在本类中实现的，编译期绑定，不具有多态行为。

static 与 interface，interface 中的 method 也不能是 static 的。理由同上。但其数据成员 are all static, no matter you mark it static or not 。

多态只限于方法，所以，无论 static 还是 non-static 的成员变量，引用的是哪个在编译期就已经确定。

20. java 多态的实现原理。

参考虚机的静态分配和动态分配

21. 实现多线程的两种方法：

继承 Thread 类与实现 Runnable 接口。

然后 new Thread(Runnable r).start();开启一个子线程 //其中 thread 实现了 runnable 接口

或者 new Thread() {public void run() {} }.start();//匿名内部类

22. 线程同步的方法：sychronized、lock、reentrantLock 等。

23. 锁的等级：方法锁、对象锁、类锁。

24. 写出生产者消费者模式。

25. ThreadLocal（thread local variable）的设计理念与作用。
理念：Java 中的 ThreadLocal 类允许我们创建的 ThreadLocal 对象只能被同一个线程读写的变量。因此，如果一段代码中有一个 ThreadLocal 变量得引用，即使两个变量同时来执行这段代码，他们也无法访问到对方的 ThreadLocal 变量。
要是想初始化变量 则需要重写 initialValue()方法
作用：ThreadLocal 是隔离多个线程的数据共享
线程同步：为了同步多个线程对相同资源的并发访问，所以 ThreadLocal 不能解决这个问题
关于 InheritableThreadLocal
InheritableThreadLocal 类是 ThreadLocal 类的子类。ThreadLocal 中每个线程拥有它自己的值，与 ThreadLocal 不同的是，InheritableThreadLocal 允许一个线程以及该线程创建的所有子线程都可以访问它保存的值。
26. ThreadPool 用法与优势。
优势：
合理利用线程池能够带来三个好处。第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。第二：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。第三：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的分配，调优和监控。
用法：
newThreadPoolExecutor(corePoolSize,maximumPoolSize,keepAliveTime,
milliseconds,runnableTaskQueue, RejectedExecutionHandler)
corePoolsize:线程池大小 也就是说创建多大的线程
maximumPoolSize：最多容许多少线程
keepAliveTime：存活时间
TimeUnit unit：时间的单位
runnableTaskQueue：
-ArrayBlockingQueue：是一个基于数组结构的有界阻塞队列，此队列按 FIFO（先进先出）原则对元素进行排序。
-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。
-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。
-PriorityBlockingQueue：一个具有优先级的无限阻塞队列。
RejectedExecutionHandler：当队列和线程池都满了的时候
这是一个接口，实现类有
ThreadPoolExecutor.AbortPolicy, 直接抛出异常
ThreadPoolExecutor.CallerRunsPolicy, 只用调用者所在线程来运行任务
ThreadPoolExecutor.DiscardOldestPolicy, 丢弃队列里最近的一个任务，并执行当前任务
ThreadPoolExecutor.DiscardPolicy，不处理，丢弃掉
线程池的监控
通过线程池提供的参数进行监控。线程池里有一些属性在监控线程池的时候可以使用

taskCount：线程池需要执行的任务数量。
completedTaskCount：线程池在运行过程中已完成的任务数量。小于或等于 taskCount。
largestPoolSize：线程池曾经创建过的最大线程数量。通过这个数据可以知道线程池是否满过。如等于线程池的最大大小，则表示线程池曾经满了。
getPoolSize:线程池的线程数量。如果线程池不销毁的话，池里的线程不会自动销毁，所以这个大小只增不+ getActiveCount：获取活动的线程数。
通过扩展线程池进行监控。通过继承线程池并重写线程池的 beforeExecute，afterExecute 和 terminated 方法，我们可以在任务执行前，执行后和线程池关闭前干一些事情。如监控任务的平均执行时间，最大执行时间和最小执行时间等。这几个方法在线程池里是空方法。如：

```
protected void beforeExecute(Thread t, Runnable r) { }
```

27. Concurrent 包里的其他东西：ArrayBlockingQueue、CountDownLatch 等等。
java.util.concurrent 包含许多线程安全、测试良好、高性能的并发构建块。不客气地说，创建 java.util.concurrent 的目的就是要实现 Collection 框架对数据结构所执行的并发操作。通过提供一组可靠的、高性能并发构建块，开发人员可以提高并发类的线程安全、可伸缩性、性能、可读性和可靠性。
28. wait()和 sleep()的区别。

sleep()是让某个线程暂停运行一段时间,其控制范围是由当前线程决定,也就是说,在线程里面决定.好比如说,我要做的事情是 "点火->烧水->煮面",而当我点完火之后我不立即烧水,我要休息一段时间再烧.对于运行的主动权是由我的流程来控制.

而 wait(),首先,这是由某个确定的对象来调用的,将这个对象理解成一个传话的人,当这个人在某个线程里面说"暂停!",也是 thisOBJ.wait(),这里的暂停是阻塞,还是"点火->烧水->煮饭",thisOBJ 就好比一个监督我的人站在我旁边,本来该线 程应该执行 1 后执行 2,再执行 3,而在 2 处被那个对象喊暂停,那么我就会一直等在这里而不执行 3,但正个流程并没有结束,我一直想去煮饭,但还没被允许, 直到那个对象在某个地方说"通知暂停的线程启动!",也就是 thisOBJ.notify()的时候,那么我就可以煮饭了,这个被暂停的线程就会从暂停处 继续执行.

其实两者都可以让线程暂停一段时间,但是本质的区别是一个线程的运行状态控制,一个是线程之间的通讯的问题
sleep 和 wait 的区别有：
1，这两个方法来自不同的类分别是 Thread 和 Object
2，最主要是 sleep 方法没有释放锁，而 wait 方法释放了锁，使得其他线程可以使用同步控制块或者方法。
3，wait，notify 和 notifyAll 只能在同步控制方法或者同步控制块里面使用，而 sleep 可以在

```
 任何地方使用
synchronized(x){
   x.notify()
  //或者 wait()
}
```

4,sleep 必须捕获异常，而 wait，notify 和 notifyAll 不需要捕获异常

29. foreach 与正常 for 循环效率对比。

30. Java IO 与 NIO。

Java NIO(New IO)是一个可以替代标准 Java IO API 的 IO API（从 Java 1.4 开始)，Java NIO 提供了与标准 IO 不同的 IO 工作方式。

Java NIO: Channels and Buffers（通道和缓冲区）

标准的 IO 基于字节流和字符流进行操作的，而 NIO 是基于通道（Channel）和缓冲区（Buffer）进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。

Java NIO: Non-blocking IO（非阻塞 IO）

Java NIO 可以让你非阻塞的使用 IO，例如：当线程从通道读取数据到缓冲区时，线程还是可以进行其他事情。当数据被写入到缓冲区时，线程可以继续处理它。从缓冲区写入通道也类似。

基本上，所有的 IO 在 NIO 中都从一个 Channel 开始。Channel 有点象流。 数据可以从 Channel 读到 Buffer 中，也可以从 Buffer 写到 Channel 中。这里有个图示：

Channel 和 Buffer 有好几种类型。下面是 JAVA NIO 中的一些主要 Channel 的实现：

FileChannel DatagramChannel SocketChannel ServerSocketChannel

正如你所看到的，这些通道涵盖了 UDP 和 TCP 网络 IO，以及文件 IO。

与这些类一起的有一些有趣的接口，但为简单起见，我尽量在概述中不提到它们。本教程其它章节与它们相关的地方我会进行解释。

以下是 Java NIO 里关键的 Buffer 实现：

ByteBuffer CharBuffer DoubleBuffer FloatBuffer IntBuffer LongBuffer ShortBuffer

这些 Buffer 覆盖了你能通过 IO 发送的基本数据类型：byte, short, int, long, float, double 和 char。

Java NIO 还有个 MappedByteBuffer，用于表示内存映射文件， 我也不打算在概述中说明。

Selector

Selector 允许单线程处理多个 Channel。如果你的应用打开了多个连接（通道），但每个连接的流量都很低，使用 Selector 就会很方便。例如，在一个聊天服务器中

要使用 Selector，得向 Selector 注册 Channel，然后调用它的 select()方法。这个方法会一直阻塞到某个注册的通道有事件就绪。一旦这个方法返回，线程就可以处理这些事件，事件的例子有如新连接进来，数据接收等。

Java NIO: Selectors（选择器）

Java NIO 引入了选择器的概念，选择器用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个的线程可以监听多个数据通道。

31. 反射的作用与原理。

允许运行中的 Java 程序对自身进行检查，或者说“自审”，并能直接操作程序的内部属性和方法。

在运行时通过 classloader 加载 class

32. 泛型常用特点，List<String>能否转为 List<Object>。

33. 解析 XML 的几种方式的原理与特点：DOM、SAX、PULL。

Sax 特点

1. 解析效率高，占用内存少
2.可以随时停止解析
3.不能载入整个文档到内存

4.不能写入 xml

5.SAX 解析 xml 文件采用的是事件驱动

DOM 的特点

>优点

1.整个文档树在内存中，便于操作；支持删除、修改、重新排列等多种功能

2.通过树形结构存取 xml 文档

3.可以在树的某个节点上向前或向后移动

>缺点

1.将整个文档调入内存（包括无用的节点），浪费时间和空间

pull 解析器简介

1.pull 解析器是 android 内置的解析器，解析原理与 sax 类似

2.pull 它提供了类似的事件。

如：开始元素和结束元素事件，使用 parse.next()可以进入下一个元素并触发相应的事件，事件将作为数值代码被发送

因此可以使用一个 switch 对感兴趣的事件进行处理。当元素开始解析时，调用 parser.nextText()方法获取下一个 Text 类型节点的值

》pull 与 sax 的不同之处

1.pull 读取 xml 文件后触发相应的事件调用方法返回的是数字。

2.pull 可以在程序中控制，想解析到哪里就可以停止到哪里

3.Android 中更推荐使用 pull 解析

34. Java 与 C++对比。

35. Java1.7 与 1.8 新特性。

1.7:

1.二进制面值 前面加 ob

2.数字中可以加下划线,好看好分辨

3.switch 对 string 的支持

4.创建泛型时类型推断

1.8

支持 lamda 表达式

36. 设计模式：单例、工厂、适配器、责任链、观察者等等。

37. JNI 的使用。

1.编写带有 native 方法的 Java 类, 使用 javac 工具编译 Java 类

2.使用 javah 来生成与 native 方法对应的头文件

3.实现相应的头文件, 并编译为动态链接库(windows 下是.dll, linux 下是.so)

Java 里有很多很杂的东西，有时候需要你阅读源码，大多数可能书里面讲的不是太清楚，需要你在网上寻找答案。

推荐书籍：《java 核心技术卷 I》《Thinking in java》《java 并发编程》《effictive java》《大话设计模式》

JVM

1. 内存模型以及分区，需要详细到每个区放什么。
2. 堆里面的分区：Eden，survival from to，老年代，各自的特点。
3. 对象创建方法，对象的内存分配，对象的访问定位。
4. GC 的两种判定方法：引用计数与引用链。
5. GC 的三种收集方法：标记清除、标记整理、复制算法的原理与特点，分别用在什么地方，如果让你优化收集方法，有什么思路？
6. GC 收集器有哪些？CMS 收集器与 G1 收集器的特点。
7. Minor GC 与 Full GC 分别在什么时候发生？
8. 几种常用的内存调试工具：jmap、jstack、jconsole。
9. 类加载的五个过程：加载、验证、准备、解析、初始化。
10. 双亲委派模型：Bootstrap ClassLoader、Extension ClassLoader、ApplicationClassLoader。
11. 分派：静态分派与动态分派。

JVM 过去过来就问了这么些问题，没怎么变，内存模型和 GC 算法这块问得比较多，可以在网上多找几篇博客来看看。

推荐书籍：《深入理解 java 虚拟机》

操作系统

1. 进程和线程的区别。

线程和进程的主要区别是:他们是操作系统不同的管理系统资源的方式。进程拥有独立的地址空间，一个进程崩溃后，在保护模式下不会对其他的进程造成任何的影响，而线程只是进程不同的执行路径而已。线程有自己的堆栈和局部变量，但是线程没有独立的地址空间。通常可以将线程看成是一个轻量级的进程。

在 cpu 的调度方面，线程是调度的基本单位。并且线程的调度是比较轻量级的，提高了系统的并发性能。同一进程中的线程调度不会引起进程的调度，但是不同进程之间的线程切换还是会引起进程的切换的。

在执行过程来看，进程拥有独立的内存单元，同一进程下的线程可以共享改内存区域，提高了运行效率。

从逻辑角度来看：（重要区别）

多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但是，操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理及资源分配。

2. 死锁的必要条件，怎么处理死锁。

四个必要的条件：

1 互斥条件：一个资源每次只能被一个进程使用

2 请求与保持条件：一个进程因请求资源而阻塞，对已获得的资源保持不放。

3 不剥夺条件：进程获得的资源，在未使用完之前，不可以被强行剥夺

4 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系

处理死锁：

1 死锁的预防：破坏四个死锁的条件 但是都没一定的局限性 相应系统的执行效率

2 死锁的避免：银行家算法

3 死锁的解决：

（1）最简单，最常用的方法就是进行系统的重新启动，不过这种方法代价很大，它意味着在这之前所有的进程已经完成的计算工作都将付之东流，包括参与死锁的那些进程，以及未参与死锁的进程。

（2）撤消进程，剥夺资源。终止参与死锁的进程，收回它们占有的资源，从而解除死锁。这时又分两种情况：一次性撤消参与死锁的全部进程，剥夺全部资源；或者逐步撤消参与死锁的进程，逐步收回死锁进程占有的资源。一般来说，选择逐步撤消的进程时要按照一定的原则进行，目的是撤消那些代价最小的进程，比如按进程的优先级确定进程的代价；考虑进程运行时的代价和与此进程相关的外部作业的代价等因素。

3. Window 内存管理方式：段存储，页存储，段页存储。
分页存储管理基本思想：
用户程序的地址空间被划分成若干固定大小的区域，称为“页”，相应地，内存空间分成若干个物理块，页和块的大小相等。可将用户程序的任一页放在内存的任一块中，实现了离散分配。
页式管理的优点是没有外碎片，每个内碎片不超过页的大小。缺点是,程序全部装入内存，要求有相应的硬件支持。
分段存储管理基本思想：
将用户程序地址空间分成若干个大小不等的段，每段可以定义一组相对完整的逻辑信息。存储分配时，以段为单位，段与段在内存中可以不相邻接，也实现了离散分配。
段式管理优点是可以分别编写和编译，可以针对不同类型的段采用不同的保护，可以按段为单位来进行共享，包括通过动态链接进行代码共享。缺点是会产生碎片。
段页式存储管理基本思想：
分页系统能有效地提高内存的利用率，而分段系统能反映程序的逻辑结构，便于段的共享与保护，将分页与分段两种存储方式结合起来，就形成了段页式存储管理方式。
在段页式存储管理系统中，作业的地址空间首先被分成若干个逻辑分段，每段都有自己的段号，然后再将每段分成若干个大小相等的页。对于主存空间也分成大小相等的页，主存的分配以页为单位。
段页式系统中，作业的地址结构包含三部分的内容：段号 页号 页内位移量
程序员按照分段系统的地址结构将地址分为段号与段内位移量，地址变换机构将段内位移量分解为页号和页内位移量。
为实现段页式存储管理，系统应为每个进程设置一个段表，包括每段的段号，该段的页表始址和页表长度。每个段有自己的页表，记录段中的每一页的页号和存放在主存中的物理块号。段页式管理是段式管理与页式管理方案结合而成的所以具有他们两者的优点。但反过来说，由于管理软件的增加，复杂性和开销也就随之增加了。

4. 进程的几种状态。
创建，就绪，运行，阻塞，退出

1）就绪——执行：对就绪状态的进程，当进程调度程序按一种选定的策略从中选中一个就

绪进程，为之分配了处理机后，该进程便由就绪状态变为执行状态；
2）执行——阻塞：正在执行的进程因发生某等待事件而无法执行，则进程由执行状态变为阻塞状态，如进程提出输入/输出请求而变成等待外部设备传输信息的状态，进程申请资源（主存空间或外部设备）得不到满足时变成等待资源状态，进程运行中出现了故障（程序出错或主存储器读写错等）变成等待干预状态等等；
3）阻塞——就绪：处于阻塞状态的进程，在其等待的事件已经发生，如输入/输出完成，资源得到满足或错误处理完毕时，处于等待状态的进程并不马上转入执行状态，而是先转入就绪状态，然后再由系统进程调度程序在适当的时候将该进程转为执行状态；
4）执行——就绪：正在执行的进程，因时间片用完而被暂停执行，或在采用抢先式优先级调度算法的系统中,当有更高优先级的进程要运行而被迫让出处理机时，该进程便由执行状态转变为就绪状态。
5. IPC 几种通信方式。

通信方法	无法介于内核态与用户态的原因
管道（不包括命名管道）	局限于父子进程间的通信。
消息队列	在硬、软中断中无法无阻塞地接收数据。
信号量	无法介于内核态和用户态使用。
内存共享	需要信号量辅助，而信号量又无法使用。
套接字	在硬、软中断中无法无阻塞地接收数据。

共享文件

信号量:
信号机制是 UNIX 为进程中断处理而设置的。它只是一组预定义的值，因此不能用于信息交换，仅用于进程中断控制。例如在发生浮点错、非法内存访问、执行无 效指令、某些按键（如 ctrl-c、del 等）等都会产生一个信号，操作系统就会调用有关的系统调用或用户定义的处理过程来处理。
管道:
无名管道实际上是内存中的一个临时存储区，它由系统安全控制，并且独立于创建它的进程的内存区。管道对数据采用先进先出方式管理，并严格按顺序操作，例如不能对管道进行搜索，管道中的信息只能读一次。无名管道只能用于两个相互协作的进程之间的通信，并且访问无名管道的进程必须有共同的祖先。
有名管道的操作和无名管道类似,不同的地方在于使用有名管道的进程不需要具有共同的祖先，其它进程，只要知道该管道的名字，就可以访问它。管道非常适合进程之间快速交换信息。
共享储存段:
共享存储段是主存的一部分，它由一个或多个独立的进程共享。各进程的数据段与共享存储段相关联，对每个进程来说，共享存储段有不同的虚拟地址。
信号灯:
信号灯是一组进程共享的数据结构，当几个进程竞争同一资源时（文件、共享内存或消息队列等），它们的操作便由信号灯来同步，以防止互相干扰。

信号灯保证了某一时刻只有一个进程访问某一临界资源，所有请求该资源的其它进程都将被挂起，一旦该资源得到释放，系统才允许其它进程访问该资源。信号灯通常配对使用，以便实现资源的加锁和解锁。
进程间通信的实现技术的特点是：操作系统提供实现机制和编程接口，由用户在程序中实现，保证进程间可以进行快速的信息交换和大量数据的共享。但是，上述方式主要适合在同一台计算机系统内部的进程之间的通信。
6. 什么是存虚拟内。
虚拟内存是计算机系统**内存**管理的一种技术。 它使得应用程序认为它拥有连续的可用的**内存**（一个连续完整的地址空间），而实际上，它通常**是**被分隔成多个物理**内存**碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。
7. 虚拟地址、逻辑地址、线性地址、物理地址的区别。
逻辑地址（Logical Address） 是指由程序产生的与段相关的偏移地址部分。例如，你在进行 C 语言指针编程中，可以读取指针变量本身值(&操作)，实际上这个值就是逻辑地址，它是相对于你当前进程数据段的地址，不和绝对物理地址相干。只有在 Intel 实模式下，逻辑地址才和物理地址相等（因为实模式没有分段或分页机制,Cpu 不进行自动地址转换）；逻辑也就是在 Intel 保护模式下程序执行代码段限长内的偏移地址（假定代码段、数据段如果完全一样）。应用程序员仅需与逻辑地址打交道，而分段和分页机制对您来说是完全透明的，仅由系统编程人员涉及。应用程序员虽然自己可以直接操作内存，那也只能在操作系统给你分配的内存段操作。

线性地址（Linear Address） 是逻辑地址到物理地址变换之间的中间层。程序代码会产生逻辑地址，或者说是段中的偏移地址，加上相应段的基地址就生成了一个线性地址。如果启用了分页机制，那么线性地址可以再经变换以产生一个物理地址。若没有启用分页机制，那么线性地址直接就是物理地址。Intel 80386 的线性地址空间容量为 4G（2 的 32 次方即 32 根地址总线寻址）。

物理地址（Physical Address） 是指出现在 CPU 外部地址总线上的寻址物理内存的地址信号，是地址变换的最终结果地址。如果启用了分页机制，那么线性地址会使用页目录和页表中的项变换成物理地址。如果没有启用分页机制，那么线性地址就直接成为物理地址了。

虚拟内存（Virtual Memory） 是指计算机呈现出要比实际拥有的内存大得多的内存量。因此它允许程序员编制并运行比实际系统拥有的内存大得多的程序。这使得许多大型项目也能够在具有有限内存资源的系统上实现。一个很恰当的比喻是：你不需要很长的轨道就可以让一列火车从上海开到北京。你只需要足够长的铁轨（比如说 3 公里）就可以完成这个任务。采取的方法是把后面的铁轨立刻铺到火车的前面，只要你的操作足够快并能满足要求，列车就能象在一条完整的轨道上运行。这也就是虚拟内存管理需要完成的任务。在 Linux 0.11 内核中，给每个程序（进程）都划分了总容量为 64MB 的虚拟内存空间。因此程序的逻辑地址范围是 0x0000000 到 0x4000000。
因为是做 android 的这一块问得比较少一点，还有可能上我简历上没有写操作系统的原因。
推荐书籍：《深入理解现代操作系统》

TCP/IP

1. OSI 与 TCP/IP 各层的结构与功能，都有哪些协议。

OSI:物理层,数据链路层,网络层,运输层,会话层,表示层,应用层

五层模型:物理层.数据链路层,网络层,运输层,应用层

TCP/IP：网络接口层，网际层，运输层，应用层

应用层：通过应用进程间的交互来完成特定的网络应用，应用层协议有：http 协议，SMTP 协议(电子邮件)，FTP（文件传输）

运输层：为两个进程之间的通信提供通用的数据传输服务

传输控制协议（TCP）：提供面向连接的，可靠的数据传输服务，传输的单位是报文段，但是是面向字节流传输的，保证每一个字节都是正确传输的

用户数据报协议（UDP）：提供无连接的，尽最大可能交付的数据传输协议，传输单位是用户的数据报

网络层：网络层负责为分组交换网上的不同主机提供通信服务，协议有 IP 协议，传输单位是 IP 数据报

数据链数层：将 IP 数据报封装成帧，在相邻结点间的链路上传送帧

物理层：

2. TCP 与 UDP 的区别。

	TCP	UDP
是否连接	面向连接	面向非连接
传输可靠性	可靠的	不可靠的
应用场合	传输大里的数据	少里数据
速度	慢	快

2. TCP 报文结构。

UDP:

源端口	目的端口	长度	检验和

4. TCP 的三次握手与四次挥手过程，各个状态名称与含义，TIMEWAIT 的作用。

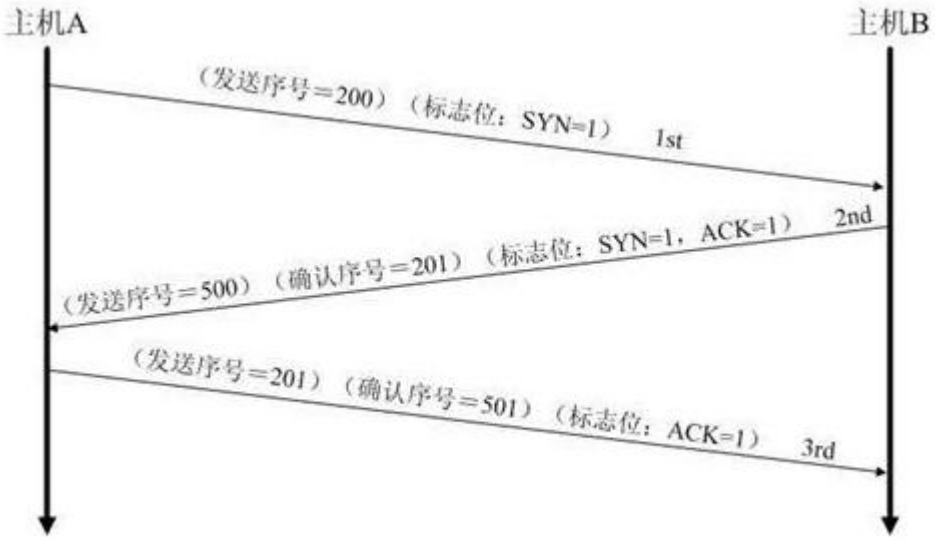
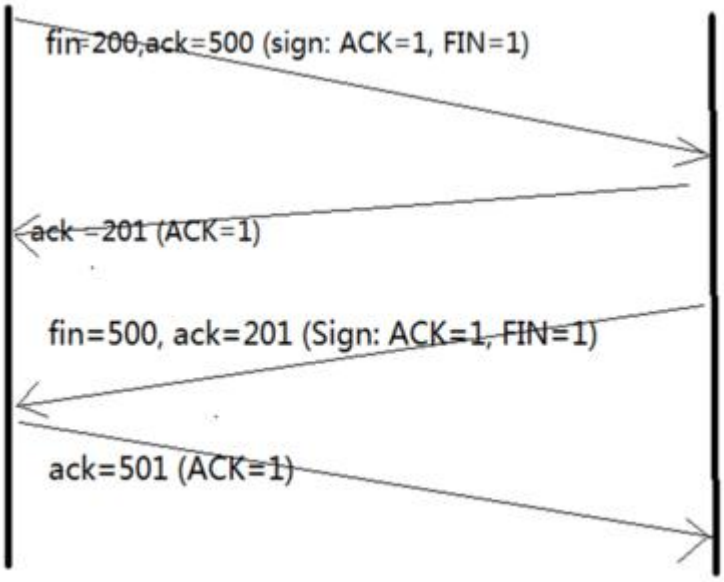


图1 TCP三次握手建立连接



为什么三次握手却四次挥手?

那可能有人会有疑问，在 tcp 连接握手时为何 ACK 是和 SYN 一起发送，这里 ACK 却没有和 FIN 一起发送呢。原因是因为 tcp 是**全双工模式，接收到 FIN 时意味将没有数据再发来，但是还是可以继续发送数据。**

TIME-WAIT

1 保证 A 最后发送的最后一个 ACK 报文段可以到达 B

2 防止出现”已失效的连接请求报文”

5. TCP 拥塞控制。

1 慢开始 2^0, 2^1,2^2 指数级增长

2 拥塞避免算法

3 快开始

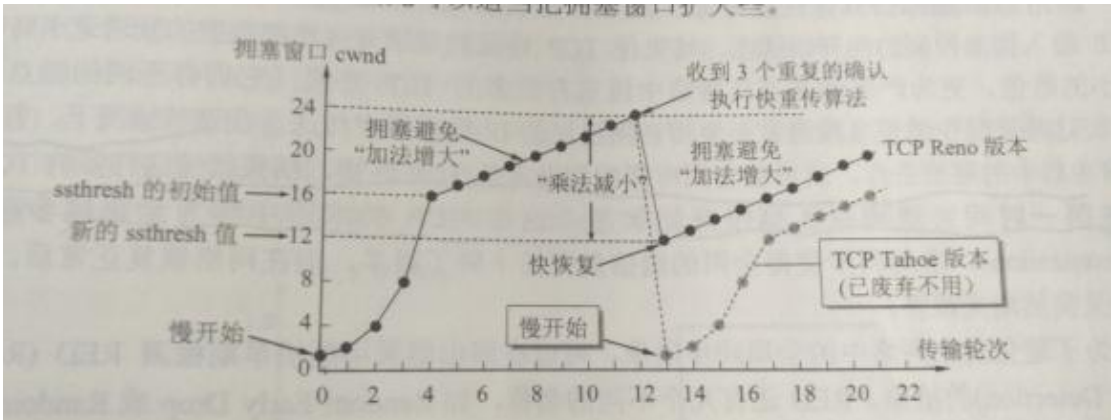
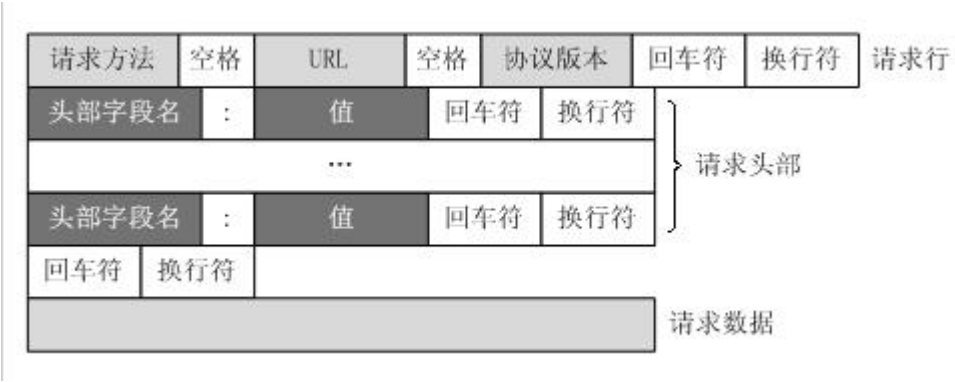


图 5-27 从连续收到三个重复的确认转入拥塞避免

6. TCP 滑动窗口与回退 N 针协议。

7. Http 的报文结构。
请求报文:
请求行,请求头部,请求数据



响应报文:状态行,消息报头,响应正文
状态行: 版本 状态码 短语
首部字段: 值
.
.
.
首部字段: 值

实体主体
8. Http 的状态码含义。
1xx:信息响应类，表示接收到请求并且继续处理
2xx:处理成功响应类，表示动作被成功接收、理解和接受
3xx:重定向响应类，为了完成指定的动作，必须接受进一步处理
4xx:客户端错误，客户请求包含语法错误或者是不能正确执行
5xx:服务端错误，服务器不能正确执行一个正确的请求
9. Http request 的几种类型。
HTTP GET: 获取资源

HTTP PUT/POST: 创建/添加资源
HTTP PUT/POST: 修改/增补资源
HTTP DELETE: 删除资源
10. Http1.1 和 Http1.0 的区别
1 长连接
HTTP 1.1 支持长连接（PersistentConnection）和请求的流水线（Pipelining）处理，在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟。例如：一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个单独的网页文件的请求和应答仍然需要使用各自的连接。
2 缓存
在 HTTP/1.0 中，使用 Expire 头域来判断资源的 fresh 或 stale，并使用条件请求（conditional request）来判断资源是否仍有效。例如，cache 服务器通过 If-Modified-Since 头域向服务器验证资源的 Last-Modefied 头域是否有更新，源服务器可能返回 304（Not Modified），则表明该对象仍有效；也可能返回 200（OK）替换请求的 Cache 对象。
此外，HTTP/1.0 中还定义了 Pragma:no-cache 头域，客户端使用该头域说明请求资源不能从 cache 中获取，而必须回源获取。
11. Http 怎么处理长连接。
在 HTTP 1.0 中
没有官方的 keepalive 的操作。通常是在现有协议上添加一个指数。如果浏览器支持 keep-alive，它会在请求的包头中添加：
Connection: Keep-Alive
然后当服务器收到请求，作出回应的时候，它也添加一个头在响应中：
Connection: Keep-Alive
这样做，连接就不会中断，而是保持连接。当客户端发送另一个请求时，它会使用同一个连接。这一直继续到客户端或服务器端认为会话已经结束，其中一方中断连接。
在 HTTP 1.1 中
所有的连接默认都是持续连接，除非特殊声明不支持。

所以现在各个浏览器的高版本基本都是支持长连接。

12. Cookie 与 Session 的作用于原理。
Session 用于保存每个用户的专用信息. 每个客户端用户访问时，服务器都为每个用户分配一个唯一的会话 ID（Session ID）. 她的生存期是用户持续请求时间再加上一段时间(一般是 20 分钟左右).Session 中的信息保存在 Web 服务器内容中,保存的数据量可大可小.当 Session 超时或被关闭时将自动释放保存的数据信息.由于用户停止使用应用程序后它仍然在内存中保持一段时间,因此使用 Session 对象使保存用户数据的方法效率很低.对于小量的数据,使用 Session 对象保存还是一个不错的选择
Cookie 用于保存客户浏览器请求服务器页面的请求信息,程序员也可以用它存放非敏感性的用户信息，信息保存的时间可以根据需要设置.如果没有设置 Cookie 失效日期,它们仅保存到关闭浏览器程序为止.如果将 Cookie 对象的 Expires 属性设置为 Minvalue,则表示 Cookie 永远不会过期.Cookie 存储的数据量很受限制,大多数浏览器支持最大容量为 4K,因此不要用来保存数据集及其他大量数据.由于并非所有的浏览器都支持 Cookie,并且数据信息是以明文文本的形式保存在客户端的计算机中,因此最好不要保存敏感的,未加密的数据,否则会影响网站的安全性

13. 电脑上访问一个网页，整个过程是怎么样的：DNS、HTTP、TCP、OSPF、IP、ARP。

域名解析 --> 发起 TCP 的 3 次握手 --> 建立 TCP 连接后发起 http 请求 --> 服务器响应 http 请求，浏览器得到 html 代码 --> 浏览器解析 html 代码，并请求 html 代码中的资源（如 js、css、图片等） --> 浏览器对页面进行渲染呈现给用户

14. Ping 的整个过程。ICMP 报文是什么。
ICMP 协议:网际控制报文协议
15. C/S 模式下使用 socket 通信，几个关键函数。

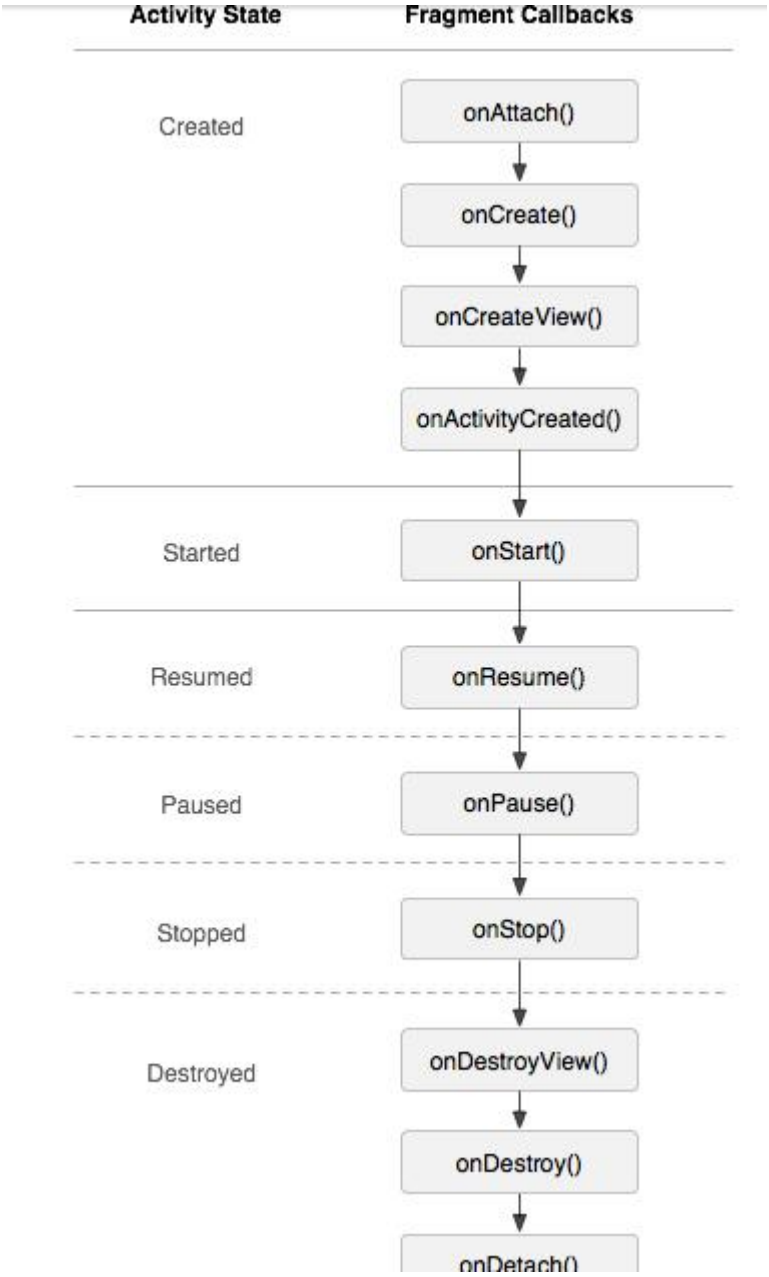
16. IP 地址分类。

17. 路由器与交换机区别。
交换机
用于同一网络内部数据的快速传输
转发决策通过查看二层头部完成
转发不需要修改数据帧
工作在 TCP/IP 协议的二层 —— 数据链路层
工作简单，直接使用硬件处理
路由器
用于不同网络间数据的跨网络传输
转发决策通过查看三层头部完成
转发需要修改 TTL ，IP 头部校验和需要重新计算，数据帧需要重新封装
工作在 TCP/IP 协议的三层 —— 网络层
工作复杂，使用软件处理
网络其实大体分为两块，一个 TCP 协议，一个 HTTP 协议，只要把这两块以及相关协议搞清楚，一般问题不大。
推荐书籍：《TCP/IP 协议族》

数据结构与算法
1. 链表与数组。
2. 队列和栈，出栈与入栈。
3. 链表的删除、插入、反向。
4. 字符串操作。
5. Hash 表的 hash 函数，冲突解决方法有哪些。
6. 各种排序：冒泡、选择、插入、希尔、归并、快排、堆排、桶排、基数的原理、平均时间复杂度、最坏时间复杂度、空间复杂度、是否稳定。
7. 快排的 partition 函数与归并的 Merge 函数。
8. 对冒泡与快排的改进。
9. 二分查找，与变种二分查找。
10. 二叉树、B+树、AVL 树、红黑树、哈夫曼树。
11. 二叉树的前中后续遍历：递归与非递归写法，层序遍历算法。
12. 图的 BFS 与 DFS 算法，最小生成树 prim 算法与最短路径 Dijkstra 算法。

13. KMP 算法。
14. 排列组合问题。
15. 动态规划、贪心算法、分治算法。（一般不会问到）
16. 大数据处理：类似 10 亿条数据找出最大的 1000 个数.....等等
算法的话其实是个重点，因为最后都是要你写代码，所以算法还是需要花不少时间准备，这里有太多算法题，写不全，我的建议是没事多在 OJ 上刷刷题（牛客网、leetcode 等），剑指 offer 上的算法要能理解并自己写出来，编程之美也推荐看一看。
推荐书籍：《大话数据结构》《剑指 offer》《编程之美》

Android
1. Activity 与 Fragment 的生命周期。



被创建时候的生命周期:

1. onAttach()//附着 2.onCreate()3.onCreateView()4.onActivityCreate()

当变成用户可见的时候：

2.onStart()2.onResume()

3.当进入后台模式的时候

1.onPause()2.onStop()

4.当被销毁的时候

1.onPause()2.onStop()3.omDestoryView()4.onDestory()5.onDetach()//分离

前三个的时候可以通过 bundle 恢复数据 使用 onSaveInstanceState()保存数据

Fragment 获取 activity：getActivity（）

Activity 获取 fragment：getFragmentManager.findFragmentById()

Fragment 与 fragment 通信：先获取 activity 然后在由 activity 获取到相应的 fragment

2. Acitivty 的四中启动模式与特点。

Standard:

SingleTop:

singleTask:

singleInstance:

3. Activity 缓存方法。

onSaveInstanceState()

onRestoreInstanceState()

当应用遇到意外情况（如：内存不足、用户直接按 Home 键）由系统销毁一个 Activity 时，onSaveInstanceState() 会被调用。但是当用户主动去销毁一个 Activity 时，例如在应用中按返回键，onSaveInstanceState()就不会被调用。因为在这种情况下，用户的行为决定了不需要保存 Activity 的状态。通常 onSaveInstanceState()只适合用于保存一些临时性的状态，而 onPause()适合用于数据的持久化保存。

在 activity 被杀掉之前调用保存每个实例的状态,以保证该状态可以在 onCreate(Bundle)或者 onRestoreInstanceState(Bundle) (传入的 Bundle 参数是由 onSaveInstanceState 封装好的)中恢复。这个方法在一个 activity 被杀死前调用，当该 activity 在将来某个时刻回来时可以恢复其先前状态。

onSaveInstanceState:在 onstop()方法之前

onRestoreInstanceState：在 onstart()之后

4. Service 的生命周期，两种启动方法，有什么区别。

生命周期：

Android Service 的生命周期并不像 Activity 那么复杂，它只继承了 onCreate(),onStart(),onDestroy()三个方法，当我们第一次启动 Service 时，先后调用了 onCreate(),onStart()这两个方法，当停止 Service 时，则执行 onDestroy()方法，这里需要注意的是，如果 Service 已经启动了，当我们再次启动 Service 时，不会在执行 onCreate()方法，而是直接执行 onStart()方法，具体的可以看下面的实例。

//现在多加了一个 onstartcommand 命令 官方建议用此代替 onstart onstart 被弃用

BindService()

startService()

与 activity 生命周期绑定

和 activity 无关，自己 stopSelf()

5. 怎么保证 service 不被杀死。

1. 双进程守护

2. onStartCommand 方法，返回 START_STICKY//因为内存不足被杀死后可以再次启动

1):START_STICKY：如果 service 进程被 kill 掉，保留 service 的状态为开始状态，但不保留递送的 intent 对象。随后系统会尝试重新创建 service，由于服务状态为开始状态，所以创建服务后一定会调用 onStartCommand(Intent,int,int)方法。如果在此期间没有任何启动命令被传递到 service，那么参数 Intent 将为 null。

2):START_NOT_STICKY：“非粘性的”。使用这个返回值时，如果在执行完 onStartCommand 后，服务被异常 kill 掉，系统不会自动重启该服务

3):START_REDELIVER_INTENT：重传 Intent。使用这个返回值时，如果在执行完 onStartCommand

后，服务被异常 kill 掉，系统会自动重启该服务，并将 Intent 的值传入。

3. 提升 service 优先级

```
<service
    android:name="com.dbjtech.acbxt.waiqin.UploadService"
    android:enabled="true" >
    <intent-filter android:priority="1000" >  //最大一千
        <action android:name="com.dbjtech.myservice" />
    </intent-filter>
</service>
```

4.提升线程的优先级

1.前台进程(FOREGROUND_APP)

2.可视进程(VISIBLE_APP)

3. 次要服务进程(SECONDARY_SERVER)

4.后台进程 (HIDDEN_APP)

5.内容供应节点(CONTENT_PROVIDER)

6.空进程(EMPTY_APP)

Startforefround()来启动一个前台进程 比如网易云音乐;

6. 在 ondestory 中通过广播再次启动 service //强制关闭 不回调用 ondestory

7. 监听一些系统广播 //5.0 以后开机广播等一些敏感广播不允许被非系统应用接受

6. 广播的两种注册方法，有什么区别。

静态和动态

7. Intent 的使用方法，可以传递哪些数据类型。

基本数据类型 string 实现序列化接口 // Serializable Parcelable

8. ContentProvider 使用方法。

1.在 mainfest 文件中注册 contentprovider

```
<provider android:name=".SomeProvider"
android:multiprocess="true" //允许多进程操作
android:authorities="com.your-company.SomeProvider"/>
```

2.创建 PersonProvider 类：继承自 contentProvide

onCreate、query、insert、update、delete 和 getType 这几个方法

9. Thread、AsycTask、IntentService 的使用场景与特点。

Thread：异步处理一些耗时的操作 一般要结合 handler 和 threadPool 一起使用会效率高点

asycTask：对 threadhandler 还有 threadpool 的封装，可以让我们很简单的实现以一些异步操作

intentService：自带异步处理的 service

重写他的 onHandleIntent 方法

10. 五种布局： FrameLayout 、 LinearLayout 、 AbsoluteLayout 、 RelativeLayout 、 TableLayout 各自特点及绘制效率对比。

11. Android 的数据存储形式。
sharePreference
file
sqlite
contentProvide
网络存储
12. Sqlite 的基本操作。
SQLiteDatabase：
Static 方法：SQLiteDatabase openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory ,int)
13. Android 中的 MVC 模式。
M 层：适合做一些业务逻辑处理，比如数据库存取操作，网络操作，复杂的算法，耗时的任务等都在 model 层处理。
V 层：应用层中处理数据显示的部分，XML 布局可以视为 V 层，显示 Model 层的数据结果。
C 层：在 Android 中，Activity 处理用户交互问题，因此可以认为 Activity 是控制器，Activity 读取 V 视图层的数据（eg.读取当前 EditText 控件的数据），控制用户输入（eg.EditText 控件数据的输入），并向 Model 发送数据请求（eg.发起网络请求等）。
14. Merge、ViewStub 的作用。
Merge:它可以删减多余的层级，优化 UI。<merge/>多用于替换 FrameLayout 或者当一个布局包含另一个时，<merge/>标签消除视图层次结构中多余的视图组。
viewStub:
<ViewStub />标签最大的优点是当你需要时才会加载，使用他并不会影响 UI 初始化时的性能。各种不常用的布局想进度条、显示错误消息等可以使用<ViewStub />标签，以减少内存使用量，加快渲染速度。<ViewStub />是一个不可见的，大小为 0 的 View。
`android:inflatedId="@+id/stub_comm_lv"` `findviewbyid` 然后调用 `inflate` 方法
判断是否已经加载过， 如果通过 setVisibility 来加载，那么通过判断可见性即可；如果通过 inflate()来加载是不可以通过判断可见性来处理的，而需要使用方式 2 来进行判断。
findViewById 的问题，注意 ViewStub 中是否设置了 inflatedId，如果设置了则需要通过 inflatedId 来查找目标布局的根元素。
15. Json 有什么优劣势。
结构简单 易于读写 有很成熟的解析框架 Gson fastJson
1.编码方面的区别
JSON 的编码更为清晰且冗余更少些，而 XML 比较适合于标记文档。JSON 网站提供了对 JSON 语法的严格描述，只是描述较简短。JSON 更适于进行数据交换处理。
2.编码可读性之间的区别
XML 有明显的优势，毕竟人类的语言更贴近这样的说明结构。JSON 读起来更像一个数据块，读起来就比较费解了。不过，我们读起来费解的语言，恰恰是适合机器阅读。
16. 动画有哪两类，各有什么特点？
一种方式是补间动画 Tween Animation、
Tween Animation

Tween Animation 定义在 xml 文件中。可以对 view 实现一系列的转换，例如：移动、渐变、伸缩、旋转。
另一种叫逐帧动画 Frame Animation
使用 Frame Animation 注意一下问题：

要在代码中调用 Imageview 的 setBackgroundResource 方法，如果直接在 XML 布局文件中设置其 src 属性当触发动画时会 FC。
在动画 start()之前要先 stop()，不然在第一次动画之后会停在最后一帧，这样动画就只会触发一次。
最后一点是 SDK 中提到的，不要在 onCreate 中调用 start，因为 AnimationDrawable 还没有完全跟 Window 相关联，如果想要界面显示时就开始动画的话，可以在 onWindowFoucsChanged()中调用 start()。
17. Handler、Loop 消息队列模型，各部分的作用。

18. 怎样退出终止 App。
MyApplication 类（储存每一个 Activity，并实现关闭所有 Activity 的操作）
广播关闭正在运行的 activity
主动制造异常结束程序，但是要 Thread.setDefaultUncaughtExceptionHandler(this);来接管异常信息抛出后的操作
19. Asset 目录与 res 目录的区别。
assets:用于存放需要打包到应用程序的静态文件，以便部署到设备中。与 res/raw 不同点在于，ASSETS 支持任意深度的子目录。这些文件不会生成任何资源 ID，必须使用/assets 开始（不包含它）的相对路径名。
1. assets 目录
AssetManager a = getAssets() ;
//fileName 为 assets 目录下需要访问的文件的名称
InputStream is = a.open(fileName) ;
//然后就可以通过输入流来读取 fileName 的内容了。
res:用于存放应用程序的资源（如图标、GUI 布局等），将被打包到编译后的 Java 中。不支持深度子目录
res/raw 目录
InputStream is = getResources().openRawResource(R.id.fileNameID) ;
//R.id.fileNameID 为需要访问的文件对应的资源 ID.接着我们就可以通过输入流来读取相应文件的内容了。

*res/raw 和 assets 的相同点：
1.两者目录下的文件在打包后会原封不动的保存在 apk 包中，不会被编译成二进制。

*res/raw 和 assets 的不同点：
1.res/raw 中的文件会被映射到 R.java 文件中，访问的时候直接使用资源 ID 即 R.id.filename；assets 文件夹下的文件不会被映射到 R.java 中，访问的时候需要 AssetManager 类。
2.res/raw 不可以有目录结构，而 assets 则可以有目录结构，也就是 assets 目录下可以再建立文件夹
20. Android 怎么加速启动 Activity。

现在很多的应用一开始点击的时候总会出现黑屏或者白屏，甚至前段时间微信也有同样的问题。其实白屏或者黑屏还是一些其他的东西，都是因为 Android 主题的问题，只要自己自定义一个启动主题，问题完美解决。

解决如下：

style 文件中添加：

```
<style name="AppStartLoad" parent="@android:style/Theme.NoTitleBar.Fullscreen">
       <item name="android:windowBackground">@drawable/splash</item>
       <item name="android:windowNoTitle">true</item>
   </style>
```

只要设置一个和启动 activity 一样的背景即可。

manifest 中引用：

```
<activity
           android:name="com.pztuan.module.Splash"
           android:theme="@style/AppStartLoad" >
           <intent-filter>
               <action android:name="android.intent.action.MAIN" />

               <category android:name="android.intent.category.LAUNCHER" />
           </intent-filter>
       </activity>
```

22. Android 内存优化方法：
ListView 优化，
及时关闭资源，
图片缓存等等。
减少 view 的层级
内存泄露可以引发很多的问题：
1.程序卡顿，响应速度慢（内存占用高时 JVM 虚拟机会频繁触发 GC）
2.莫名消失（当你的程序所占内存越大，它在后台的时候就越可能被干掉。反之内存占用越小，在后台存在的时间就越长）
3.直接崩溃（OutOfMemoryError）
22. Android 中弱引用与软引用的应用场景。

图片缓存

23. Bitmap 的四中属性，与每种属性队形的大小。

在 Android 中图片有四种属性，分别是：

ALPHA_8：每个像素占用 1byte 内存

ARGB_4444：每个像素占用 2byte 内存

ARGB_8888：每个像素占用 4byte 内存 （默认）

RGB_565：每个像素占用 2byte 内存(没有 alpha 属性)

24. View 与 View Group 分类。

自定义 View 过程：onMeasure()、onLayout()、onDraw()。

25. Touch 事件分发机制。

一、Touch 事件分析

事件分发：public boolean dispatchTouchEvent(MotionEvent ev)

Touch 事件发生时 Activity 的 dispatchTouchEvent(MotionEvent ev) 方法会以隧道方式（从根元素依次往下传递直到最内层子元素或在中间某一元素中由于某一条件停止传递）将事件传递给最外层 View 的 dispatchTouchEvent(MotionEvent ev) 方法，并由该 View 的 dispatchTouchEvent(MotionEvent ev) 方法对事件进行分发。dispatchTouchEvent 的事件分发逻辑如下：

- 如果 return true，事件会分发给当前 View 并由 dispatchTouchEvent 方法进行消费，同时事件会停止向下传递；
- 如果 return false，事件分发分为两种情况：
    1. 如果当前 View 获取的事件直接来自 Activity，则会将事件返回给 Activity 的 onTouchEvent 进行消费；
    2. 如果当前 View 获取的事件来自外层父控件，则会将事件返回给父 View 的 onTouchEvent 进行消费。
- 如果返回系统默认的 super.dispatchTouchEvent(ev)，事件会自动的分发给当前 View 的 onInterceptTouchEvent 方法。

事件拦截：public boolean onInterceptTouchEvent(MotionEvent ev)

在外层 View 的 dispatchTouchEvent(MotionEvent ev) 方法返回系统默认的 super.dispatchTouchEvent(ev) 情况下，事件会自动的分发给当前 View 的 onInterceptTouchEvent 方法。onInterceptTouchEvent 的事件拦截逻辑如下：

- 如果 onInterceptTouchEvent 返回 true，则表示将事件进行拦截，并将拦截到的事件交由当前 View 的 onTouchEvent 进行处理；
- 如果 onInterceptTouchEvent 返回 false，则表示将事件放行，当前 View 上的事件会被传递到子 View 上，再由子 View 的 dispatchTouchEvent 来开始这个事件的分发；
- 如果 onInterceptTouchEvent 返回 super.onInterceptTouchEvent(ev)，事件默认会被拦截，并将拦截到的事件交由当前 View 的 onTouchEvent 进行处理。

事件响应：public boolean onTouchEvent(MotionEvent ev)

在 dispatchTouchEvent 返回 super.dispatchTouchEvent(ev) 并且 onInterceptTouchEvent 返回 true 或返回 super.onInterceptTouchEvent(ev) 的情况下 onTouchEvent 会被调用。onTouchEvent 的事件响应逻辑如下：

- 如果事件传递到当前 View 的 onTouchEvent 方法，而该方法返回了 false，那么这个事件会从当前 View 向上传递，并且都是由上层 View 的 onTouchEvent 来接收，如果传递到上面的 onTouchEvent 也返回 false，这个事件就会“消失”，而且接收不到下一次事件。
- 如果返回了 true 则会接收并消费该事件。
- 如果返回 super.onTouchEvent(ev) 默认处理事件的逻辑和返回 false 时相同。

到这里，与 Touch 事件相关的三个方法就分析完毕了。下面的内容会通过各种不同的的测试案例来验证上文中三个方法对事件的处理逻辑。

26. Android 长连接，怎么处理心跳机制。

27. Zygote 的启动过程。

28. Android IPC:Binder 原理。
29. 你用过什么框架，是否看过源码，是否知道底层原理。
30. Android5.0、6.0 新特性。
蓝牙 4.1
更新 weiview 可以请求摄像头和麦克风的调用

运行时权限申请
移除了 httpClient

运行时权限：

1 在 AndroidManifest 文件中添加需要的权限。

2 这里涉及到一个 API，ContextCompat.checkSelfPermission，主要用于检测某个权限是否已经被授予，方法返回值为 PackageManager.PERMISSION_DENIED 或者 PackageManager.PERMISSION_GRANTED。当返回 DENIED 就需要进行申请授权了。
3 该方法是异步的，第一个参数是 Context；第二个参数是需要申请的权限的字符串数组；第三个参数为 requestCode，主要用于回调的时候检测。可以从方法名 requestPermissions 以及第二个参数看出，是支持一次性申请多个权限的，系统会通过对话框逐一询问用户是否授权。

```
ActivityCompat.requestPermissions(thisActivity,
                    new String[]{Manifest.permission.READ_CONTACTS},
                    MY_PERMISSIONS_REQUEST_READ_CONTACTS);
```

4 处理权限申请回调

```
public void onRequestPermissionsResult(int requestCode,
        String permissions[], int[] grantResults)
```