# Java基础

1. 面向对象的几个性质，举例说明

封装、继承、多态

1. 多态通常怎么实现
2. 多态的定义：多态是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用，在编译时不确定而是在程序运行期间确定的。感觉这个解释更合理一些。
3. 实现多态的技术称为：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法
4. 多态存在的三个必要条件

一、要有继承；二、要有重写；三、父类引用指向子类对象。（向上转型）

1. Java中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。
2. Java多态的实现原理
3. 多态的概念，
   1. 多态指使用父类对象的地方都可以使用子类的对象。比如有一个函数，参数是一个父类的对象，但是在调用该函数的时候，可以传一个子类的对象。·
   2. 同一操作，作用于不同的对象，可以有不同的解释，有不同的执行结果，这就是多态。简单的说就是父类的引用指向子类。多态的特性：消除了类型之间的耦合关系。优点：既改善了代码的组织结构和可读性，同时又保证了可扩展性。
4. 底层实现原理：Java使用了后期绑定的概念，当向对象发送消息时，在编译阶段，编译器只能保证被调用方法的存在，并对调用参数和返回类型进行检查，但是并不知道将被执行的确切代码，被调用的代码直到运行时才能确定。
5. Java中除了static方法、final方法之外，其他方法都属于后期绑定，后期绑定会涉及到JVM管理下一个重要的数据结构，方法表，方法表以数组的形式记录当前类及其父类的可见方法字节码在内存中的直接地址。
6. 动态绑定的具体调用过程：
   1. 首先会找到被调用方法的全限定名；
   2. 在此类的方法表中寻找被调用方法，如果找到，会将方法表中此方法的索引项记录到常量池中，如果没有，编译失败；
   3. 根据具体实例化的对象找到方法区中此对象的方法表，再找到方法表中被调用的方法，最后通过直接地址找到字节码所在的内存空间。
7. override和overload的区别
8. 重写是父类与子类之间多态性的一种表现，如果在子类中定义某方法与其父类有相同的名称和参数，我们称这个方法被重写；
9. 重载是一个类中多态性的一种表现，如果一个类中定义了多个同名的方法，他们或有不同的参数个数或有不同的参数类型或有不同的参数次序，则称为方法的重载。
10. 重写的注意事项：
    1. 覆盖的方法的标志必须要同被覆盖的方法的标志完全匹配，才能达到覆盖的效果；
    2. 覆盖的方法的返回值必须和被覆盖方法的返回一致；
    3. 覆盖的方法所抛出的异常必须和被覆盖的方法所抛出的异常一致，或者是其子类；
    4. 方法被定义为final则不能被重写；
11. 重载的注意事项：
    1. 在使用重载的时候，只能通过不同的参数形式；
    2. 不能通过访问权限、返回类型、抛出的异常进行重载；
12. 重载是一种参数多态机制，代码通过参数的类型或个数不同而实现的多态机制，是一种静态的绑定机制；
13. 重写是一种动态绑定的多态机制，即在父类和子类中同名函数有不同的实现代码。
14. String类可不可以继承

因为Sting是这样定义的：public final class String extends Object，里边有final关键字，所以不能被继承。

1. final关键字的作用

一旦将引用声明为final，则将不能改变这个引用。

1. final作用于类：则表示这个类是不能被继承的；
2. final作用于方法：代表这个方法不可以被子类的方法重写；
3. final作用于变量：代表这个变量是只读的，不可改变。
4. Static 的作用
5. Static可以被用来修饰成员方法、类的成员变量、例外可以编写static的静态代码块；
6. Static修饰成员方法，成为静态方法；静态方法不依赖任何的对象就能访问；在静态方法中不能访问非静态的成员变量和非静态的成员方法，因为非静态的成员变量和成员方法都必须依赖具体的对象才能被调用。
7. Static修饰成员变量：称为静态变量。静态变量和非静态变量的区别是：静态变量被所有的对象所共享，在内存中只有一个副本，他当且仅当在类初次加载时被初始化。而非静态变量是对象所拥有的，在创建对象的时候被初始化，存在多个副本，各个对象的副本之间不影响。
8. Static代码块：在类初次被加载的时候，会按照static块的顺序来执行每个static块，并且只执行一次。
9. Static 关键字
10. Static表示全局或者是静态的意思，用来修饰成员变量和成员方法，也可以用来修饰静态代码块。
11. 被static修饰的成员变量和成员方法独立于该类的任何对象，也就是说，它不依赖类特定的实例，被类的所有实例所共享，只要这个类被加载，Java虚拟机就能根据类名在运行时数据区的方法区内找到他们，static对象可以在他的任何对象创建之前访问，无需引用任何对象。
12. Static变量：
    1. Static修饰的变量称为静态变量，对于静态变量在内存中只有一个拷贝，JVM只为静态变量分配一次内存，在加载类的过程中完成静态变量的内存分配，可以用类名直接访问。但对于实例变量，每创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响。
13. 静态方法：
    1. 静态方法可以直接通过类名调用，任何实例也都可以调用，静态方法中不能有this和super关键字；
    2. 静态方法不能访问所属类的实例变量和实例方法，只能访问所属类的静态成员变量和成员方法。
14. 静态代码块：

静态代码块是独立于类成员的，JVM加载类时会执行这些静态代码块，如果静态代码块有多个，JVM将按照其在类中出现的先后顺序依次执行他们。

1. ==和equals的区别
2. == 用于基本类型的比较；判断引用是否指向堆内存的同一块地址；
3. Equals用于判断两个变量是否是对同一个对象的引用，即堆中的内容是否相同。
4. String类型比较内容是否相同应该用equals。
5. 如何实现一个不可变类

如果一个类是不可变的，那么它的所有数据域必须都是私有的，而且没有对任何一个数据域提供公共的Set方法。

要是一个类成为不可变的，他必须满足：

1. 所有数据域都是私有的；
2. 没有修改器方法（也就是Set方法）；
3. 没有一个访问器方法，他会返回一个指向可变数据域的引用。
4. String、StringBuffer与StringBuilder的区别

String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象， StringBuffer和StringBuilder底层是char[]数组实现的，StringBuffer是线程安全的，而StringBuilder是线程不安全的。

1. Final关键字
2. final关键字有这是无法改变的或者终态的含义；
3. final类不能被继承，没有子类，final类中的方法默认是final的；
4. final方法不能被子类的方法覆盖，但是可以被继承；
5. final成员变量表示常量，只能被赋值一次，赋值后值不再改变；
6. final不能用于修饰构造方法；
7. 当函数参数类型是final类型时，你可以使用该参数，但是无法改变该参数的值。
8. 什么是抽象类

如果一个父类设计的十分抽象，以致于他没有任何具体的实例，称其为抽象类。

抽象类不能用new操作符创建他的实例，抽象方法只有定义没有实现，它的实现由子类提供。一个包含抽象方法的类必须被称为抽象类。抽象类的构造方法设置为protected，因为他只能被子类使用。

1. 什么是接口

接口中只包含常量和抽象方法，接口中的常量都定义为public final static ，所有的方法都是public abstract。

1. 抽象类和接口的区别

|  |  |  |  |
| --- | --- | --- | --- |
|  | 变量 | 构造方法 | 方法 |
| 抽象类 | 无限制 | 子类通过构造方法链调用构造方法，抽象类不能用new操作符实例化。 | 无限制 |
| 接口 | 所有变量都是public static final | 没有构造方法，接口不能用new操作符实例化 | 所有方法必须是公共的抽象实例方法。 |

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，但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是抽象的，不能有非抽象的普通方法；
4. 抽象类中可以包含静态方法，接口中不能包含静态方法。
5. 一个类可以实现多个接口，但只能继承一个抽象类。
6. 对多线程的理解：
7. 多线程出现的原因：

为了解决负载均衡问题，充分利用CPU资源，为了提高CPU的使用率，采用多线程的方式去同时完成几件事而互不干扰。

1. 采用多线程的好处：
   1. 使用线程可以把占据时间长的程序中的任务放到后台去处理；
   2. 程序运行的效率更高；
   3. 在一些等待的任务实现上如用于输入，文件读取和网络收发等，线程就比较有用。
2. 使用多线程的缺点：
   1. 如果有大量的线程，可能会影响性能，因为操作系统需要在他们之间切换；
   2. 更多的线程需要更多的内存空间
   3. 线程终止需要考虑对程序的影响
   4. 通常块模型数据是在多个线程之间共享的，需要防止线程死锁情况的发生。
3. Java锁机制，线程同步：

同步指的是多个线程并发的访问共享数据时，保证共享数据在同一时刻只被一个线程使用；

实现同步的方式：

1. 互斥同步
   1. Synchronized关键字：
      1. Synchronized关键字能够保证在同一时刻最多只有一个线程执行该代码。
      2. synchronized关键字在编译后，会在同步代码块前后形成monitorenter和monitorexit两条指令，这两个字节码都需要一个引用类型的参数来指明要锁定和解锁的对象。如果synchronized明确指明了对象参数，那就是这个对象的引用，如果没有指明，synchronized修饰的是实例方法，去取对应的对象实例来作为锁对象，synchronized修饰的是类方法，去取Class对象来作为锁对象。
      3. 另外在monitorenter执行时，首先要尝试获取对象的锁，如果这个对象没有被锁定，或者当前线程已经有了那个对象的锁，把锁的计数器加1，在执行monitorexit指令时，会将锁的计数器减1，当计数器等于0的时候，锁就会被释放。
   2. 使用java.util.concurrrent包下的重入锁来实现同步，（ReentrantLock）：
      1. 基本的使用方法与synchronized相似，只是代码写法有点不同，一个表现为API层面的互斥锁，一个是原生语法层面的互斥锁。
      2. 比synchronized多一些高级的性能：如等待可中断、公平锁、锁绑定多个条件。
2. 非阻塞同步

上面的互斥同步，总是认为不进行同步就是不安全的，无论是否会出现数据的竞争，都需要进行加锁，引发的问题就是线程阻塞和唤醒的性能问题。

随着硬件指令集的发展，可以采取一些基于冲突检测的乐观并发策略，就是先进行操作，没有其他线程争用共享数据的话操作就成功了，如果有数据共享争用，就产生了冲突，那就再采取其他的补偿措施。这种同步方式称为非阻塞同步。

1. 使用volatile变量
   1. 被volatile修饰的变量，可以保证变量对所有线程是可见的，可见性是指当一条线程修改了这个变量的值，新值对其他线程来说是立刻可知的。
2. java锁机制，线程同步
3. 线程同步：
   1. 使用synchronized同步方法（用synchronized关键字修饰方法），由于java每个对象都有一个内置锁，当用此关键字修饰方法时，内置锁会保护整个方法，在调用该方法前，需要获得内置锁，否则就处于阻塞状态。
   2. 使用synchronized同步代码块；
   3. Volatile关键字；volatile关键字为域变量提供了一种免锁机制，使用volatile修饰域相当于告诉虚拟机该域可能会被其他线程更新，因此每次使用该域就要重新计算，而不是使用寄存器的值。
   4. 使用重入锁（ReentrantLock）实现线程同步
   5. 使用局部变量（ThreadLocal）实现线程同步
4. java四种修饰符（private,protected,public,default）的特点
5. 对于public修饰符，它具有最大的访问权限，可以访问任何一个在CLASSPATH下的类、接口、异常等。它往往用于对外的情况，也就是对象或类对外的一种接口的形式。
6. 对于protected修饰符，它主要的作用就是用来保护子类的。它的含义在于子类可以用它修饰的成员，其他的不可以，它相当于传递给子类的一种继承的东西。
7. 对于default来说，有的时候也成为friendly（友员），它是针对本包访问而设计的，任何处于本包下的类、接口、异常等，都可以相互访问，即使是父类没有用protected修饰的成员也可以。
8. 对于private来说，它的访问权限仅限于类的内部，是一种封装的体现，例如，大多数的成员变量都是修饰符为private的，它们不希望被其他任何外部的类访问。

下表为[Java](http://lib.csdn.net/base/javase)访问控制符的含义和使用情况

| 修饰符 | 类内部 | 本包 | 子类 | 外部包 |
| --- | --- | --- | --- | --- |
| public | √ | √ | √ | √ |
| protected | √ | √ | √ | X |
| default | √ | √ | X | X |
| private | √ | X | X | X |

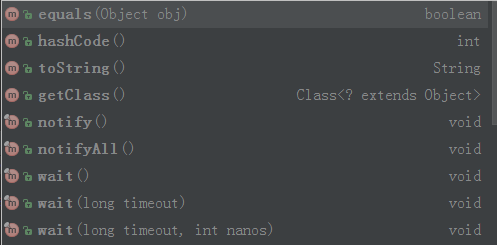
**注意**：Java的访问控制是停留在编译层的，也就是它不会在.class文件中留下任何的痕迹，只在编译的时候进行访问控制的检查。其实，通过反射的手段，是可以访问任何包下任何类中的成员，例如，访问类的私有成员也是可能的。

**区别：**   
1. public：可以被所有其他类所访问。   
2. protected：自身，子类及同一个包中类可以访问。   
3. default（默认）：同一包中的类可以访问，声明时没有加修饰符，认为是friendly。   
4. private：只能被自己访问和修改。

1. 静态内部类、非静态内部类、匿名内部类的特点和区别
2. 非静态内部类
   1. 变量和方法不能声明为静态的；
   2. 实例化的时候需要依附在外部类上面。比如：B是A的非静态内部类，实例化B，则A.B b = new A().new B();
   3. 内部类可以引用外部类的静态的或者非静态属性或者方法。
3. 静态内部类：
   1. 属性或者方法可以声明为静态的或者非静态的；
   2. 实例化静态内部类，比如B是A的静态内部类，A.B b = new A.B();
   3. 内部类只能引用外部类的静态的属性或者方法；
   4. 如果属性或者方法声明为静态的，那么可以直接通过类名直接使用，比如B是A的静态内部类，b()是B的一个静态属性，则可以 A.B.b()。
4. 匿名内部类：
   1. 匿名内部类是不能加访问修饰符的，
   2. 注意匿名内部类的外部类的形参，当所在的方法的形参需要被内部类里面使用时，该形参必须为final。

理解就是：拷贝引用，为了避免引用值发生改变，例如被外部类修改等，而导致内部类得到的值不一致，于是用final来让该引用不可改变。

1. Object类及其常用方法



* + 1. toString（）方法：

该方法在打印对象时被调用，将对象信息变为字符串返回，默认输出对象地址。

可以通过重写Object类的toString（）方法来输出对象属性信息。

* + 1. equals()方法：

用于比较对象是否相等，比较的是两个对象的地址，如果想要比较内容是否相等，可以采用重写equals方法的方式。

* + 1. hashCode（） 方法：

该方法用来返回其所在对象的物理地址（哈希码值）

通常会和equals方法同时重写，确保相等的两个对象拥有相等的hashCode。

* + 1. wait方法

wait方法是使当前线程等待该对象的锁，当前线程必须是该对象的拥有者，也就是具有该对象的锁。

* + 1. notify方法

该方法唤醒在该对象上等待的某个线程。

* + 1. notifyAll方法

该方法唤醒在该对象上等待的所有线程。

* + 1. finalize（）方法

该方法用于释放资源，因为无法确定该方法什么时候被调用，很少使用。

1. equals方法和hashCode方法
   * 1. 重写equals方法时一定要重写hashCode方法；
     2. 两个对象如果equals，那么这两个对象的hashCode值一定相同；
     3. 如果两个对象的hashCode值相等，那么这两个对象不一定相等。
2. Java的序列化与反序列化
   * 1. Java序列化就是把对象转换成字节序列，Java反序列化就是字节序列还原成Java对象。
     2. 具体的实现方式，可以采用Parcelable与Serializable

a) Serializable序列化接口，实现的时候特别方便，只需要继承Serializable接口即可，但是缺点是使用了反射，序列化的过程比较慢，这种机制在序列化的时候创建了许多临时对象，容易触发垃圾回收。

b) Parcelable 速度比较快，实现起来比较麻烦；

需要重写writeToParcel方法，以及调用Parcelable.Creator方法。

1. 说说java里面的集合框架，各自是如何使用，有哪些区别和缺点，

Java中的集合框架主要支持三种接口：List、Set、Queue。Set实例用于存储一组不重复的元素，List实例用于存储一个由元素构成的有序集合，Queue实例用于存储先进先出方式处理的对象。

1. Collection集合框架



Set实例中不允许有重复的元素；HashSet中的元素没有规律，LinkedHashSet可以使得元素按照插入的顺序排列，TreeSet可以保证元素是有序的。

List中允许有重复的元素，ArrayList是基于数组实现，LinkedList是基于链表实现的；

1. Map框架



1. Map、Set、List、Queue、Stack的特点与用法。

Map：采用键值对的存储方式；

Set：集合内的元素不允许重复；

List：集合内元素时有序的；

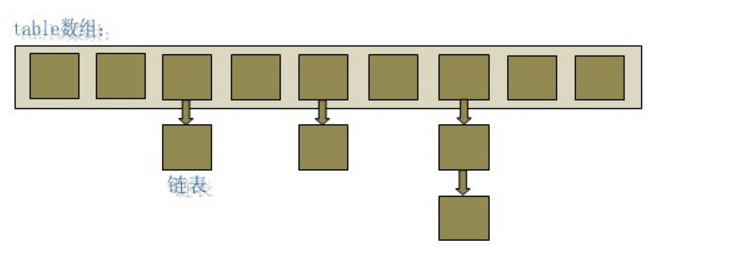
Queue：先进先出；

Stack：后进先出；继承自Vector，Vector由实现了LISt接口

1. Collection与Collections的区别
2. Collection是集合类的上级接口，它的具体实现有List和Set等。
3. Collections是针对集合类的一个帮助类，提供了一系列的静态方法实现对各种集合的搜索、排序、线程安全等操作。此类不能实例化。
4. Collections.sort排序内部原理

Collections.sort()内部实现的时候，对于基本的数据类型，使用快排来实现，对于对象类型，采用归并实现，对于对象类型的排序稳定性是非常关键的，归并排序的实现是稳定的，而快排是不稳定的。

1. Comparator 和Comparable之间的区别
   * 1. Comparable是内比较器，很多类都会实现这个接口，给类提供默认的比较功能；含有一个抽象方法int compareTo();
     2. Comparator是外比较器，一般情况下，在类没有实现comparable接口时，可以在Collections.Sort()方法中用一个匿名内部类的方式，实现一个Comparator接口，让其具有比较的功能；其有两个抽象方法：int compare();和boolean equals（）,但是一般情况下，不用实现equals方法，因为其继承自Object，Object中已经实现了该方法。
2. TreeMap、HashMap、LindedHashMap的区别。
3. HashMap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，遍历时取得的数据顺序完全是随机的。
4. TreeMap实现了SortMap接口，能够把保存的记录按照键排序，默认排序是按照升序排列。
5. LinkedHashMap保存了记录的插入顺序，在用Iterator遍历时，得到的顺序是插入顺序，速度比HashMap慢。
6. HashMap的内部原理，java8做的改变
7. HashMap是基于哈希表的Map接口的非同步实现，HashMap不是同步的，如果多个线程同时访问一个HashMap，而其中至少一个线程从结构上修改了，必须保持外部同步，以防止对映射进行意外的非同步访问。
8. HashMap时通过数组和指针两种数据结构实现的，HashMap实际上是一个“链表散列”的数据结构，底层是一个数组结构，数组中每一项又是一个链表，当新建一个链表的时候就会初始化一个数组。



1. 当我们往HashMap中put元素时，先根据key的hashCode重新计算hash值，根据hash值得到这个元素在数组中的位置，如果数组该位置上已经存放有其他元素，那么在该位置上的元素将以链表的形式存放，新加入的放在链表头，最先加入的放在链表尾。如果数组该位置没有元素，就直接将该元素放到此数组中的该位置上。（尽量将元素存在数组的不同位置上，这样子查找的时候就不必再在链表中查找，以提高效率）；
2. HashMap 在底层将 key-value 当成一个整体进行处理，这个整体就是一个 Entry 对象。HashMap 底层采用一个 Entry[] 数组来保存所有的 key-value 对，当需要存储一个 Entry 对象时，会根据 hash 算法来决定其在数组中的存储位置，在根据 equals 方法决定其在该数组位置上的链表中的存储位置；当需要取出一个Entry 时，也会根据 hash 算法找到其在数组中的存储位置，再根据 equals 方法从该位置上的链表中取出该Entry。
3. HashMap默认的初始容量是16，默认的加载因子0.75。
4. 扩容是根据数组大小和加载因子决定的，当数组中的元素超过数组大小 \* 加载因子时，就将数组扩大成数组大小\* 2；
5. Hashmap中加载因子的作用

加载因子主要用于扩容时门限值的计算，当数组的容量大于数组大小\* 加载因子时，就将数组扩大到原来的2倍。

当加载因子过大的时候，给数组扩容的可能性就小，所以占用的相对内容就少，但是每条entry链上的元素会相对较多，查询的时间也会增长，当加载因子过小的时候给数组扩容的可能性高，内存占用多，但是entry链上的元素就会相对较少，查询时间也会减少。所以加载因子是时间和空间上的折中。。

1. HashMap的rehash

Hashmap在扩容时，新数组的容量将是原来的2倍，由于容量发生变化，原有的每个元素需要重新计算bucketIndex，再存放到新数组中，也就是所谓的rehash。

Hashmap中计算index值是通过哈希值%容量得到的，当发生hash冲突时，采用链地址法来解决冲突。

1. HashMap与HashTable的区别
2. HashMap和HashTable都实现了Map接口，他们的主要区别有线程安全性、同步、以及速度；
3. HashMap几乎可以等同是HashTable，除了HashMap是非synchronized的，并可以接受null的键值，但是HashTable不行；
4. HashMap是非synchronized的，而HashTable是synchronized，这意味着HashTable是线程安全的，多个线程可以共享一个HashTable，如果没有正确同步的话，多个线程是不能同步HashMap的。
5. HashMap的迭代器是fail-fast迭代器，而HashTable的迭代器是enumerator迭代器。
6. HashTable是线程安全的也是synchronized，所以在单线程环境下他比HashMap要慢，如果不需要同步，只需要单线程，使用HashMap的性能要比HashTable好。
7. 两种方式让HashMap是安全的
   * 1. 通过Collections.synchronizedMap（）返回一个新的Map，这个map是线程安全的。
     2. 用concurrentHashMap
     3. 两种方式的对比
        1. 实现原理
           1. 前者使用了经典的synchronized来进行互斥；并使用代理模式new了一个新的类，这个类同样实现了Map接口。
           2. 后者使用了新的锁机制（可以称为是乐观锁），把HashMap进行了拆分，拆分成了对立的块，这样在高并发的情况下减少了冲突的可能性。
        2. 锁机制
           1. 前者使用synchronized方法，是一种悲观锁，在进入之前需要获得锁，确保独享当前对象，然后做相应的修改/读取。
           2. 后者使用的是乐观锁，只有在需要修改对象时，比较和之前的值是否被人修改了，如果被修改了则返回失败。
        3. 如何得到/释放锁
        4. 优缺点
           1. 前者代码实现非常简单，后者需要互斥的代码比较少。
8. ConcurrentHashMap
   * 1. ConcurrentHashMap是将数据分成一段一段的存储，然后给每一段数据分别加锁，当一个线程占用一个锁访问一段数据时，其他的线程也可以访问其他的分段。
     2. ConcurrentHashMap实现的过程中，由hashEntry和segment组成，hashEntry用于保存键值对，segment是一个锁的代表。一个ConcurrentHashMap由多个segment数组组成，一个segment由多个桶组成，每个桶又是由多个hashEntry链接的链表组成。
     3. 默认情况下segment的大小为16个。
     4. 与HashTable的区别：HashTable只有一个对象锁，在并发的条件下，只允许一个线程访问容器。
9. ConncurrentHashMap和hashtable比较，ConncurrentHashMap问的非常深入，感觉必须的看点源码才行（比如两个线程并发访问map中同一条链，一个线程在尾部删除，一个线程在前面遍历查找，问为什么前面的线程还能正确的查找到后面被另一个线程删除的节点）
10. Hashmap本质是数组加链表。根据key取得hash值，然后计算出数组下标，如果多个key对应到同一个下标，就用链表串起来，新插入的在前面。
11. ConcurrentHashMap：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，避免多线程锁的几率，提高并发效率。
12. 与HashTable的区别：HashTable只有一个对象锁，在并发的条件下，只允许一个线程访问容器。
13. LinkedHashMap
    * 1. LinkedHashMap继承自hashmap，但是他能保证插入顺序和输出顺序相同。
      2. 底层实现使用了hashmap的hash表（也就是底层的数组结构）以及重新构建的双向链表。在数组中存入的entry，不仅保存了当前对象的引用，也保存了前一个befor和后一个after的引用。
      3. 在插入元素时，不在调用put方法，二是调用addEntry方法，以链表的形式进行插入和删除操作。
      4. 因为在初始化的时候调用的是hashmap的初始化函数，所以基本的扩容机制应该是一致的（自己理解的，并没有被验证）
14. hashSet的底层实现

HashSet是基于HashMap实现的，HashSet的元素都存放在hashMap的key上，而value中的值都是统一的一个private static final Object PRESENT = new Object（）;hashSet中的add方法调用的是底层的hashMap中的put方法，而如果在hashMap中调用put首先会判断key是否存在，如果key存在，则修改value的值，如果key不存在，就插入key-value。而在Set中因为value值没有用，也就不存在修改value值的说法，因此在hashSet中插入元素，首先判断元素是否存在，如果不存在，则插入，如果存在则不插入。

1. ArrayList与LinkedList的区别，以及与Vector的区别
2. ArrayList是基于动态数组实现的，LinkedList是基于链表的数据结构；
3. 对于随机访问get和set，ArrayList的性能优于LinkedList，（参考数组和链表来思考）；
4. 对于新增和删除操作，LinkedList性能优于ArrayList，LinkedList只需要对指针进行修改，但是ArrayList需要移动数据来填补被删除的对象空间。
5. 两者都是线程不安全的；
6. Vector采用数组方式存储数据，随机的读取性能好，但是插入数据慢，Vector使用了synchronized方法线程安全的，所以性能上比ArrayList差。因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例如，添加或删除了一些元素），这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该异常。
7. ArrayList和Vector的底层实现
8. ArrayList的底层实现是靠数组实现的，默认是初始化一个长度为10的数组，随着数据的增加，创建一个新的数组然后将旧数组中的数据拷贝过来。创建新数组的长度为：

新的容量 = （旧的容量 \* 3） / 2 + 1;

1. Vector中的public方法都添加了synchronized关键字，以确保方法同步；
   1. Vector中有三个属性，分别是存储数据的数组，存储记录的数目，扩展数组大小的扩容因子。
   2. 当扩容因子大于0时，新数组长度为原来数组长度+扩容因子，否则新数组长度为原数组长度的2倍。
2. 进程与线程的区别
3. 进程是并发执行的程序在执行过程中分配和管理资源的基本单位。每一个进程都有一个自己的地址空间，进程有五种状态：初始态、执行态、等待状态、就绪状态和终止状态。
4. 线程，在网络或者多用户环境下，一个服务器通常需要就接收大量且不确定数量用户的并发请求，为每一个请求都创建一个进程显然是行不通的，无论是从系统资源开销方面或者响应用户请求的效率方面来看，因此引入了线程。线程是进程的一部分。
5. 进程是系统进行资源分配的基本单位，有独立的内存空间。
6. 线程是CPU调度的基本单位，没有单独地址空间，有独立的栈，局部变量、寄存器、程序计数器。
7. 进程拥有一个完整的虚拟地址空间，不依赖与线程而独立存在，反之，线程是进程的一部分，没有自己的地址空间，与进程内其他线程一起共享分配给该进程的所有资源。
8. 创建进程的开销大，包括创建虚拟地址空间等需要大量系统资源； 创建线程开销小，基本上只有一个内核对象和一个堆栈。
9. 进程的状态
10. 就绪状态
11. 运行状态
12. 阻塞状态
13. 线程的几种状态

线程从创建、运行到结束总是处于五个状态：新建状态、就绪状态、运行状态、阻塞状态、以及死亡状态。

1. 新建状态:当用new操作符创建一个线程时，线程还没开始运行，此时处于新建状态；
2. 就绪状态：当线程对象调用start方法即启动了线程，start方法创建线程运行的系统资源，并调度线程运行的run方法，当start方法返回之后，线程就处于就绪状态。
3. 运行状态：当线程获得了CPU时间后，他才进入运行态，真正开始执行run方法；
4. 阻塞状态：进入阻塞状态的原因
   1. 线程调用sleep方法
   2. 线程调用一个在I/O上被阻塞的操作，即该操作在输入输出完成之前不会返回到它的调用者；
   3. 线程试得到一个锁，但是该锁被其他线程持有
   4. 线程在等待某个触发条件

阻塞状态是正在运行的线程没有运行结束，暂时让出CPU，这时其他处于就绪状态的线程就可以获得CPU时间，进入运行状态。

1. 死亡状态：有两个原因会导致线程死亡，一是，run方法正常退出而自然死亡；二是一个未捕获的异常终止了run方法而使线程猝死。
2. 线程池的概念、好处、常见的线程池举例
3. ThreadPoolExecutor中构造函数中参数的意义：
   1. CorePoolSize：核心线程数，默认情况下，核心线程会在线程池中一直存活，即使他们处于空闲状态；
   2. maximumPoolSize：线程池所能容纳的最大线程数，当活动线程数达到这个值后，后续任务将会被阻塞；
   3. KeepAliveTime：非核心线程闲置时的超时时长，超过这个值，非核心线程会被回收。
   4. WorkQueue: 线程池中的任务队列。
4. 线程池的好处
   1. 重用线程池中的线程，避免因为线程的创建和销毁所带来的性能开销；
   2. 能有效控制线程池的最大并发数，避免大量的线程之间因互相抢占系统资源而导致的阻塞现象；
   3. 能够对线程进行简单的管理，并提供定时执行以及指定间隔循环执行等功能。
5. 常见的线程池举例
   1. FixedThreadPool：一种线程数量固定的线程池，当线程处于空闲状态时，他们并不会被回收，除非线程池被关闭了。当所有的线程都处于活动状态时，新任务都处于等待状态，直到有线程空闲出来。只有核心线程并且核心线程没有超时限制，任务队列也没有大小限制。
   2. CachedThreadPool：一种线程数量不定的线程池，他只有非核心线程，并且其最大的线程数为Integer.MAX\_VALUE。当线程池中的线程都处于活动状态时，线程池会创建新的线程来处理新的任务，否则就会利用空闲线程来处理新的任务。
   3. ScheduledThreadPool：他的核心线程数量是固定的，而非核心线程数是没有限制的，并且当非核心线程闲置时会被立即回收。，主要用于执行定时任务和具有固定周期的重复任务。
   4. SingleThreadExecutor：这类线程池内部只有一个核心线程，确保所有的任务都在同一线程中按顺序执行。
6. 线程池的使用场景：
   1. 比如在列表中加载图片的时候，就应该使用线程池，因为列表在滑动的过程中，会产生多个线程，造成资源的浪费，此时应该使用线程池。
7. ThreadPoolExecutor执行任务遵循的原则：
   1. 如果线程池中的线程数量未达到核心线程的数量，那么直接启动一个核心线程；
   2. 如果线程池中的线程数量已经达到或者已经超过核心线程的数量，那么任务会被插入到任务队列中排队等待执行。
   3. 如果在2中无法将任务插入到任务队列中，原因是任务队列已满，这个时候如果线程数量未达到线程池规定的最大值，那么会启动一个非核心线程来执行任务。
   4. 如果3中线程数量已经达到线程池规定的最大值，那么就拒绝执行该任务。
8. 线程池按以下行为执行任务
9. 当线程数小于核心线程数时，创建线程；
10. 当线程数大于等于核心线程数，且任务队列未满时，将任务放入任务队列；
11. 当线程数大于等于核心线程数，且任务队列已满
    1. 若线程数小于最大线程数，创建线程；
    2. 若线程数等于最大线程数，抛出异常，拒绝任务。
12. Java中多线程的通信方式
13. 同步：本质上是共享内存式的通信；
14. While轮询的方式
15. Wait/notify机制
16. 管道通信

其实本质上就是两种，一种是共享内存机制，另一种是消息通信机制。

1. 字符串等于null和空字符串的区别
2. 空字符串是””,他在内存中分配了地址，只是字符串的长度为0；
3. 字符串等于null：没有在内存中分配地址，调用字符串的toString方法会发生空指针异常。
4. Wait()和sleep（）的区别
5. 这两个方法来自不同的类，sleep来自Thread类，而Wait来自Object类；

Sleep是Thread类的静态方法，谁调用sleep类，谁就去睡觉，比如说，在线程a调用线程b的sleep方法，实际上还是a去睡觉，要让线程b睡觉要在b的代码中调用sleep。

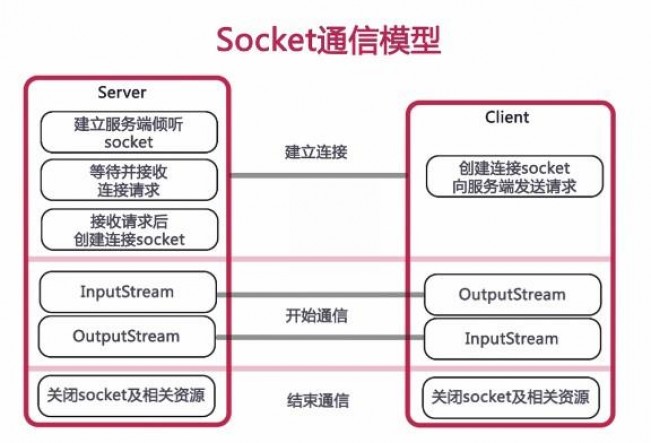
1. Sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可以使用同步控制块或者方法。Sleep不让出系统资源，wait是进入线程等待池等待，出让系统资源。
2. Wait 、notify、notifyAll只能在同步控制方法或者同步块中使用，sleep方法可以在任何地方使用。
3. Callable和Runnable的区别
4. Callable规定的方法是call，Runnable规定的方法是run;
5. Callable的任务执行后可返回值，而Runnable的任务是不能返回值的；
6. Call方法可以抛出异常，run方法不可以；
7. 运行Callable方法可以拿到一个Future对象，Future表示异步计算的结果。他提供了检查计算是否完成的方法，以等待计算的完成，并获取计算的结果。
8. Run（）方法和start（）方法的区别
9. 系统调用线程类的start方法来开启一个线程，是线程处于就绪状态，而非运行状态，此时也就意味着这个线程可以被JVM来调度，在调度过程中，JVM通过调用线程类的run方法来完成实际的操作，当run方法结束后，线程就会被终止。
10. 如果线程直接调用线程类的run方法，会被当成一个普通方法，程序中仍然只要一个线程。Start方法能够异步的调用run方法，直接调用run方法却是同步的，无法达到多线程的目的。
11. 线程的终止
12. run方法执行完毕
13. 调用stop方法(不建议使用，会产生不可预料的结果)
14. 使用interrupt方法中断线程。
15. 出现异常，异常中断
16. Break和continue的区别：
17. Break：退出循环，直接执行循环外的语句；对于for循环来说，就是跳过for循环，直接执行循环外的语句；
18. Continue：退出本次循环，执行下次循环。对于for循环来说，就是将循环条件加1，再次进行for循环。
19. 继承和组合之间的关系
20. 继承是一种is-a的关系，而组合是一种has-a的关系，比如三个类：交通工具、汽车、轮子；汽车和交通工具之间就是继承关系，而汽车和轮胎之间是组合关系；
21. 在Java中，只支持单继承，如果想实现继承两个类或者多个类，是不能实现的，所以在Java中尽量用组合和接口不用继承。
22. Clone的作用
23. 由于在Java中，“=”赋值时采用引用传递，所有不能从已有对象A创建出一个与A具有相同状态的对象B，并且对象B的修改不会影响到A的状态；
24. 为了实现上述的需求，Object类中提供了一个clone方法，这个方法返回一个Object对象的赋值，返回的是一个新的对象而不是一个引用。
25. 此处引申出一个浅复制与深复制
    1. 当一个对象中只含有基本数据类型的时候，复制对象只需要进行浅复制即可；
    2. 当一个对象中不仅含有基本数据类型还含有其他对象时，复制对象不仅要进行浅复制，还有对包含的对象进行复制，此时称为深复制。
26. Java反射
    * + 1. 定义：反射机制是在运行状态中，对于任意一个类，都能知道类的所有属性和方法，对于任意一个对象，都能够调用他的任意一个方法和属性，这种动态获取的信息以及动态调用对象的方法的功能称为Java语言的反射机制。
        2. 反射机制的作用：在运行时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；在运行时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对象的方法；生成动态代理。

反射就是把Java类中的各种成分映射成相应的Java类；一个类中的组成部分：成员变量、方法、构造方法、包等等信息都可以用java类来表示，它们是Field、Method、Contructor、Package。

1. 得到一各对象所属的类，获取一个类的所有成员变量和方法，在执行时创建对象，在运行时调用对象的方法；
2. 获取Class类的方法：
   1. Class.forName(“类的路径”)
   2. 类名.Class
   3. 实例.getClass
3. 9种基本的数据类型和他们的封装类
4. 基本数据类型：boolean、byte、char、int、short、long、double、float、void
5. 封装类：Boolean、Byte、String、Character、Short、Long、Double、Float、Void
6. 基本类型与封装类型的区别：
   1. 基本数据类型只能按值传递，封装类型则按引用传递；
   2. 基本数据类型存储在栈中的局部变量表中，封装类型的引用也存储在局部变量表中，而封装类型的实例存储在堆中。
7. Java中的自动装箱与拆箱
8. 自动装箱的目的是将原始类型值自动地转换成对应的对象；
9. 自动装箱就是Java自动将原始类型值转换成对应的对象，比如将int的变量转换成Integer对象，将Integer对象转换成int类型的值，这个过程叫做拆箱。
10. 自动装箱时编译器调用valueOf将原始类型的值转换成对象，同时自动拆箱时，编译器通过调用类似的intValue()、doubleValue()这类的方法将对象转换成原始类型的值。
11. 自动装箱发生在两种情况下，一种是赋值时，一种是方法调用时。
12. try catch finally，try里有return，finally还执行么？
13. 不管有木有出现异常，finally块中代码都会执行；
14. 当try和catch中有return时，finally仍然会执行；
15. Finally是在return后面的表达式运算后执行的（此时并没有返回运算后的值，而是先把要返回的值保存起来，管finally中的代码怎么样，返回的值都不会改变，仍然是之前保存的值），所有函数返回值是在finally执行前确定的；
16. Finally中最好不要包含return，否则程序会提前退出，返回值不是try或catch中保存的返回值。
17. Java为什么是平台无关性的

JVM是Java平台无关的基础，在JVM上，有一个Java解释器用来解释Java编译器编译后的程序。Java编程人员在编写完软件后，通过Java编译器将Java源程序编译为JVM的字节代码。任何一台机器只要配备了Java解释器，就可以运行这个程序，而不管这种字节码是在何种平台上生成的（过程如图1所示）。

1. Socket通信的流程



1. 创建一个服务器端socket，并绑定端口；
2. 服务器端开始监听，等待客户端的连接；
3. 创建一个客户端socket，并绑定服务器的IP地址以及端口号；
4. 客户端获取输出流向客户端发送消息；
5. 服务器端获取输入流，读取客户端发送的消息；并获取输出流，向客户端发送消息；
6. 最后通信完毕之后，关闭socket。
7. Java多线程之间的通信方式：
8. 同步是指多个线程通过synchronized关键字这种方式实现线程间的通信；这种方式，本质上是“共享内存”式的通信，多个线程需要访问同一个共享变量，谁拿到了锁，谁可以执行。
9. While轮询方式：在这种方式中，线程A不断地改变条件，线程B不停的通过while语句检测这个条件是否成立，从而实现线程之间的通信。但是这种方式会浪费CPU资源。
10. Wait和notify机制
11. 为什么遍历时用Iterator不用for
12. 从数据结构角度分析,for循环适合访问顺序结构,可以根据下标快速获取指定元素.而Iterator 适合访问链式结构,因为迭代器是通过next()和Pre()来定位的.可以访问没有顺序的集合.
13. 而使用 Iterator 的好处在于可以使用相同方式去遍历集合中元素，而不用考虑集合类的内部实现（只要它实现了 java.lang.Iterable 接口），如果使用 Iterator 来遍历集合中元素，一旦不再使用 List 转而使用 Set 来组织数据，那遍历元素的代码不用做任何修改，如果使用 for 来遍历，那所有遍历此集合的算法都得做相应调整,因为List有序,Set无序,结构不同,他们的访问算法也不一样.
14. 死锁

操作系统中有若干进程并发执行， 它们不断申请、使用、释放系统资源，虽然系统的进程协调、通信机构会对它们进行控制，但也可能出现若干进程都相互等待对方释放资源才能继续运行，否则就阻塞的情况。此时，若不借助外界因素， 谁也不能释放资源， 谁也不能解 除阻塞状态

死锁是指两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，他们都将无法推进下去。举例子就是独木桥，A占了独木桥的一端，B占了独木桥的另一端，A要占B占的资源，B要占A占的资源，彼此都不退让，此时就产生了死锁。

产生死锁的条件：

1. 互斥条件：即某一个资源同一时刻只能被一个进程占有，不能同时被两个或两个以上的进程占有。
2. 不可抢占条件：进程所获得的资源在未使用完毕之前，资源申请者不能强行的从资源占有者手中夺取资源，而只能由该资源的占有者进程自行释放。
3. 占有且申请条件：进程至少已经占有一个资源，但有申请新的资源，由于新申请的资源已被另外的进程占有，此时，该进程阻塞，在等待新资源之时，仍继续占有已占有的资源。
4. 循环等待条件：存在一个进程等待序列{p1,p2,…pn},其中P1等待p2所占有的资源，p2等待p3占有的资源，而pn等待p1锁占有的资源，形成一个进程循环等待。
5. 死锁的产生原因
6. 因为系统资源不足
7. 进程运行推进的顺序不合适
8. 资源分配不当
9. 死锁的预防

死锁的预防是通过设置某些条件，去破坏产生死锁的四个必要条件中的一个或者几个，来预防死锁。但是由于所施加的限制条件往往太严苛，可能会导致系统资源利用率和系统吞吐量下降。

1. 打破互斥条件：允许进程同时访问某些资源，但是有些资源是不允许同时访问的，如打印机资源，所以这个方法的实用性并不好。
2. 打破不可抢占条件：允许进程强行从占有者那里夺取某些资源，
3. 打破占有且申请条件：可以实行资源预先分配策略，进程在运行前一次性地向系统申请他所需要的全部资源。
4. 打破循环等待条件，实行资源有序分配策略，把资源事先分类编号，按号分配，使进程在申请资源时不会形成环路。
5. Throwable Error与Exception的区别

Throwable是Error和Exception类的父类

Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢出等，对于这种错误导致的应用程序中断，仅靠程序本身无法恢复和预防，遇到这样的错误，建议让程序终止。

Exception类表示程序可以处理的异常，可以捕获且可能恢复，应该尽量处理这种异常，使程序恢复运行。

常见的异常;

ArrayIndexOutOfBoundsException 数组下标越界异常，

ArithmaticException 算数异常 如除数为零

NullPointerException 空指针异常

IllegalArgumentException 不合法参数异常

1. hashCode 与equals的区别
2. hashCode方法主要用于索引，判断两个对象的索引位置是否相同。
3. equals方法用于判断两个对象的内容是否相同。
4. equals方法返回true，则hashCode方法也返回true
5. hashCode方法返回TRUE，但是equals方法未必返回TRUE。
6. HashCode的作用
7. hashCode的主要作用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。
8. 如果两个对象相同，就适用于equals方法，那么这两个对象的hashCode一定相同；
9. 如果对象的equals方法被重写，那么对象的hashCode方法也尽量重写，并且产生hashCode使用的对象，一定要和equals方法中使用一致。
10. 两个对象的hashCode值相等，并不能说明这两个对象相等，并不试用于equals方法。
11. Hash的原理

Hash是在记录的存储位置与记录的关键字之间建立一种确定的关系f，每个关键字key对应一个存储位置。

Hash表的原理：利用hash技术将记录存储在一块连续的存储空间中，这块连续的存储空间称为Hash表。

1. 怎么解决hash冲突
   * 1. 开放地址法

基本思想是当关键字key的hash地址p出现冲突，以p为基础，再建立一个hash地址p1.如果p1还冲突，则以p1为基础再建立一个p2，直到出现一个不冲突的地址，将元素存入其中。

常用的再hash的形式： h = (h +di) % m;m为表长，di为增量序列。

有三种增量序列的取值方式：

线性探测再hash

二次探测再hash

伪随机探测再hash

* + 1. 再哈希法

这种方法是同时构造多个不同的hash函数，如果哈希地址1发生冲突，再用第二个hash函数计算。

* + 1. 链地址法

基本思想是将所有哈希地址为i的元素够狠才能一个称为同义词链的单链表，并将单链表的表头存储在第i个单元中，因而查找和删除要在同义词链中进行，适用于经常进行查找和删除的情况。

* + 1. 建立公共溢出区

基本思想是将哈希表分为公共区和溢出区两部分，凡是和基本表的元素发生冲突的元素，一律放进溢出表中。

1. Java泛型中T和？的区别
2. ？和T都表示不确定的类型，但如果是T的话，函数里面可以对T进行操作。
3. 泛型的三种用法：
   1. ArrayList<T> list = new ArrayList<T>();指定集合元素只能是T类型；
   2. ArrayList<?> list = new ArrayList<?>()；集合元素可以是任意类型，这种没有意义，一般方法中，只是为了说明用法；
   3. ArrayList<? Extends T> list = new ArrayList<? Extends T>();

？ extends T ：接收T类型或者是T的父类型

？ super T：接收T类型或者T的父类型

1. Java中import的用法
2. Import导入的时候分为单类型导入和按需类型导入。单导入是在给出了具体要导入哪个类，如java.util.Scanner;按需类型导入如java.util.\*;
3. 按需导入不是导入一个包下的所有类，只是按需进行导入，也就是说他并非导入整个包，而仅仅导入当前类需要使用的类。
4. 单类型导入和按需导入对类文件定位的算法不一样，Java编译器会从启动目录，扩展目录和用户类路径下定位需要导入的类，这些目录仅仅是给出了类的顶层目录，编译器的类文件定位方法大致可以理解为：
   1. 顶层路径名 / 包名/文件名.class = 绝对路径
   2. 对于单类型导入，包名和文件名都已经确认，可以一次性查找定位。
   3. 对于按需类型导入则比较复杂，编译器会把包名和文件名进行排列组合，然后对所有的可能性进行类文件查找定位。

按需进行导入不会影响Java的性能，只会影响Java的编译速度。

1. Java中import语句是在什么时候
   1. Java中import语句并没有任何关联的运行时行为，import并不导致类被加载，他的存在时为了方便写代码，让大家把别的package的名字引入到当前源码文件中直接使用。在Java源码编译器进行编译的时候，每个名字都会进行解析找到其全名，等到class文件交给JVM的时候，JVM根本看不到Import语句。在源码中靠import写省略package的名字，从JVM来看一样都是全名。
2. LRUCache的实现

LruCache中LRU算法的实现是通过LinkedHashMap来实现的。LinkedhashMap继承是hashMap，他使用了一个双向链表来存储Map中的Entry的顺序关系，这种顺序有两种，一种是LRU顺序（访问顺序），一种是插入顺序，可以通过LinkedHashMap的三个参数的构造函数指定，三个参数分别是初始容量，加载因子大小，访问顺序。对于get、put、remove等操作，LinkedHashMap除了要做HashMap要做的事情，还要做一些调整Entry顺序链表的工作。LruCache将linkedhashmap中entry顺序关系设置为访问顺序，每次调用get，则将该对象移到链表的尾端，调用put插入新对象也在链表的尾端，这样当内存缓存达到设定的最大值，将链表头部的对象移除。

1. 网络服务器高并发编程的几种方案:
   * 1. 多进程
        + 1. 优势1：具有很好的可靠性，其中一个进程挂掉后，系统在整体上仍可以正常运行。
          2. 优势2：资源回收简单，close一个进程后，所占资源能尽快回收，简单粗暴。
          3. 劣势1：资源开销大，进程占用较多资源，而且调度开销大；
          4. 劣势2：进程通信麻烦，一般通过share memory进行；
          5. 代表语言：PHP
     2. 多线程
        + 1. 优势1：编程简单，在每个线程中处理一个客户的连接，同步阻塞处理I/O，不易出错；
          2. 优势2：线程直接通信简单；
          3. 劣势：一个线程约占1M左右的堆栈空间，创建、调度以及销毁的开销仍然很大。
          4. 代表语言：Java
     3. 异步I/O
        + 1. 优势1：资源利用率高，CPU可以持续运转；
          2. 优势2：系统开销低，可以同时创建大量的连接
          3. 劣势1：编程复杂，业务流程被划分成多个处理片段，对问题反应不自然；
          4. 劣势2：单核应用，不能很好的利用多核的优势
          5. 代表语言：node.js
     4. 协程
        + 1. 优势1：协程是非常轻量的线程（4K左右），调度非常快，通过系统线程可以并行运行在多核上；
          2. 协程中I/O阻塞时，会自动交出CPU执行权 ，可以采用在协程中同步阻塞式编程；
          3. 劣势1：GC性能相对JVM较低，但会得到根本解决；
          4. 代表语言：golang
2. 内存泄漏和内存溢出

内存溢出：out of memory 是指程序在申请内存的时，没有足够的内存空间供其使用，出现out of memory。比如申请了一个int类型的空间，但是却存了一个long型的数据，就是内存溢出。当应用的堆资源超过了Dalvik虚拟机分配的内存就会内存溢出。

内存泄漏：memory leak，是指程序在申请内存后，无法释放已经申请的内存空间，一次内存泄漏危害可以忽略不计，但是内存泄漏堆积后果很严重，无论多少内存迟早会被占光，内存泄漏最终会导致内存溢出。为什么无法释放已经申请的空间，这是因为当一个对象已经不再需要使用了，本该被回收时，而又另外一个正在使用的对象持有他的引用，从而导致了对象不能被GC回收。

# 数据结构与算法

* + 1. 用两个栈实现队列：

思路：栈是先进后出的结构，队列是先进先出的结构，用两个栈来实现队列，首先实现入队的功能，入队比较简单直接向栈A中push元素即可，然后实现出队，出队时首先判断栈B是否为空，不为空直接将栈B pop即可，假如栈B为空，则将栈A中的元素全部添加到栈B中，然后再调用栈B的pop方法。

* + 1. 两个队列实现栈

首先定义两个队列，分别是q1和q2；出栈的操作是将q1中的前n-1个元素，依次放进q2中，然后将q1中的第n个元素出栈；入栈操作是判断q1和q2队列中哪个队列不为空，就将其放入哪个队列中。

* + 1. 求解一个二叉树的深度

采用递归的方式求解；二叉树的深度等于左子树的深度 和右子树深度的最大值加1；不断的递归，递归终止的条件是根节点为空。

* + 1. 求解旋转数组中出现的最小的数字

思路就是旋转数组肯定由两部分组成，两个递增序列。用3个指针，一个向第一个元素，一个指向中间元素，一个指定最后一个元素，假如中间元素大于第一元素，则中间元素属于前面的递增序列，则最小数字位于后面的递增序列，将中间指针赋值给第一个指针，再次进行上述过程。假如中间元素小于最后元素，则中间元素位于后面的递增序列，则将中间指针赋值给最后一个指针。

* + 1. 有两个链表，如何判断他们是否相交：

如果两个链表相交的话，那么其最后的节点一定是相同的。

或者是先求出两个链表的长度，然后长的链表先走两个长度之差步，然后再一起走，没走一步判断一下两个链表的节点是否相同

* + 1. 1-100个数，随机的输出这100个数，不能重复，怎么实现。

将100个数放到数组中，先随机产生一个1-100的随机数，然后输出对应下标的数字，然后数组大小减1，再随机产生一个1-99的随机数，然后再随机输出对应下表的数字，如此重复循环。

* + 1. 判断一个字符串是不是合法的IPV4地址
    2. 先增后减序列求峰值

采用二分查找，找到中间的下标mid，如果a[mid] > a[mid+1] and a[mid] > a[mid -1]，则返回a[mid];如果a[mid] > a[mid -1 ] ,则最大值在右边，如果a[mid] > a[mid +1 ] 则最大值在左边，不断进行下去，知道找到为止。

* + 1. 找零钱问题
    2. 链表反转

ListNode pre = null;

ListNode next = null;

While(head != null){

Next = head.next;

Head.next = pre;

Pre = head;

Head = next;

}

Return pre;

* + 1. 合并区间

给出若干闭合区间，合并所有重叠的部分。

思路是按照区间的开始值对区间进行排序，然后依次判断第一个区间的终止值与第二个区间的开始值之间的关系，进行合并。注意一种情况就是可能第一个区间完全包含第二个区间，如[1,4],[2,3]。题目原题是在LeetCode上 merge-intervals。

* + 1. 字符串中第一个只出现一次的字符

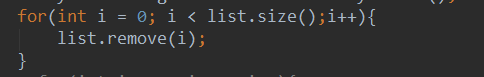
可以用一个hashMap将字符出现的次数存储起来，然后再遍历字符串，找到第一个是1的字符。

* + 1. 二叉树中，找到距离最远的两个节点的距离

思路：遍历每个节点，找出以当前节点为跟的最长路径，然后找出所有最长路径中的最大值。类似于最大路径和

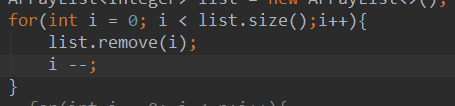
* + 1. 删除List中某一个Object

如果写成



当删除第i个时，后面元素的索引以及List的长度都会变；会出错

应该写成：



删除完成之后，将索引值减1；

* + 1. 二叉树的中序非递归实现

碰到一个节点p，如果p非空，则将p入栈，让p等于他的左孩子；

如果p为空，出栈栈顶节点，并访问该节点，然后让p等与栈顶节点的又孩子

* + 1. 使用3个线程依次打印1-30；

代码如下

package thread;  
  
*/\*\*  
 \* Created by Administrator on 2017/5/1.  
 \*/*public class Thread1 implements Runnable{  
 static int *i* = 1;  
 int id;  
  
 public Thread1(int id) {  
 this.id = id;  
 }  
  
 @Override  
 public void run() {  
 synchronized (Thread1.class){  
   
 while (*i* <= 30){  
 if (*i* % 3 == id){  
 System.*out*.print("id" + id + ":");  
 System.*out*.print(*i* ++ + ",");  
 System.*out*.println();  
 Thread1.class.notifyAll();  
 }else {  
 try {  
 Thread1.class.wait();  
 }catch (InterruptedException e){  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
 }  
 public static void main(String args[]){  
 new Thread(new Thread1(0)).start();  
 new Thread(new Thread1(1)).start();  
 new Thread(new Thread1(2)).start();  
 }  
}

* + 1. 使用三个线程，将123依次打印10次

1. package thread;  
     
   */\*\*  
    \* Created by Administrator on 2017/5/1.  
    \*/*public class Thread1 implements Runnable{  
    static int *i* = 1;  
    int id;  
     
    public Thread1(int id) {  
    this.id = id;  
    }  
     
    @Override  
    public void run() {  
    synchronized (Thread1.class){  
     
    while (*i* <= 30){  
    if (*i* % 3 == 0){  
    System.*out*.print(3);  
    Thread1.class.notifyAll();  
    *i*++;  
    }else if(*i* % 3 == 1){  
    System.*out*.print(1);  
    Thread1.class.notifyAll();  
    *i*++;  
    }else if(*i* % 3 == 2){  
    System.*out*.print(2);  
    Thread1.class.notifyAll();  
    *i*++;  
    }  
    else {  
    try {  
    Thread1.class.wait();  
    }catch (InterruptedException e){  
    e.printStackTrace();  
    }  
    }  
    }  
    }  
    }  
    public static void main(String args[]){  
    new Thread(new Thread1(0)).start();  
    new Thread(new Thread1(1)).start();  
    new Thread(new Thread1(2)).start();  
    }  
   }
2. 手写一个算法题，对字符串的处理，分别统计出任意一个字符串中所出现的字符的个数
3. 有两个链表，如何判断它们是否相交
4. 有两个单链表（不存在环），不借用任何其他数据结构，怎么遍历一次就判断是否相交

# 海量数据处理

## 有一个1GB大小的文件，里面的每一行是一个词，词的大小不超过16个字节，内存限制大小是1MB。返回频数最高的100个词。

先遍历1G的大文件，按照Hash%1000取模将大文件映射成1000个小文件，保证了相同的词分到同一个小文件中，这样每个小文件大概1M大(也可以分多些小文件，这样每个文件大小就更小)，就可以放入内存中。在每个小文件中使用Hash\_map，以单词为key，出现的次数为value，然后用最小堆得到每个小文件出现频数最多的100个单词，最后归并排序，就可以得到了

## 有10个文件，每个文件1GB，每个文件的每一行存放的都是用户的query，每个文件的query都可能重复。按照query的频度排序。

1. 顺序读取10个文件，按照hash(query)%10的结果将query写入到另外10个文件（记为）中。
2. 这样新生成的文件每个的大小大约也1G（假设hash函数是随机的）。
3. 找一台内存在2G左右的机器，依次对用hash\_map(query, query\_count)来统计每个query出现的次数。
4. 利用快速/堆/归并排序按照出现次数进行排序。
5. 将排序好的query和对应的query\_cout输出到文件中。这样得到了10个排好序的文件（记为）。
6. 对这10个文件进行归并排序（内排序与外排序相结合）。

## .海量日志数据，提取出某日访问百度次数最多的那个IP。

方案1：首先是这一天，并且是访问百度的日志中的IP取出来，逐个写入到一个大文件中。注意到IP是32位的，最多有clip\_image018个IP。同样可以采用映射的方法，比如模1000，把整个大文件映射为1000个小文件，再找出每个小文中出现频率最大的IP（可以采用hash\_map进行频率统计，然后再找出频率最大的几个）及相应的频率。然后再在这1000个最大的IP中，找出那个频率最大的IP，即为所求。

# 设计模式

1. java里面单例模式通常是怎么实现的，各自有哪些特点

单例模式的实现：

1. 构造函数一般不对外开放，为private；
2. 通过一个静态方法返回单例类对象；
3. 确保单例类的对象有且只有一个，尤其是在多线程的情况下；
4. 饿汉模式

public class Singleton{

private static Singleton singleton = new Singleton();

private Singleton(){}

public static Singleton getInstance(){

return singleton;

}

}

1. 懒汉模式

public class Singleton{

private static Singleton instance;

private Singleton(){}

public static synchronized Singleton getInstance(){

if(instance == null){

instance = new Singleton();

}

return instance;

}

优点：能在多线程的情况下保证单例对象的唯一性。

1. Double Check Lock(DCL)实现单例

public class Singleton{

private static Singleton instance;

private Singleton(){}

public static Singleton getInstance(){

if(instance == null){

synchronized（Singleton.class）{

if(instance == null){

instance = new Singleton();

}

}

}

return instance;

}

优点：既能在需要时进行初始化，又能保证线程安全，且单例对象初始化后调用getInstance不进行同步。

1. 适配器模式
   1. 定义：将一个类的接口转换成客户希望的另一个类的接口，Adapter模式使得原本由于接口不兼容不能一起工作的类可以一起工作。
   2. 有三个部分组成：
      1. Target：目标角色，就是期望得到的接口；
      2. Adapter：通过在内部包装一个Adaptee对象，把源端口转换成目标接口。
      3. Adaptee：现在需要适配的接口；
2. public class Target  
   {  
    public void request(){  
    System.*out*.println("普通请求");  
    }  
   }
3. public class Adaptee  
   {  
    public void specialRequest(){  
    System.*out*.println("特殊请求");  
    }  
   }
4. public class Apater extends Target {  
    private Adaptee adaptee = new Adaptee();  
     
    @Override  
    public void request() {  
    adaptee.specialRequest();  
    }  
    public static void main(String args[]){  
    Target target = new Apater();  
    target.request();  
    }  
   }
   1. 应用场景

Adapter通常应用于进行不兼容的类型转换的场景，还有一种就是输入有无数中情况，但是输出类型是同一的，我们可以通过Adapter返回一个同一的输出。

* + 1. Android中的ListView等的适配器，

ItemView是各种各样的，但最终都是View类型，ListView只需要知道getView返回是一个View即可，具体是什么View类型不需要ListView关心。

# JVM

1. Jvm运行时内存区域的分配

主要分为程序计数器、虚拟机栈、本地方法栈、堆、方法区。

程序计数器可以看成是当前线程所执行的字节码的行号指示器；

虚拟机栈描述的是Java方法执行的内存模型，每个方法在执行的同时会创建一个栈帧，用来存储局部变量表、操作数栈、动态链接、方法出口等信息。

本地方法栈：虚拟机栈为虚拟机执行Java方法服务，本地方法栈为虚拟机使用到的Native方法服务。

堆：Java堆是被所有线程共享的一块内存区域，用来存放对象实例。

方法区：所有线程所共享的，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。

1. Java垃圾回收机制

Java 垃圾回收的算法

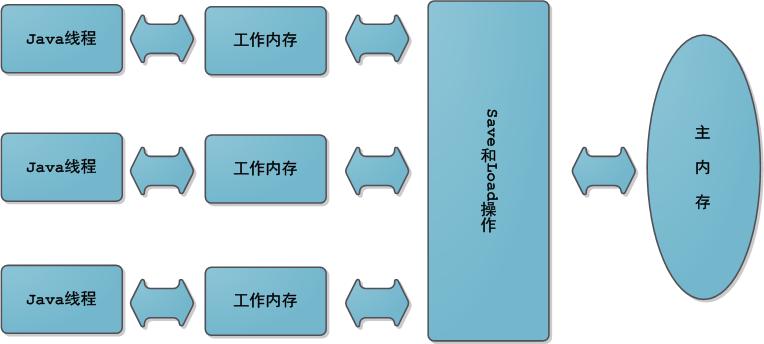
1. 标记-清除算法：首先标记出所有需要回收的对象，在标记完成后统一回收被标记的对象。
2. 复制算法：将可用内存按容量大小分成相等的两块，每次使用的时候，只使用其中的一块，每次当这一块的内存用完了，就将还存活的对象复制到另一块内存中，然后再将使用过的内存一次清理掉。
3. 标记-整理算法：首先标记出所有需要回收的对象，让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。
4. 分代收集算法：根据对象的存活周期不用将内存划分成几块，一般把Java堆分成新生代和老年代，根据各个年代的特点采用最适合的收集算法，新生代采用复制算法，老年代采用标记清理活标记整理算法。
5. 垃圾回收是在什么时候执行的
   1. 大多数情况下，对象在新生代Eden区中进行分配，当Eden区中没有足够空间进行分配时，虚拟机将发起一次Minor GC。
   2. Full GC ，发生在老年代的GC，当老年代没有足够的空间时，发生full GC，发生full GC一般都会有一次Minor GC
   3. 发生Minor GC时，虚拟机会检测老年代的剩余空间大小是否大于之前每次晋升到老年代的平均大小，如果大于，则进行Minor GC，如果小于，查看handlePromotionFailure设置是否允许担保，如果允许，只会进行minor GC，如果不允许，改为进行一次full GC。
6. 什么样的内存可以回收
7. 不可达的对象并不是非死不可的，还要进行两次标记。如果对象在进行可达性分析时，发现没有与GC roots相连接的引用链，它会第一次标记，并进行筛选，筛选条件是此对象是否有必要执行finalize方法，当对象没有覆盖finalize（）方法，或者finalize方法已经被虚拟机调用过，都认为没有必要执行。
8. 对于没有必要执行finalize方法的将会被GC.
9. 垃圾回收干了什么
10. 删除不使用的对象，腾出内存空间。
11. 停止其他线程，运行finalize
12. 新生代做的是复制清理，from survivor to survivor。老年代做的是标记整理，
13. Java的四种引用
14. 强引用：在程序代码中普遍存在的，如Object object = new Object();只要强引用还存在，垃圾收集器永远不会回收掉被引用的对象。
15. 软引用：如果一个对象具有软引用，则内存空间足够，垃圾回收器就不会回收它，如果内存空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存。

用来描述一些还有用但并非必需的对象，对于软引用关联着的对象，在系统将要发生内存溢出之前，将会把这些对象列入回收范围之中进行第二次回收。

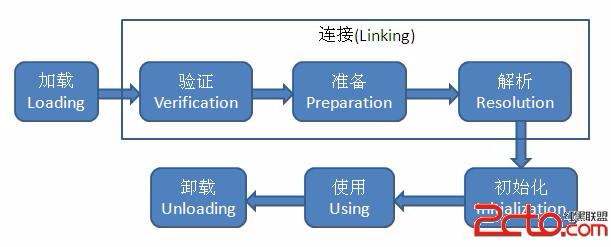
1. 弱引用：描述非必需对象，他的强度比软引用更弱一些，被弱引用关联的对象只能生存到下一次垃圾收集发生之前。当垃圾收集器工作时，无论当前内存是否够用，都只会回收掉只被弱引用关联的对象。
2. 虚引用：为一个对象设置虚引用的目的是能在这个对象被收集器回收时收到一条系统通知，用来跟踪对象被垃圾回收器回收的活动。
3. XMs 、Xmx、Xmn

Xms：初始堆大小，Xmx：最大堆大小，Xmn：年轻代大小

1. volatile和Synchronized区别
2. volatile:主要用于在多线程中同步变量
3. volatile关键字能够保证此变量对所有线程的可见性，这个可见性是指当一条线程修改了这个变量的值，新值对于其他线程来说是立即得知的。
4. 一般情况下，为提高性能，每个线程运行时都会将主内存中的变量保存一份在自己的内存中中作为变量副本，但是这样就很容易出现多个线程中保存的副本变量不一致，或与主内存中的变量不一致，当一个变量被volatile修饰之后，不允许有不同于主内存变量的拷贝。Volatile只能保证操作是在一块内存中，并不能保证操作的原子性。
5. synchronized关键字是Java利用锁的机制自动实现的，一般有同步方法和同步方法块两种使用方式，Java中所有的对象都自动含有单一的锁，当在对象上调用其任意的synchronized方法时，此对象被加锁，同时在线程从该方法返回之前，该对象内其他所有要代用类中被标记为synchronized的方法的线程都会被阻塞。
6. 原子性、可见性的概念
7. 原子性：由Java内存模型来直接保证的原子性变量操作包括read、load、assign、use、store和write，我们可以认为基本的数据类型的访问读写是具备原子性的。
8. 可见性：可见性是指当一个线程修改了共享变量的值，其他线程能立即得知这个修改。Java内存模型是通过在变量修改后将新值同步到主内存，在变量读取前从主内存刷新变量值这种依赖主内存做为传递媒介的方式来实现可见性。
9. 有序性：如果在本线程内观察，所有的操作都是有序的，如果在一个线程中观察另一个线程，所有操作都是无序的。
10. JVM如何分区，每个区都是干嘛的？
11. JVM分为方法区、虚拟机栈、本地方法栈、堆、程序计数器
12. 方法区：线程共享的区域，用于存储已被虚拟机加载的类信息、常量、静态变量、即时编译后的代码数据；
13. 虚拟机栈：线程私有的，描述的是Java方法执行的内存模型；为虚拟机执行Java方法服务；
14. 本地方法栈：为虚拟机使用到的Native方法服务；
15. 程序计数器：当前线程所执行的字节码的行号指示器，线程私有的，
16. 堆：线程共享的，存放对象实例。
17. Java内存模型 ，他问我工作线程的内存如何释放
18. Java内存模型：Java内存模型规定了所有的变量都存储在主内存中，每条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（读取、赋值）都必须在工作内存中进行，而不能直接读写主内存中的变量，不同的线程之间也无法直接访问对方工作内存的变量，线程间值的传递需要通过主内存来完成。



1. Java中的类加载过程



1. **加载：是类加载过程的一个阶段**

**在加载阶段，java虚拟机需要完成以下3件事：**

**a.通过一个类的全限定名来获取定义此类的二进制字节流。**

**b.将定义类的二进制字节流所代表的静态存储结构转换为方法区的运行时数据结构。**

**c.在java堆中生成一个代表该类的java.lang.Class对象，作为方法区数据的访问入口。**

**加载阶段与连接阶段是交叉进行的，加载阶段尚未完成，连接阶段可能已经开始，这些夹在加载阶段之中进行的动作仍然属于连接阶段，加载和连接阶段仍然保持着固定的先后顺序。**

1. **验证：是连接阶段的第一步，目的是为了确保Class文件的字节流中包含的信息符合当前虚拟机的要求，并且不会危害虚拟机自身的安全。因为Class文件并不一定要求用java源码编译而来。**

**验证阶段大致上会完成下面4个阶段的检验动作：文件格式验证、元数据验证、字节码验证、符号引用验证**

**a.文件格式验证**

**可能的验证点：魔数是否以0xcafebabe开头；主次版本号是否在当前虚拟机的处理范围内；常量池中常量是否有不支持的类型；**

**验证点远远不止这些。验证的主要目的是保证输入的字节流能正确的解析并存储于方法区内，格式上符合描述一个Java类型信息的要求。**

**b.元数据验证**

**是对字节码描述的信息进行语义分析，以保证其描述的信息符合Java语言规范的要求；**

**可能的验证点：这个类是否含有父类；类是否继承了不允许被继承的类（被final修饰的类）；不是抽象类，是否实现了父类或者接口中方法；类中的字段。方法是否与父类产生矛盾。**

**目的是对类的元数据信息进行语义校验，保证不存在不符合java语言规范的元数据信息。**

**c.字节码验证**

**是整个验证过程中最复杂的，主要进行数据流和控制流分析，如保证跳转指令不会跳转到方法体之外的字节码指令、数据类型转换安全有效等。  
    d.符号引用验证**

**发生在虚拟机将符号引用转换为直接引用的时候。符号引用验证的目的是确保解析动作能正常执行，如果无法通过符号引用验证，则会抛出java.lang.IncompatibleClassChangeError异常的子类异常。**

1. **准备：是正式为类变量分配内存并设置初始值的阶段，这些变量所使用的内存都将在方法区中进行分配。**

**这个时候进行内存分配的仅包括类变量（被static修饰的变量），而不包括实例变量。实例变量将会在对象实例化的时候随着对象一起分配在java堆中。**

1. **解析：解析阶段是虚拟机将常量池内的符号引用替换成直接引用的过程。**

**符号引用：以一组符号来描述所引用的目标，符号可以是任何形式的字面量，只要能够无歧义的定位到目标即可。符号引用和虚拟机实现的内存布局无关。**

**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。直接引用和虚拟机实现的内存布局是相关的。**

1. **初始化**

**类初始化过程是类加载过程的最后一步。在这个阶段才开始执行类中定义的java程序代码。**

**在初始化阶段，变量已经赋过一次系统要求的初始值，而在初始化阶段，根据程序员通过程序制定的主观计划去初始化类变量和其他资源。**

1. 如何判断对象是否存活
2. 引用计数算法：给对象添加一个引用计数器，每当有一个地方引用它时，计数器值就加1，当引用失效时，计数器值减1,；任何时候，计数器为0的对象就是不可能再被使用的。缺点是很难解决对象之间循环引用的问题。
3. 可达性分析算法：通过一系列的称为“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没有任何引用链相连时，则证明该对象时不可用的。
4. 什么是classloader

程序在启动的时候，并不会一次性加载程序所要的所有class文件，二是根据程序的需要，通过Java类加载机制（classloader）来动态加载某个class到内存中，从而只有class文件被加载到内存后，才能被其他class所引用，所以classloader是用来动态加载class文件到内存当中用。

启动类加载器：Java类加载层次中最顶层的类加载器，负载加载JDK中的核心类库。

扩展类加载器：负责加载Java的扩展类库

应用程序类加载器：负责加载classpath目录下的所有jar和class文件。

1. 双亲委派模式

启动类加载器，扩展类加载器，应用程序类加载器，自定义的类加载器，每个类加载器再加载的时候，都不自己先加载，都先交由自己的父类去加载，最终都到启动类加载器；当父类加载器不能完成加载时，然后调用子类的类加载器去加载。

# TCP/IP

tcp和ip协议有什么区别，如果我们华为这边要和其他公司进行数据通信，请你设计一个通信方案

1. TCP/IP协议的分层

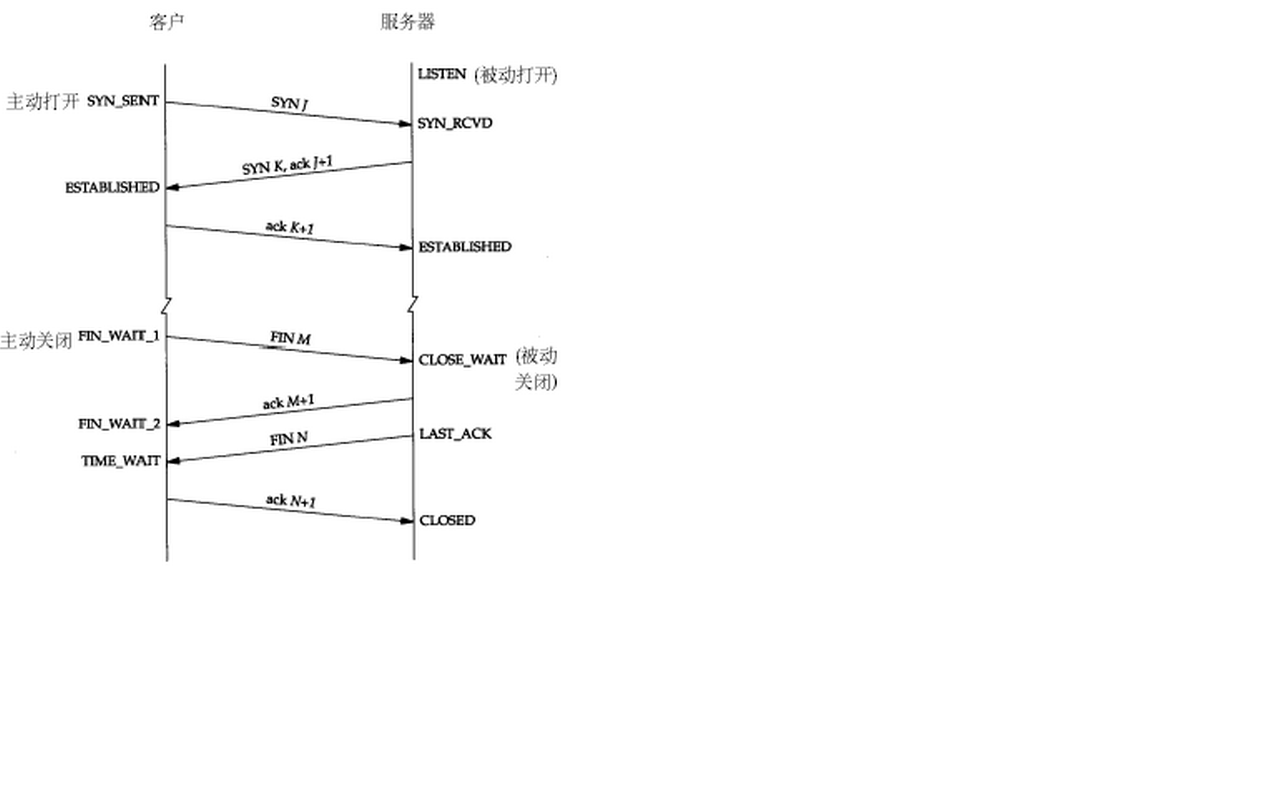
从下到上依次分为数据链路层 网络层、传输层、应用层。

1. TCP是如何来保证可靠的传输的

从以下几方面入手去分析：

* 1. 确认和重传：接收方收到报文之后会进行确认，发送方发送一段时间之后没有收到应答会重传。
  2. 数据校验
  3. 数据合理的分片和排序；TCP会按照MTU合理的分片，接收方会对分片进行重排。
  4. 流量控制
  5. 拥塞控制

1. 画一下tcp三次握手的过程，写出来状态和发送的信息



为什么需要4次挥手？因为TCP是全双工的，可以将其看成两条单工的链路，在断开链接的时候，需要一条一条的断开。假如是三次的话，B收到A的终止请求，只能说明A不给B发消息了，但是B还是可以给A发送消息的。为什么需要3次握手，是因为两次握手会有一定的不确定性，四次的话又有点浪费资源，可以将中间的两次合并。

1. TCP中的拥塞控制

在TCP中采用了慢启动、拥塞避免和加速递减等技术来进行拥塞避免。

介绍几个关键的窗口：

通知窗口：用来通知发端在未收到应答之前可以发送的最多字节数；

拥塞窗口：根据网络的拥塞情况动态的调整的。

发端真正使用的窗口等于通知窗口和拥塞窗口的最小值。

有三个过程：慢启动、拥塞避免、加速递减

1. 写一下tcp主动关闭的一方的几个状态，并且解释一下这些状态
   1. FIN\_WAIT\_1：在实际工作中很少见，主动方调用close函数关闭连接后立即进入FIN\_WAIT\_1状态；
   2. FIN\_WAIT\_2: 我们在收到了对端对FIN的确认之后，进入FIN\_WAIT\_2状态，等待对方发送FIN命令。
   3. TIME\_WAIT：我们在收到了对端的FIN命令之后进入TIME\_WAIT状态。可以让TCP再次发送最后的ACK以防这个ACK丢失。此段的时间是2倍的报文最大生存时间。
2. 为什么连接的时候是三次握手，关闭的时候是四次握手

因为当server收到client的syn连接请求报文后，可以直接发送syn+ack报文，其中ack报文是用来应答的，syn报文是用来同步的。但是关闭连接的时候，当server端收到fin报文后，很可能不会立即关闭socket，所以先回复一个ack报文，表明你发的fin命令我收到了，只有等到我server端所有报文都发送完了，我才能发送fin报文，因此不能一起发送，需要四次握手。

1. 为什么要采用三次握手，两次不行吗？

保活计时器：为了防止已失效的连接请求报文又传送到服务器，因此产生错误。

考虑下面的场景：

客户端已主动与服务器建立连接，但是后来客户端主机出现错误。

通常设置为2小时，若两小时内没有收到客户端的数据，服务器就发出一个探测报文，以后每隔75分钟发送一次，若一连发送10个探测报文后仍无客户端的响应，服务器就认为客户端出现了错误，接着就关闭这个链接。

1. 用netstat查看端口状态为什么会出现大量的TimeWait状态。

原因：是有大量的高并发的短连接；

例子：对于一个处理大量短连接的服务器，如果是由服务器主动关闭客户端的连接，将导致服务器端存在大量处于TimeWait状态的Socket。

解决办法：修改注册表中的TCPTimeWaitDelay中的值，windows默认是4分钟，可以修改为1分钟。

1. http的常用的状态有哪些，301和302的区别是什么，503是什么意思
2. 2XX系列：表明请求被正确接收了；
   1. 200 OK 表示从客户端发来的请求在客户端被正常处理了；
   2. 204 Not Content 表名服务器接收的请求已成功处理，但在返回的响应报文中不含实体的主体部分；
   3. 206 Partial Content ： 表示客户端进行了范围请求，而服务器成功执行了这部分的Get请求；
3. 3XX 重定向：表明浏览器需要执行某些特殊的处理以正确处理请求
   1. 301 Moved Permanently ： 表示请求的资源已被分配了新的URI，以后应该使用资源现在所指的URI。永久性重定向
   2. 302 ： 表示请求的资源已被分配了新的URI，希望用户（本次）能使用新的URI访问。临时起重定向
   3. 303 See Other：表示由于请求对应的资源存在着另一个URI，应使用GET方法定向获取请求的资源
4. 4XX系列：表明客户端是发生错误的原因所在
   1. 400 Bad Request:表明请求报文中存在语法错误；
   2. 401 表示发送的请求需要有通过HTTP认证的认证信息；
   3. 403 表示对请求资源的访问被服务器拒绝了
   4. 404 服务器无法找到请求的资源
5. 5XX 表示服务器本身发生错误
   1. 500 表明服务器在执行请求时发生了错误；
   2. 503 表明服务器现在处于超负载或正在进行停机维护，现在无法处理请求；
6. https和http有什么区别，区别的细节是什么样子的
7. Http的缺点：
   1. 通信使用明文（不加密），内容有可能被窃听；
   2. 不验证通信方的身份，因此可能遭遇伪装
   3. 无法证明报文的完整性，所以有可能遭篡改；
8. https = Http + 加密 + 认证 + 完整性保护
   1. Https是身披SSL的HTTP，通常HTTP是直接和TCP进行通信的，但是Https是Http先和SSL进行通信，然后再跟TCP进行通信。
   2. http和https使用的是完全不同的连接方式，用到端口也不一样，前者用的端口是80，后者用的端口是443。
9. 网络中Http get和post的区别
10. Get是向服务器方索取数据的一种请求，而POST是向服务器提交数据的一种请求。
11. Get是获取信息，而不是修改信息，类似数据库的查询功能一样，数据不会被修改。
12. POST表示可能修改服务器上的资源的请求，在服务器端，用POST方式提交的数据只能用Request.Form来获取。
13. 讲讲http协议，输入一个网址到浏览器呈现出界面的过程是什么样子的
    * 1. 输入网址后，先将域名解析为IP地址；DNS
      2. 客户端与服务端建立连接；TCP
      3. 发送Http请求；Http

对于MAC层做的就是通过ARP和RARP协议将IP地址转换成具体的MAC地址

1. TCP的长连接和短连接
2. 长连接是指在一个TCP连接上可以连续的发送多个数据包，在TCP连接保持期间，如果没有数据包发送，需要双方发检测包以维持此连接。定时发送心跳包以维持连接；
3. 短连接是指通信双方有数据交互时，就建立一个TCP连接，数据发送完成后，就断开连接。
4. 适用场景：
   1. 长连接适用于操作频繁的时候，比如数据库的连接是长连接。
   2. 短连接比如Http服务一般都是短连接。
5. 点击搜索按钮发生了什么
   * + 1. 点击搜索的那零点几秒之间，他的程序在巨大的数据库里按照关键字进行查找，计算所有关于关键字的网页。
       2. 搜索引擎先在网上找来所有信息（这个过程将永不停息的进行下去）建立一个数据库，然后对这些信息创建一个索引，当用户来搜索的时候，直接用自己归纳编排好的索引，找自己数据库里的信息反馈给用户。
6. Http报文结构：
7. http请求报文：

http请求报文由请求行、请求头部、空行、请求数据四部分组成；

* 1. 请求行： 由请求方法、URL、协议版本3个字段组成；
  2. 请求头部：由一系列的键值对组成典型的有：
     1. UserAgent：产生请求的浏览器类型；
     2. Accept：客户端可识别的内容类型列表；
     3. Host：要请求的主机名；
     4. Accept-Language：客户端可接受的自然语言；
     5. Accept-Encoding：客户端可接受的编码压缩格式；
     6. Connection：连接方式；
     7. Cookie：
  3. 空行
  4. 请求数据

1. http响应报文
   1. 响应报文由状态行、响应头部、空行和响应数据组成
   2. 状态行由Http协议版本、服务器返回的响应状态码和响应状态码的文本描述
   3. 响应头部：
      1. Location：用于重定向接受者到一个新的位置；
      2. Server：包含了服务器用来处理请求的软件信息及其版本；
      3. Connection：连接方式
2. 3、HTTP的特点
3. 4、HTTP与HTTPS的区别

# 数据库

数据库表设计，事务，锁，SQL编写，存储过程，触发器等

1. MySQL的存储引擎

MySQL 5.5之前默认的存储引擎是MyISAM，5.5之后改为了InnoDB。

1. MyISAM不支持事务，也不支持外键，优势是访问速度快，对事务的完整性没有要求或者以select、insert为主的应用基本都可以使用这个引擎来创建表；

访问速度快的原因是MyISAM将表定义、数据文件、索引文件放置在不同的目录下，平均分布IO，速度更快。

1. InnoDB存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全，InnoDB写的处理效率差一些，会占用更多的磁盘空间以保留数据和索引。

支持外键；

1. 存储引擎的使用场景
2. MyISAM适用于以读操作和插入操作为主，只有很少的更新和删除操作，并且对事务的完整性和并发性要求不是很高的情况下。
3. InnoDB用于事务处理应用程序，支持外键，对事务的完整性有比较高的要求，在并发条件下要求数据的一致性，数据操作处理插入和查询以外，还包括很多更新、删除操作的情况下。
4. 事务

事务是并发控制的基本单位，是一个操作序列，这些操作要么都执行，要么都不执行，它是一个不可分割的工作单位。如银行转账工作，从一个账号上扣款并使另一个账号上增款，这两个操作要么都执行，要么都不执行，应该把他们看成是一个事务。

事务的提出是为了解决并发情况下保持数据一致性的问题。

事务的4个基本特征：

* 原子性：事务是数据库的逻辑工作单位，他对数据库的修改要么全部执行，要么全部不执行；
* 一致性：事务前后，数据库的状态都满足所有的完整性约束；
* 隔离性：数据库提供一定的隔离机制，保证事务在不受外部并发操作
* 持久性：在事务完成后，该事务对数据库所做的更好将持久的保存在数据库中，不会被回滚。

1. 并发事务带来的问题
2. 更新丢失：当两个或多个事务选择同一行，然后基于最初选定的值更新该行时，由于每个事务都不知道其他事务的存在，就会发生丢失更新问题，最后的更新会覆盖了由其他事务所做的更新。
3. 脏读：一个事务正在对一个记录做修改，在这个事务完成并提交前，这条记录的数据就处于不一致的状态，这时，另一个事务来读取同一条记录，如果不加控制，第二个事务就读取了这些脏数据；
4. 不可重复读：一个事务在读取某些数据后的某个时间，再次读取以前读过的数据，却发现其读出的数据已经发生了改变或某些记录已经被删除了。
5. 幻读：一个事务按相同的查询条件重新读取以前检索过的数据，却发现其他事务插入了满足其查询条件的新数据。
6. 事务的隔离级别
7. 未提交读
8. 已提交读
9. 可重复读
10. 可序列化
11. 数据库索引的实现原理
12. 索引问题其实是一个查找问题，数据库索引是数据库管理系统中的一个排序的数据结构，以协助快速查找、更新数据库表中的数据，索引的实现通常使用B树及其变种B+树；
13. 为表设置索引付出的代价（用空间代价换取时间代价）
    1. 增加了数据库的存储空间，索引会占用一定的空间；
    2. 在插入和修改数据时要花费较多的时间，因为索引也要随之变动。
14. 创建索引可以大大提高系统的性能
    1. 创建唯一索引，保证数据库表中每一行数据的唯一性；
    2. 大大加快数据的检索速度（创建索引最主要的原因）
    3. 加速表和表之间的连接
    4. 使用索引可以在查询过程中，优化隐藏器，提高系统的性能；
15. 设计索引的时候，需要注意的事项
16. 在下面的这些列上创建索引：
    1. 在经常需要搜索的列上，可以加快搜索速度
    2. 在作为主键的列上，强制该列的唯一性和组织表中数据的排列结构；
    3. 在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度；
    4. 在经常需要进行排序的列上创建索引
    5. 在经常使用where子句中的列上创建索引，加快条件的判断速度；
17. 在下面的列上创建索引
    1. 对于那些在查询中很少使用的列上不应该创建索引
    2. 对于只有很少数据值的列上不应该创建索引，比如表示用户性别的列
    3. 对于定义为text、image、bit数据类型的列上不应该创建索引
    4. 当修改性能远大于搜索性能时，不应该创建索引，修改性能和检索性能时矛盾的，增加索引会提高检索性能但会降低搜索性能。
18. 数据库的优化
19. 使表在磁盘上占据的空间尽可能的小。这能带来性能上极大提升，因为数据小，磁盘读入速度较快，在更小的列上建立索引，索引也会占用更少的资源；
    1. 使用合适的数据类型，不要将数字存储成字符串；
    2. 尽可能使用最有效的数据类型；
    3. 尽可能使用最小的数据类型，比如MEDIUMINT就比INT好；
    4. 如果可能将列声明为not null；
20. 建立索引；可以提高select语句的性能；
21. 使用查询语句的时候，尽量避免使用select \* ，因为会导致全表扫描；
22. 优化insert语句:如果从同一个客户插入很多行，尽量使用多个值表的insert语句；会大大缩减客户端与数据库之间的连接、关闭等消耗，使得效率比分开执行的单个insert语句快；
23. 建表时：将索引文件和数据文件放在不同的磁盘上进行
24. 数据库的主键和外键

主键：主键是数据表的唯一索引，比如学生表中有姓名和学号，姓名有可能是重复的，但是学号是唯一的，可以将学号看成是唯一的索引。也就是主键，一般主键设置成为自增长类型。

外键：是两个表之间的一种关联关系，比如有一个电脑表，其中包含一项数据就是CPU型号，还有一个配件表，配件表中也有一个CPU型号，电脑表中的CPU型号由配件表中的CPU型号来决定，可以给两个表的CPU型号设置一个外键，将两个表关联起来。

外键用于与另一张表的关联，是能确定另一张表记录的字段，用于保持数据的一致性。

1. 查询成绩大于80分的学生，并求其平均值

Select name from table where score > 80;

求平均值用avg函数

Select avg(score) from table where name in (select name from table where score > 80)

1. 数据库子查询
2. 列子查询
   1. 查询返回的是一个数字，在使用的时候可以直接使用 = > <
   2. 查询返回的是一个列的时候，使用的时候不能用运算符，只能使用 集合操作符 in，not in，any ，all
3. 行子查询
   1. 行子查询需要限定返回的只有一行，用distinct 或者limit 1实现
   2. 不能直接将返回的行当做参数，需要用括号构建一个新的行

格式如下（name,socre） = select name，score from table where name = ‘li’ and score = 80 limit 1;

1. 表子查询

跟在from 后面 ，将返回的表当做参数，此时需要将表用as进行重命名。

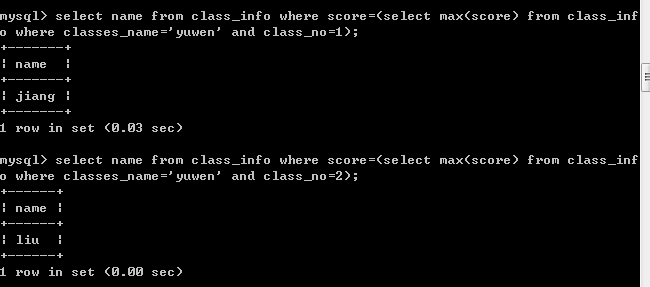
如：select name from ( select name ,score form table where name like ‘Li%’ and socre > 80) as temp where score = 85;

1. Exists（subquery）
   1. 判断依据，如果子查询可以返回数据，则认为exists表达式返回true；
2. 数据库索引
   * 1. 数据库索引是用于提高数据库表的数据访问速度的；
     2. 特点
        + 1. 避免进行数据库全表扫描，只需要扫描较少的索引页和数据页，而不是查询所有数据页。对于非聚集索引，有时不需要访问数据页即可得到数据。
          2. 聚集索引可以避免数据的插入操作，集中于表的最后一个数据页面。
          3. 在某些情况下，索引可以避免排序操作。
3. 表中含有姓名、班级。课程名、分数，找出语文分数最高的学生，求每个班语文成绩最高的学生
4. 求语文成绩最高的学生
   1. 使用max函数找出最高的成绩，
   2. 
   3. 找出分数等于max的那个学生

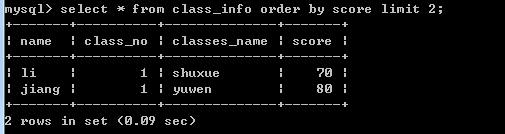


1. 找出每个班语文成绩最高的学生

在上面查询的基础之上，改变where后面的条件如下图所示



1. 如何查看最大的3个



# Linux

## 进程资源的占用情况 top

Top命令：用于显示系统中各个进程的资源占用情况。

## 内存的使用情况free

Free命令用于查看当前系统内存的使用情况

## 如何查看服务器的DNS信息

1. 查看cat /etc/resolv.com
2. 使用nslookup
3. 使用dig命令来查看DNS信息

## Traceroute命令

Traceroute命令用来跟踪数据包到达网络主机所经过的路由的工具；

traceroute 是用来发出数据包的主机到目标主机之间所经过的网关的工具。traceroute 的原理是试图以最小的TTL发出探测包来跟踪数据包到达目标主机所经过的网关，然后监听一个来自网关ICMP的应答。发送数据包的大小默认为 38个字节。

## 进程IO相关Iostat

iostat工具查看进程IO请求下发的数量、系统处理IO请求的耗时，进而分析进程与操作系统的交互过程中IO方面是否存在瓶颈。

## 守护进程

1. 守护进程是一种运行在后台的特殊进程，它独立于控制终端并且周期性的执行某种任务或等待处理某些发生的事件。大多数服务都是通过守护进程实现的。

## Libevent

1. Libevent是一个事件通知库，内部使用select、epoll、kqueue、IOPC等系统调用管理事件机制

## 进程相关：查看进程与终止进程

1. 终止进程

Kill PID 杀死指定PID的进程（）ProcessID

Kill -9 PID 杀死相关进程

1. 查询进程

ps

查询正在运行的进程 ps –ef

## 查看端口的占用情况

Netstat –l 仅列出正在监听的服务状态

如lsof –I : 端口号

## 磁盘管理查看磁盘空间大小，目录大小

1. 查看磁盘空间利用大小： df –h
2. 查看目录大小：du –sh
3. 打包： tar –cvf 这个命令只打包不压缩
4. 压缩：gzip 文件名，生成的是.gz文件
5. 解包 ： tar –xvf
6. 解压缩 gunzip bzip

## Linux中如何查找一个含有AA的行

1. 首先先查看文件内容图test.txt

Cat test.txt

1. 其次在找出所在的行

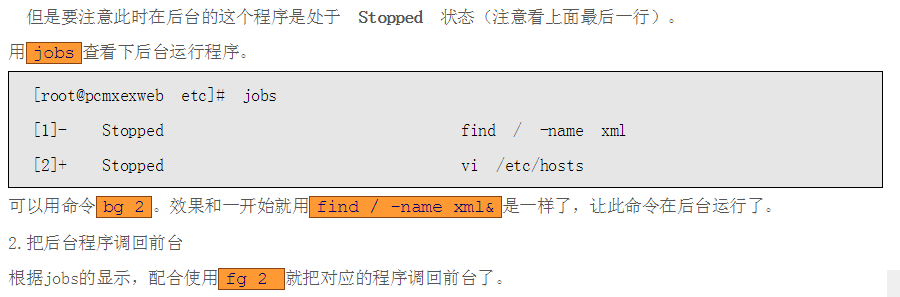
Grep –e‘AA’-e ‘BB’ ./test.txt

## Linux后台运行程序及恢复

1. 后台运行：ctrl + z 此时程序是处于stop状态的，在配合bg 2 就可以是使程序运行在后台。
2. 把后台运行的程序切换到前台：

先用jobs命令，查看后台运行的程序，然后再配合命令fg 2，就可以将对应的程序切换到前台。





## Top命令的参数意义 （P518）

动态查看进程的变化

1. -d top执行的间隔数
2. -p 监测固定PID的进程
3. -b 以批次的形式执行top，一般会将结果输出到文件中
4. -n 执行几次top

## Linux中显示一个文件的某几行

可以用管道来实现:

显示1000行到3000行，这个用到了广义表

Cat filename | head –n 3000 | tail –n + 1000

从3000行开始显示 显示1000行，也就是3000-3999

Cat filename |tail –n +3000 | head –n 1000

tail –n 1000 显示最后的1000行

tail -n +1000 从1000开始显示最后的

## 查找大文件

Find / -size +1000k;

## 在Linux下写一个统计文件中的单词数

egrep -o "\b[[:alpha:]]+\b" $filename

## Uniq命令

Uniq命令表示删除重复的行

Uniq 直接去除重复的行

Uniq –c 去除重复的行，并将重复次数显示出来。

## Netstat命令

1. Netstat命令用于显示各种网络相关信息，如网络连接、路由表、接口状态等
2. 列出所有端口：netstat –a
3. 列出所有TCP端口：netstat –at
4. 列出所有udp端口：netstat –au
5. 列出所有处于监听状态的socket： netstat –l
6. 在netstat输出中显示PID和进程名称：netstat –p

## Linux查看文本文件中出现次数最多的IP

命令： sort ip.txt | uniq –c | sort –nr | head –n 1

解释：

首先排序sort会将相同的IP排在一起，然后uniq –c会去掉重复的IP，只保留一条，同时记录重复数在行首，在按照反序排序即可，sort –nr，-n 是按数字，-r是按降序，取出第一个。

## 将文本里面的多行内容合并成一行

1. 使用sed
2. 使用awk
3. 使用cat file | xargs

## Awk命令

1. Awk是一个强大的文本分析工具
2. 可以用来过滤出一行

# 操作系统

## 虚拟内存

虚拟内存是计算机系统内存管理的一种，它使程序认为它拥有连续的可用内存。但实际上他是被分成多个物理内存碎片，还有部分存储在外部内存磁盘存储器上，在需要时进行数据交换。

## 线程间通信的方式

互斥量，条件变量，读写锁，自旋锁，屏障，原子类。

## 中断和异常

1. 中断：系统停止正在执行的程序而转向其他的服务，原因可能有优先级较高的服务到来，或者是人为的进行中断，中断属于正常现象。
2. 异常：异常是软件运行中开发没有考虑到的程序错误。

# Android问题

* 1. Activity的生命周期

OnCreate -> onstart -> onResume -> onPause -> onstop -> onDestory， onrestart

* 1. Activtiy A启动Activity B。两个Activity的生命周期

onCreateA –>onstartA –> onResumeA –>onpauseA –>onCreateB –> onStartB –> onResumeB –>onstopA

所以在onpause函数中不宜进行耗时操作。

* 1. onStart和onStop，onResume 和onPause的区别  
     onStart和onStop是从Activity是否可见这个角度回调的，而onResume和onPause是从Activity是否位于前台回调的，位于前台可以获得焦点。
  2. Activity的缓存方法 onSaveInstanceState和onRestoreInstanceState调用的过程和时机，onNewIntent方法

1. 如果你的系统长时间处于Stopped形态而且此时系统需要更多内存或者系统内存极为紧张的时候，系统就会回收你的Activity，而此时系统为例补偿你，会将Activity状态通过onSaveInstanceState（）方法保存到Bundle对象中，当你需要重新创建这些Activity的时候，保存的Bundle对象就会传递到Activity的onRestoreInstanceState（）方法与OnCreate方法中，这也就是onCreate方法中的Bundle savedInstanceState的来源。
2. 1、当用户按下HOME键时。   
   这是显而易见的，系统不知道你按下HOME后要运行多少其他的程序，自然也不知道activity A是否会被销毁，故系统会调用onSaveInstanceState，让用户有机会保存某些非永久性的数据。以下几种情况的分析都遵循该原则   
   2、长按HOME键，选择运行其他的程序时。   
   3、按下电源按键（关闭屏幕显示）时。   
   4、从activity A中启动一个新的activity时。   
   5、屏幕方向切换时，例如从竖屏切换到横屏时
3. onRestoreInstanceState与onSaveInstanceState不一定是成对出现的，onRestoreInstanceState被调用的条件是，Activity确实被系统销毁了。
4. onNewIntent方法，是在将Activtiy设置为SingleTask或SingleTop时调用。Activity A启动Activity B，但是Activtiy B 的启动模式是singleTask，此时，假如Activtiy B已经存在一个实例，就不会调用B的onCreate方法，而是直接调用onNewIntent方法，生命周期是onNewIntent 、onResume。但是假如不存在B的实例，则还是会调用onCreate方法。
   1. Activity的启动方式

Android有四种启动模式：standard、singleTop、singleTask、singleInstance。

1. standard ：每次启动Activity都会创建一个实例，不管这个实例是否存在。
2. singleTop ： 如果新的Activity已经位于任务栈的栈顶，那么次Activity不会被重新创建
3. singleTask：只要Activity在一个栈中存在，那么多次启动此Activity都不会重新创建实例。
4. singleInstance：除了具有singleTask模式的所有特性之外，还加强了一点就是，具有此模式的Activity只能单独的位于一个任务栈中。
   1. Android启动模式的应用场景
      1. singleTop适合接收通知启动的内容显示页面，例如，某个新闻客户端的新闻内容页面，如果收到10个新闻推送，每次都打开一个新闻内容页面是很烦人的。
      2. singleTask适合作为程序入口点，例如浏览器的主界面，不管从多个个应用启动浏览器，只会启动主界面一次，其余情况都走onNewIntent，并且或清空主界面上面的其他页面，之前打开的页面，直接打开，不用再新建。
      3. singleInstance适合需要与程序分离开的界面，如闹钟提醒，将闹铃提醒与闹铃设置分离，
   2. Android任务栈

1.一个应用程序一般都是由多个activity组成的。

2.任务栈（task stack）（别名back stack后退栈） 记录存放用户开启的activity的。

3.一个应用程序一被开启系统就给他分配一个任务栈，当所有的activity都退出的时候，任务栈就清空了。

4.任务栈的id是一个integer的数据类型 自增长的。

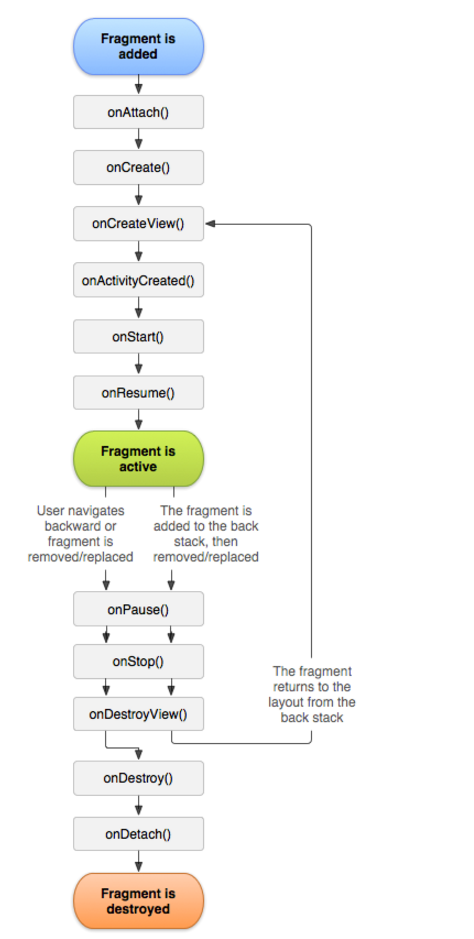
5.在android操作系统里面会存在多个任务栈，一个应用程序一个任务栈。

6.桌面应用和一般的应用程序是一样的，任务栈的行为也是一样。

7.默认情况下， 关闭掉一个应用程序，清空了这个应用程序的任务栈。应用程序的进程还会保留。

* 1. Intent能传递的数据类型
  2. 基本类型的数据和基本类型的数据数组
  3. String或CharSequence类型的数据或者是String、CharSequence类型的数组数据
  4. Serializable（序列化）:将对象序列化成二进制数据传递；
  5. Parcelable（包裹化）的数据；
  6. 对于非基本类型的数据可以通过实现Parcelable或者是Sericalizable接口
  7. CharSequence与String的区别；其中CharSequence是可读可写序列，String是可读序列。
  8. 显式Intent与隐式Intent的区别

1. 显式Intent：明确指明目标组件名称的Intent；一般情况下，不知道其他应用程序组件的名称，所以显式Intent一般用于在应用程序内部传递消息。
2. 隐式Intent：没有明确指明目标组件名称的Intent；隐式Intent用于在不同应用程序之间传递数据。隐式Intent没有指明目标组件的名称，需要Android系统寻找与Intent目标意图最匹配的组件。
   1. fragment的生命周期，和Activity如何进行通信
3. onAttach -> onCreate -> onCreateView -> onActivityCreated -> onStart -> onResume -> onPause -> onStop -> onDestoryView -> onDestory -> onDetach



* onAttach()：当该Fragment被添加到Activity时被回调。该方法只会被调用一次。
* onCreate(Bundle saveInstanceState)：创建Fragment时被回调。该方法只会被调用一次。
* onCreateView()：每次创建、绘制该Fragment的View组件时回调该方法，Fragment将会显示该方法返回的View组件。
* onActivityCreated()：当Fragment所在的Activity被启动完成后回调该方法。
* onStart()：启动Fragment时被回调。
* onResume()：恢复Fragment时被回调，在onStart()方法后一定会回调onResume()方法。
* onPause()：暂停Fragment时被回调。
* onStop()：停止Fragment时被回调。
* onDestroyView()：销毁Fragment时被回调。该方法只会被调用一次。
* onDestroy()：销毁Fragment时被回调。该方法只会被调用一次。
* onDetach()：将该Fragment从Activity中删除、替换完成时回调该方法、在onDestroy()方法后一定会回调onDetach()方法】该方法只会被调用一次。

1. 在Activity中调用fragment的方法，可以通过FragmentManager提供的findFragmentByID方法在Activity中获取到Fragment实例，进而可以调用Fragment的方法。在Fragment中调用Activity中的方法，可以通过getActivity（）方法来获取与当前fragment相关联的Activity实例，进而可以调用Activity 中的方法。
   1. 动态添加Fragment
2. 创建一个待添加Fragment的实例
3. 获取FragmentManager，在Activtiy中直接调用getFragmentManager获取；
4. 开启一个事务，通过调用FragmentManager,beginTrancation来开启；
5. 向容器内添加fragment，一般使用replace方法实现，传入容器的id和待添加的fragment实例；
6. 提交事务，调用commit方法。
7. 在FragmentTransaction中存在一个addToBackStack方法，可以将一个事务添加到返回栈中，这样在按下back键之后不会直接退出，而是返回到上一个fragment；
   1. Fragment与Activity之间的通信

由于在Android的实现机制中，fragment和Activtiy会被分别实例化为两个不相干的对象，他们之间的联系由Activtiy的一个成员对象fragmentManager维护，fragment实例化后，回到fragmentManager去注册一下，这个动作封装在fragment对象的onAttach中，所以可以在fragment中定义一些接口，当fragment调用onAttach时，将这些回调的接口实例化，这样fragment就可以调用Activtiy的成员函数了。

1. 在fragment中调用getActivity方法获取Activity实例，进而调用Activity中的方法；
2. 在Activity中调用findFragmentByID的方法，或去fragment实例，进而调用fragment方法；
3. 通过接口回调的方式：

在fragment中定义一个接口，并在activity中实现，当fragment调用onAttach方法时，将activity传递给接口引用，尽可以实现fragment和Activity之间的通信。

1. 通过广播的方式

在宿主activity中注册广播接收器，在fragment中发送广播，当需要通信的时候，可以在fragment中发送广播，在Activtiy中接收广播。

1. Handler方式；其实也是在onAttach’方法中进行完成。
   1. Android的事件分发机制，ACTION\_DOWN和后续事件如何消耗
2. 所谓的事件分发机制，其实就是对MotionEvent事件的分发过程，当一个MotionEvent产生之后，系统需要把这个事件传递给一个具体的View，这个传递的过程就是事件分发。事件分发过程主要由dispathTouchEvent。onInterceptTouchEvent、onTouchEvent事件完成。
3. 事件分发的流程是：对于一个根ViewGroup，点击事件产生后，首先会传递给他，这时它的dispatchTouchEvent就会被调用，如果这个ViewGroup的onInterceptTouchEvent方法返回TRUE就表示他要拦截当前事件，接着事件就交给这个ViewGroup处理，即他的onTouchEvent事件会被调用，如果这个ViewGroup的onInterceptTouchEvent方法返回FALSE，就表示他不拦截当前事件，这时当前事件就会继续传递给他的子元素，接着子元素的diapathchTouchEvent方法会被调用，如此反复直到事件被最终处理。
4. 对于一个普通的View，不包含ViewGroup，对事件的处理过程是：首先判断是否设置OnTouchListener，如果设置并且OnTouch方法返回TRUE，则OnTouchEvent事件不会执行，如果没有设置OnTouchListener，则执行OnTouchEvent方法。
   1. 基于事件监听、基于事件回调 假如一个touch事件发生，那是监听器先收到还是onTouch函数先收到
5. 事件的处理分为基于回调的事件处理和基于监听的事件处理，基于监听的事件处理是一种委派式的事件处理。UI组件（事件源）将发生的事件委派给特定的对象（监听器）处理。在事件处理中基于监听的事件监听器会被优先执行。基于监听的事件，事件源和事件监听器是分开的，降低了耦合性，提高了扩展性。
6. Touch事件的两种
   1. 复写View.Class中定义的onTouch 🡪 基于事件回调监听方式

@Override

public boolean onTouchEvent(MotionEvent ev) {

// TODO

return super.onTouchEvent(ev);

}

* 1. 通过监听的形式，监听View.Class中的setOnTouchListener（listener） 🡪基于监听器事件监听方式
  2. OnTouchListener ,OnTouchEvent和OnClickListener；

其中OnTouchListener的优先级最高，其次是OnTouchEvent，最后是OnClickListener；

OnClick的触发是在OnTouchEvent回调中实现的，

* 1. 创建自己的内容提供器ContentProvider

ContentProvider用于在不同应用程序之间提供数据共享的功能。

新建一个类继承ContentProvider，并重写6个抽象方法，分别是oncreate、query、insert、update、delete、getType。

query方法用于从contentprovider中查询数据，使用URI参数来确定查询哪张表；

insert方法用于向contentprovider中添加数据，使用URI参数来确定要添加的表；

update方法用于更新contentprovider中已有的内容，使用URI参数来确定更新哪一张表中的数据；

delete方法用于删除contentprovider中的数据，使用URI参数来确定删除哪一张表中的数据。

getType方法根据传入的URI返回相应的MIME类型。

借助UriMatcher来轻松实现匹配内容URI的功能。用addURI方法添加URI，（一般在静态代码块中进行添加），然后用match方法进行匹配URI。（一般在CRUD操作方法中进行match匹配）。

* 1. CotentProvider的权限问题，就是怎么样使得数据对外有选择性的提供和使用

ContentProvider如何保证隐私数据不会被泄漏出去？因为所有的CRUD操作都一定要匹配到相应的内容URI格式才能进行，我们不会向UriMatcher中添加隐私数据的URI。

* 1. 广播的两种注册方式

广播的注册方式有两种：在代码中注册和在AndroidManifest.xml文件中注册，前者是动态注册，后者是静态注册。

动态注册：创建一个广播接收器broadcastReceiver，创建一个IntentFilter，并添加相应的action，调用registerReceiver（）方法进行注册，动态注册的广播一定要取消注册；缺点是程序必须启动后才能接收到广播。

静态注册：向注册Activity一样，有一个<receiver>标签，在其中添加具体的action。优点是程序未启动就能接收到广播。

* 1. 数据持久化之文件存储

1. 将数据存入文件中
   1. 将数据存储到文件中主要调用的是Context的openFileOutput（）方法，该方法返回一个FileOutputStream对象，然后用Java流的方式向文件中写入数据；
   2. openFileOutput方法有两个参数，一个是文件名，另一个是操作模式；操作模式有两种一种是MODE\_PRIVATE（会覆盖文件中原来的文件），一种是MODE\_APPEND（不会覆盖原来的，会向原来文件后面追加）；
   3. openFileOutput不用指定文件的存储路径，因为文件默认存储在

/data/data/<package name>/files/路径下

* 1. 通过outputStreamWriter和BufferedWriter

1. 从文件中读取数据
   1. 将数据从文件中读取，主要调用Context的OpenFileInput方法，该方法只有一个参数，就是文件名，并返回一个FileInputStream对象，以Java流的方式从文件中读取数据。
   2. 数据持久化之SharedPreferences
2. 将数据存储到SharedPreferences中
   1. 首先要获取一个SharedPreferences一个对象，有三种获取方式
      1. Context类中的getSharedPerences方法，该方法有两个参数，一个是SharedPreferences的文件名，一个是操作模式。其中操作模式有两个选择MODE\_PRIVATE和MODE\_MULTI\_PROCESS（后者用于多进程中对同一个SharedPreferences进行操作）；SharedPreferences文件都默认保存在

/data/data/<package name>/shares\_prefs/目录下

* + 1. Activity中的getPreferences（）方法

只接收一个参数，即就是操作模式。使用这个方法时，会默认将当前活动的类名作为SharedPreferences的文件名。

* + 1. PreferenceManager的getDefaultSharedPreferences方法

这是一个静态方法，只接收一个Context参数，并自动使用当前应用程序的包名作为前缀来命名SharedPreferences文件。

* 1. 存储数据的步骤
     1. 调用SharedPreferences的edit方法，获取一个SharedPreferences.Editor对象
     2. 向SharedPreferences.Editor对象中添加数据，用put方法；
     3. 调用commit方法将提交的数据提交。

1. 从SharedPreferences中读取数据

首先会获取要读的SharedPreferences对象，然后调用其get方法，get方法中有两个参数，一是键值，另一个是默认值，也就是当传入相应的键找不到值时默认的返回值。

* 1. 数据持久化之数据库存储

1. 创建数据库

创建数据库需要定义一个类继承SQLiteOpenHelper，并实现onCreate和onUpdate方法；并重写一个构造方法；数据库文件一般存储在

/data/data/<package name>/databases/目录下

其中SQLiteOpenHelper的getWritableDatabase（）方法和getReadableDatabase（）方法都会返回一个SQLiteDatabase对象；

1. 升级数据库

在onCreate中添加需要增加的表，在onUpgrade（）方法中，执行Drop语句，此语句的作用是假如数据库中已经存在这个表了，进将这两个表删除，然后调用oncreate方法，为了让onUpgrade方法执行，在SQLiteOpenHelper的第四个参数中，传入比之前数据库版本号大的值就可以。

1. 添加数据

添加数据可以调用SQLiteDatabase的insert方法，有3个参数，一是要添加数据的表名；二一般传入null。三是一个ContentValues对象；

1. 更新数据

更新数据调用update方法，传入3个参数，一是表名；二是ContentValues对象，包装的是要更新的数据；三四用于约束更新某一行或者某几行的数据，不指定就默认更新所有行。

1. 删除数据

删除数据调用delete方法，传入3个参数，一是表名，第二三个参数是用于约束删除哪一行或者哪几行，不指定的话就是删除所有行。

1. 查询数据

查询数据调用query方法；query方法返回的是一个Cursor对象，Cursor对象表示的是数据库中每一行的集合，位于android.database.Cursor包下。

1. 事务：表示的是一系列的操作，这些操作要么同时发生，要么同时不发生。比如银行要将一笔钱从账户A转到账户B，那么从账户A中减去这笔钱和给账户B增加这笔钱要同时发生。

Android中事务的使用方法：

调用SQLiteDatabase的beginTransaction方法来开启一个事务，并在一个异常捕获的代码块中执行具体的数据库操作，当所有的操作完成之后，调用setTransactionSuccessful表示事务已经执行成功，最后在finally代码块中调用endTransaction来结束任务。

* 1. Handler Looper的使用方式，实现机制，设计的原因

Handler的设计原因：在Android中不允许在子线程中进行UI更新，所以设计了异步消息处理机制。

首先在需要更新的地方创建一个子线程，但不直接进行UI更新操作，将需要更新的内容以Message的方式，通过Handler的sendMessage方法传递到主线程中，进而在主线程的handleMessage方法中进行回调，对UI进行更新操作。

Handler的实现主要由Handler、MessageQueue、Looper、Message四部分完成。首先通过Handler将Messsge发送到MessageQueue中，Looper的loop方法不断的查询MessageQueue中是否有新的Message，有的话将Message传递到Handler的handleMessage中进行处理。

* 1. ThreadLocal是干嘛的，使用的地方

ThreadLocal是一个线程内部的数据存储类，通过他可以在指定的线程中存储数据，数据存储以后，只有在指定的线程中可以获取到存储到的数据，对于其他线程来说，则无法获取到数据。

一般来说，当某些数据是以线程为作用域并且不同的线程具有不同的数据副本的时候，可以使用ThreadLocal。

* 1. AsyncTask

AsyncTask 是一个异步任务类，也是基于异步消息处理机制的；他可以在线程池中执行后台任务，然后把执行的进度和最终结果传递给主线程，并在主线程中更新UI。从实现上讲，AsyncTask封装了Thread和Handler，通过AsyncTask可以更加方便的执行后台任务以及在主线程中访问ＵＩ。

提供３个泛型参数params、progress、result。并有4个方法供重写

OnPreExecute:在主线程中执行，初始化一些UI操作

doInBackground：在线程池中执行，执行后台任务

onProcessUpdate：在主线程中执行，publishProgress方法的回调

onPostExecute：doInBackground方法返回后会调用

* 1. HandlerThread有没有了解

HandlerThread继承了Thread，是一种可以使用Handler的Thread，实现是通过在run方法中通过Looper.prepare（）来创建消息队列，并通过Looper.loop（）来开启消息循环，这样就允许在HandlerThread中使用Handler了。

与普通Thread的不同之处在于，普通Thread主要在run方法中执行一个耗时任务，而HandlerThread则是在内部创建了一个消息队列，外界需要通过Handler的消息方式来通知HandlerThread执行一个具体任务。

HandlerThread比较适用于单线程+异步队列的场景，比如IO读写操作，耗时不多而且也不会产生较大的阻塞。对于网络IO操作，

* 1. 线程中Thread、Looper、MessageQueue、以及Handler之间的关系

一个线程只能有一个Loop对象，而Loop对象在创建时会自动创建一个MessageQueue，并且MessageQueue不能手动创建，也就是说一个线程只有一个Looper 和一个MessageQueue。而一个线程可以创建多个Handler，所以

Thread ： Looper ： MessageQueue : Handler = 1 : 1 : 1 : N .

* 1. 线程池的概念、好处、常见的线程池举例

1. ThreadPoolExecutor中构造函数中参数的意义：
   1. CorePoolSize：核心线程数，默认情况下，核心线程会在线程池中一直存活，即使他们处于空闲状态；
   2. maximumPoolSize：线程池所能容纳的最大线程数，当活动线程数达到这个值后，后续任务将会被阻塞；
   3. KeepAliveTime：非核心线程闲置时的超时时长，超过这个值，非核心线程会被回收。
   4. WorkQueue: 线程池中的任务队列。
2. 线程池的好处
   1. 重用线程池中的线程，避免因为线程的创建和销毁所带来的性能开销；
   2. 能有效控制线程池的最大并发数，避免大量的线程之间因互相抢占系统资源而导致的阻塞现象；
   3. 能够对线程进行简单的管理，并提供定时执行以及指定间隔循环执行等功能。
3. 常见的线程池举例
   1. FixedThreadPool：一种线程数量固定的线程池，当线程处于空闲状态时，他们并不会被回收，除非线程池被关闭了。当所有的线程都处于活动状态时，新任务都处于等待状态，直到有线程空闲出来。只有核心线程并且核心线程没有超时限制，任务队列也没有大小限制。
   2. CachedThreadPool：一种线程数量不定的线程池，他只有非核心线程，并且其最大的线程数为Integer.MAX\_VALUE。当线程池中的线程都处于活动状态时，线程池会创建新的线程来处理新的任务，否则就会利用空闲线程来处理新的任务。
   3. ScheduledThreadPool：他的核心线程数量是固定的，而非核心线程数是没有限制的，并且当非核心线程闲置时会被立即回收。，主要用于执行定时任务和具有固定周期的重复任务。
   4. SingleThreadExecutor：这类线程池内部只有一个核心线程，确保所有的任务都在同一线程中按顺序执行。
4. 线程池的使用场景：
   1. 比如在列表中加载图片的时候，就应该使用线程池，因为列表在滑动的过程中，会产生多个线程，造成资源的浪费，此时应该使用线程池。
5. ThreadPoolExecutor执行任务遵循的原则：
   1. 如果线程池中的线程数量未达到核心线程的数量，那么直接启动一个核心线程；
   2. 如果线程池中的线程数量已经达到或者已经超过核心线程的数量，那么任务会被插入到任务队列中排队等待执行。
   3. 如果在2中无法将任务插入到任务队列中，原因是任务队列已满，这个时候如果线程数量未达到线程池规定的最大值，那么会启动一个非核心线程来执行任务。
   4. 如果3中线程数量已经达到线程池规定的最大值，那么就拒绝执行该任务。
   5. View的绘制流程，MeasureSpec如何工作，父的MeasureSpec和自己的LayoutParams
6. View的绘制流程：

View的绘制流程从ViewRoot的PerformTraversals方法开始，经过measure、layout、draw三个过程最终将View绘制出来，其中measure方法用来测量View的宽和高，layout方法用来确定View在父容器中的放置位置，draw方法负责将View放置在屏幕上。

1. MeasureSpec如何工作

MeasureSpec代表一个32位的int值，高2位代表SpecMode（测量模式），低30位表示SpecSize（某种测量规格下的规格大小）

1. 在View测量时，系统会将LayoutParams在父容器的约束下转换成对应的MeasureSpec，然后再根据这个MeasureSpec来确定View测量后的宽和高。
   1. 自定义View需要注意的问题
2. **让View支持wrap\_content**   
   直接继承View或ViewGroup的控件 都不支持wrap\_content 方法 需要在onMeasure方法中设置
3. **如果有必要 支持padding**   
   直接继承View的控件，如果不在onDraw方法中处理padding 那么padding属性是无法起作用的 直接继承ViewGroup的控件，需要处理padding和子元素的margin对其的影响，否则这两个属性也无效
4. **尽量不要在view中使用handler 因为view提供了post方法**   
   View内部本身就提供了post系列方法 可以代替handler 但如果明确需要Handler 可以使用
5. **view中有线程或者动画 要及时停止**   
   这是为了防止内存泄漏，可以在onDetachedFromWindow方法中结束，这个方法回调的时机是 当View的Activity退出或者当前View被移除的时候 会调用 这时候是结束动画或者线程的好时机 另外还有一个对应的方法 onAttachedToWindow 这个方法调用的时机是在包含View的Activity启动时 回调 回调在onDraw方法 之前
6. **有嵌套滑动效果时 注意处理滑动冲突**   
   如果有滑动冲突的时候，需要合适的处理滑动冲突
   1. View的绘制流程

View在绘制的时候，ActivityThread会调用ViewRoot中的ViewRootImpl方法，从performTravels开始，首先调用performMeasure，然后调用performlayout,然后再调用performDraw，这三个方法完成顶级View的绘制，在performMeasure方法中调用measure方法，然后再调用onMeasure方法，在OnMeasure方法中对子View进行绘制此时measure过程就从父容器传递到子view中。其余两个方法也是类似。

Measure过程决定了view的测量后宽和高；

Layout过程决定了view四个顶点的位置以及实际的宽和高；

Draw过程决定了view的显示位置。

* 1. View和ViewGroup

在Android中控件基本上分成View控件和ViewGroup控件，ViewGroup作为父控件可以包含多个View和ViewGroup，通过最顶层的ViewGroup，在整个界面上的控件上形成了一个树形结构，这就是控件树，在findViewById函数中就是通过深度优先遍历View树的。

* 1. MeasureSpec和LayoutParams的关系

对于普通的View，MeasureSpec 是由其父容器的MeasureSpec以及自身的LayoutParams参数共同确定的。

* 1. MeasureSpec

是一个帮助View测量过程的类，是一个32位的int值，高2位表示测量模式，低30位表示测量大小。

其中测量模式有3种：

AT\_MOST：最大的模式；对应于指定的是wrap\_content。

EXACTLY：精确模式；对应于指定的是具体的数值或者是match\_parent时；

UNSPECIFIED：不指定测量大小的模式；

大小的表示方式：

Wrap\_content：最大模式，大小不定，但是超过窗口大小；

Match\_parent：精确模式，大小就是窗口大小；

固定大小：精确模式，大小就是传进去的大小。

* 1. 自定义View的三个构造函数
     1. 带有一个参数context的构造函数
     2. 带有两个参数（context，attributeset）的构造函数
     3. 带有三个参数（context，attributeset，defStyle）的构造函数

如果在Code中实例化一个View会调用第一个构造函数，如果在xml中定义会调用第二个构造函数，而第三个函数系统是不调用的，要由View（我们自定义的或系统预定义的View，如此处的CustomTextView和Button）显式调用，比如在这里我们在第二个构造函数中调用了第三个构造函数，并将R.attr.CustomizeStyle传给了第三个参数。

　　第三个参数的意义就如同它的名字所说的，是默认的Style，只是这里没有说清楚，这里的默认的Style是指它在当前Application或Activity所用的Theme中的默认Style，

* 1. 滑动冲突的解决方式

1. 外部拦截法：点击事件都先经过父容器的拦截处理，如果父容器需要此事件就拦截，如果不需要此事件就不拦截。
2. 内部拦截法：父容器不拦截任何事件，所有事件都传递给子元素，如果子元素需要此事件就直接消耗，否则就交给父容器处理。
   1. HttpURLConnection 与HttpClient的区别
3. HttpURLConnection与HttpClient都支持HTTPS协议，以流的形式进行上传和下载，配置超时时间，IPV6以及连接池等功能；
4. HttpClient由于API数量过多，很难再不破坏兼容性的情况下对它进行升级和扩展，所以在Android2.3之后，主流的都是用HttpURLConnection。
5. HttpURLConnection是一种多用途的、轻量级的HTTP客户端，虽然提供的API比较少，但同时使我们更容易的实现和使用他。
6. HttpURLConnection的压缩和缓存机制都可以有效的减少网络访问流量，在提升速度和省电方面也起到了很大作用。
   1. 如何解析json数据和xml
7. 使用JSonObject解析json数据

定义一个JSONArray对象，将服务器传回来的数据保存在其中；

循环遍历这个JSONArray，取出来的每一个元素都是JSONObject对象；

每个JSONObject对象中都包含有各种信息，用对应的get方法一一取出即可。

1. 使用GSON解析GSON数据
   1. 假如要解析的json数据是{“name”:”tom”,”age”:20};

在解析时，可以顶一个person字段，并加入name和age两个字段，然后只需要简单的调用 Gson gson = new Gson();

Person person = gson.fromJson(jsonData,Person.class);即可.。其中jsonData是一个从服务器返回的字符串。

* 1. 假如要解析的数据是一个JSON数组，需要借助TypeToken将期望解析的数据类型传入到fromJSON（）方法中

如List<Person> people = gson.fromJson(jsonData,new TypeToken<List<Person>>(){}.getType());

* 1. listView的优化方法

1. 使用ViewHolder模式提高效率，避免了每次在调用getView（）的时候都去通过findViewByID（）实例化控件；
2. 利用ConvertView参数，在调用getView方法的时候进行判断一下，看convertView是否为空，为空在重新加载，不为空则直接重用。ConvertView用于存储被滑出屏幕的Item。
3. 使用分段加载：有些情况下，我们需要加载网络中的数据，显示到ListView，而往往此时都是数据量比较多的一种情况，如果数据有1000条，没有优化过的ListView都是一次性将数据全部加载出来的，很显然需要一段时间才能加载，这时候我们可以使用分段加载，比如先设置每次加载数据10条，当用户滑动ListVIew到底部的时候，我们在加载20条数据，然后使用Adapter刷新ListView。
4. 使用分页加载：分段加载不能完全解决OOM崩溃的情况，因为我们虽然在分段中一次只增加10条数据到List集合中，然后我们再刷新到ListView中去，假如有10万条数据，如果我们顺利读到最后，这个List集合中还是会累积海量条数据，还是会造成OOM崩溃的情况，这时候我们就需要用到分页，比如我们将这10万条数据分成1000页，每页100条数据，每一页加载的时候都覆盖掉上一页中List集合中的内容，然后每一页再使用分批加载，这样用户体验就会好一些。
   * 1. 首先要先确定默认展示在ListView上的数据，比如默认在ListView上展示10条数据。
     2. 将数据传递到自定义的适配器上，然后加载到ListView中。
     3. 当用户将数据拉到最后一条的时候，就要开始刷新加载新数据了。
     4. 通过监听ListView的滑动事件，判断是否达到最后一条，如果达到最后一条则开始刷新。
   1. ListView 与RecycleView的区别
5. 布局效果上，ListView只支持线性布局，而RecycleView支持线性布局、网格布局、瀑布流布局；滚动方式上，ListView只支持纵向滚动，而RecycleView不仅支持纵向滚动还支持横向滚动。
6. 在ListView中，ViewHolder需要自己定义，是一种推荐的方式，在RecycleView中使用RecycleView.ViewHolder则变成了必须，但是可以解决在不使用ViewHolder造成的每次在getView方法中都需要进行findViewById（），造成性能延缓的问题。
7. 在ListView中删除Item是无法产生动画效果的，RecycleView.ItemAnimator用于在RecycleView添加、删除或移动item时处理动画效果。
8. ListView中设置item之间的间隔符时只需要在item的布局文件中添加就好了，但是对于RecycleView，假如要添加间隔符，必须使用RecycleView.ItemDecoration。
9. ListView通过AdapterView.OnItemClickListener接口来探测点击事件，但是RecycleView是通过RecycleView.OnItemClickListener接口来探测触摸事件。
   1. RecycleView，相对于ListView的优化

RecycleView相比于ListView，添加了缓存机制，当一个Item没有被创建时，调用onCreateViewHolder方法创建这个Item，同时将其放到缓存中，以ViewHolder的方式返回，当要显示一个Item的时候，就会在缓存中寻找有没有创建过这个Item，如果创建了就直接调用onBindViewHolder方法进行渲染显示。

* 1. ListView中getView（）方法的工作原理

在一个完整的ListView第一次显示的时候，每一个Item都是null，getView的时候会跑到需要inflate一个Item的代码段，假设一个ListView界面最多能显示10个Item，那么当滑动好第11个的时候，第一个Item会放到“recycler”中，然后再滑动的时候，ListVIew会看一下要显示的Item和recycler中的Item是否是一样的，如果是一样的，就直接调用，不需要再重复inflate一次。

* 1. 假如你要记录ListView滚动到的位置，要记录什么信息，view怎样获得坐标信息

记录ListView滚动到的位置的坐标，然后利用ListView.scrollTo精确的进行恢复；



* 1. Binder工作机制

1. Binder是一种client-server的通信结构；其实在提供远程过程调用的功能；
2. 从表面上看，client是通过获取一个server的代理接口，对server进行直接调用；
3. 实际上，代理接口中定义的方法与server中定义的方法时一一对应的；
4. 代理接口将该Parcel发送给内核中的binder driver；
5. Server会读取binder driver中的请求数据，如果是发送给自己的，解包Parcel对象，处理并将结果返回。
6. 整个调用的过程是一个同步的过程，在server处理的时候，client会block住。
   1. 进程间通信有哪几种方式？
7. 使用Bundle，由于Bundle实现了Parcelable接口，所以他可以很方便的在不同进程间传输；
8. 使用文件共享：两个进程通过读/写同一个文件来交换数据；
9. 使用Messenger：可以通过Messenger在不同进程中传递Message对象；在Message对象中放入我们需要传递的数据，就可以实现数据的进程间传递。Messenger的底层实现是AIDL。
   1. 你的APP里，是每个Activity都有一个Handler呢还是所有Activity共享一个Handler

两种方式都可以，只是各有优缺点。

1. 整个主UI，只创建一个Handler，即全局的Handler，然后多个Activtiy共享这个Handler发送消息。

优点：只用一个消息循环，比较能提高性能；

缺点：发送消息时，传递数据不方便，需要将Activtiy的各变量值打包传递给这个Handler。

1. 每一个Activtiy创建一个Handler，当前activity就用他自己的Handler变量来发送消息；

优点：发送消息时，传递数据很方便，因为利用的是当前acitivity类里的变量值；

缺点：创建了多个消息队列，容易忘记释放，影响性能。

* 1. Service是什么？

服务是安卓中实现程序后台运行的解决方案，适合于不需要和用户进行交互，但是又需要长期运行在后台的任务。服务不是运行在一个独立的进程中，二是在依赖于创建服务时应用程序所在的进程，一旦这个进程被销毁，服务也会被销毁。服务也不会开启一个线程，服务默认运行在主线程中。

* 1. Service的生命周期，两种启动方法，有什么区别。

1. Service的两种启动方式：一是启动类型的服务，二是绑定类型的服务；
2. 启动类型服务的生命周期：onCreate() -> onStartCommand() -> Service Running -> 调用context.stopService –> onDestory()

Onstart（）方法可以被调用多次，但是只需要调用一次stopService方法就可以将服务终止。

1. 绑定类型服务的生命周期：onCreate() -> onBind() （只能绑定一次）->Service Running -> 调用unBind() -> onDestory()
2. bindService方法获得是一个服务的持久连接。
3. **区别：**使用startService方法启动服务，调用者与服务之间没有联系，即使调用者退出了，服务仍在后台运行；bindService方法启动服务，调用者与服务绑定在一起，一旦调用者退出，服务也就终止。
   1. 如何保证service不被杀死
4. 在onStartCommand方法中，返回START\_STICKY；

START\_STICKY在运行onStartCommand后service进程被kill后，将保留在初始状态，但是不保留传入的intent，不久后service将尝试重建，因为保留在开始状态，在创建service时，能保证调用onStartCommand。

1. 提升service进程的优先级；（不太懂）
2. 在OnDestory方法中重启service；
3. 提升service进程的优先级
   1. IntentService

IntentService是一种特殊的Service，是一个抽象类，IntentService可以用来执行后台耗时任务，当任务执行后他会自动停止，由于IntentService是服务的原因，他比普通的线程优先级要高，所以IntentService适合执行一些高优先级的后台任务。在实现上IntentService封装了HandlerThread和Handler。

IntentService第一次启动的时候，会调用onCreate方法，在onCreate方法中会创建一个HandlerThread对象，并通过他的Looper方法创建一个ServiceHandler对象，这样子serviceHandler发送的消息就能在HandlerThread中执行，从这个角度看，IntentService也可以用于执行耗时任务。（我的理解是，HandlerThread是继承自Thread的，所以可以执行后台任务）

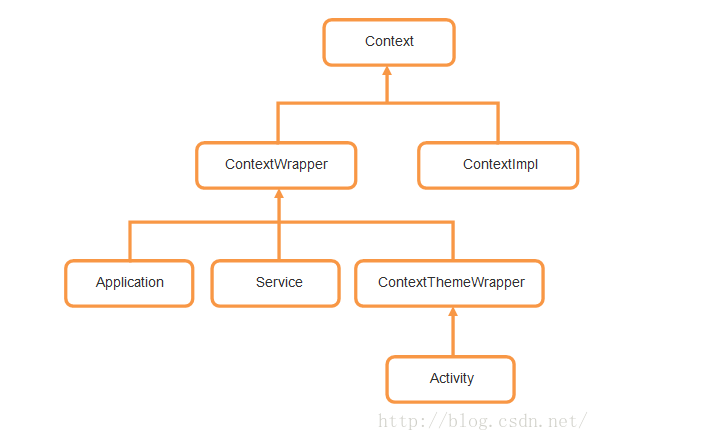
* 1. Parcelable与Serializable之间的区别

1. Serializable序列化接口，实现的时候特别方便，只需要继承Serializable接口即可，但是缺点是使用了反射，序列化的过程比较慢，这种机制在序列化的时候创建了许多临时对象，容易触发垃圾回收。
2. Parcelable 速度比较快，实现起来比较麻烦；

需要重写writeToParcel方法，以及调用Parcelable.Creator方法。

* 1. Context类型

Context是一个上下文环境，Context是维持Android程序中各组件能够正常工作的一个核心功能类。继承结构如下：



直系子类有两个，一个是ContextWrapper，一个是ContextImpl。那么从名字上就可以看出，ContextWrapper是上下文功能的封装类，而ContextImpl则是上下文功能的实现类。而ContextWrapper又有三个直接的子类，ContextThemeWrapper、Service和Application。其中，ContextThemeWrapper是一个带主题的封装类，而它有一个直接子类就是Activity。

Context一共有三种类型，分别是Application、Activity和Service。这三个类虽然分别各种承担着不同的作用，但它们都属于Context的一种，而它们具体Context的功能则是由ContextImpl类去实现的。

一个应用程序中context的数量等于activity的数量+service的数量 + 1；

* 1. Sqlite的一些问题
  2. JNI的参数ENV参数
  3. JNI工作时，java 层的类如何找到C/C++层的类
  4. AIDL的工作流程
  5. AsyncTask 为什么java已经有线程池的实现了，还要继续使用AsyncTask，AsyncTask相对于java自带的线程池的好处
  6. Android中动画的两大类

1. 视图动画Animation ：动画只能显示，不能响应事件。

有透明动画、旋转动画、缩放动画、位移动画四种方式，并提供了AnimationSet动画集合，能够混合使用多种动画。

相比于属性动画，视图动画非常大的缺陷就是不具备交互性，当某个元素发生视图动画后，其响应事件的位置还依然在动画前的地方，所以视图动画只能做普通的动画效果，尽量避免交互的产生。

1. 属性动画Animator

Animator框架中使用最多的就是AnimatorSet和ObjectAnimator配合，其中ObjectAnimator控制一个对象的一个属性，使用多个ObjectAnimator组合到AnimatorSet形成一个动画。

属性动画通过调用属性的get和set方法来真是的控制一个View的属性值，所以在响应点击事件的时候，是在新的位置响应。

* 1. Android 中动画的分类：

Android中动画的分类：一是补间动画（Tween Animation）,二是逐帧动画（Frame Animation）、三是属性动画（Property Animation）。但是补间动画和逐帧动画都是属于View Animation，只能作用于View。补间动画不改变View对象的绘制位置，不改变View对象属性。

补间动画，指通过指定View的初末状态和变化时间、方式，对View的内容完成一系列的图形变化来实现动画效果，主要包括Alpha、Scale、Translate、和Romate。

帧动画：即指定每一帧的内容和停留时间，然后播放动画。

属性动画：通过调用属性的get、set方法来真实地控制了一个View的属性，进而可以响应view的点击事件。

* 1. Style和theme的区别

Style是一组属性的集合，这些属性用于指定View和window的外观和样式。

Theme是style，用于Activity和application，而非单个View；当使用theme时，整个Activity 或Application内的View都会应用该theme的style。

共同点：

Style和theme的内容的形式是一样的，都包含如下的内容：

写在xml文件中，根节点是<resource>，<style>标签以及其唯一的name属性，<item>标签。

区别：

Style和theme的不同，在于应用对象。

Style应用于View或window；theme用于Activtiy和Application。

Style是针对窗体元素级别的，用于改变指定控件或者layout；

Theme是针对窗体级别的，改变窗体样式，如窗口标题、边框等。

* 1. 说一下MVC以及其在Android中的应用
  2. ScrollView中嵌套ListView

出现的问题：ScrollView中嵌套的ListView，ListView只能显示一行，不能全部显示。ScrollView中嵌套的ListView无法计算ListView的控件高度。scroll事件的消费处理以及ListView控件的高度设定问题。

解决方法：自定义一个可以在ScrollView中用到的ListView；

首先，继承自ListView；

其次，三个构造方法完全不用动，只需要重写OnMeasure方法。

然后，在OnMeasure方法中，重新计算ListView的Height，

具体值是：int expandSpec = MeasureSpec.makeMeasureSpec(Integer.MAX\_VALUE >> 2, MeasureSpec.AT\_MOST);

* 1. viewpager里面只能嵌套view吗 可不可以嵌套Activity
  2. 详细展开说一下所有LaunchMode的应用场景

       假如A-》B-》C，想让C后退直接到A，使用什么样的intentflag？

ActivtiyC应该使用的intentFlag是Intent.FLAG\_ACTIVITY\_CLEAR\_TOP。

       假如A-》B-》C，C使用singleTask，C后退，后退到什么地方呢？

应该回退到A。

* 1. AsyncTask内部实现机理 与Thread+Handler有什么不同
  2. Scrollview scrollview和ListView有什么相似点 有什么不同 那如果这两个是继承关系 那应该是谁继承谁？

ListView和ScrollView都可以滚动，不同点是ListView是一个树状的列表显示，而ScrollView是一个内容展示。如果有继承关系，应该是LIstView继承ScrollView，应为ListView扩展了ScrollView的功能，ListView应该是子类。

* 1. 说一下OOM以及你怎么解决的

1. 在用ViewPager加ImageView实现轮播功能的时候，设置一个Handler的PostDelay方法，在PostDelay方法中在调用一个Handler的postDelay，此时当fragment退出的时，就出现了OOM。解决办法就是在Fragment的OnDestory中调用Handler的remove。。。方法，将Handler对象释放。
2. 调用getContext方法时，出现了OOM。解决方法时调用getApplication的getContext方法。
3. 本地编译环境设置的不合理，也会出现OOM问题，解决办法是添加gradle.properties文件，然后再里面添加代码org.gradle.jvmargs=-Xmx1024m，使得分配给java虚拟机的堆内存增大
4. OOM发生在ListView加载的时候，由于加载的东西过多，而出现了OOM，解决办法就是分页加载。
5. Bitmap加载的时候
   1. 图片列表这块做了什么优化
      1. RecycleView滚动时停止加载，停止时再进行加载；
      2. ViewPager + Fragment + RecycleView中多个RecycleView复用同一个RecycleViewPool，减少内存占用。
   2. Bitmap的压缩
      1. 采用BitmapFactory.Options来加载所需尺寸的图片，通过BitmapFactory.Options可以按一定的采样率来加载缩小后的图片
      2. 主要用到它的inSampleSize参数；
      3. 流程如下：
         * 1. 将BitmapFactory.Options的inJustDecodeBounds参数设置为true并加载图片；
           2. 从BitmapFactory.Options中取出图片的原始宽高信息，他们对应于outWidth和outHeight参数；
           3. 根据采样率的规则并结合目标View的所需大小计算出采用率inSampleSize；
           4. 将BitmapFactory.Options的inJustDecodeBounds参数设置为false，然后重新加载图片。
   3. 内存泄漏和内存溢出

内存溢出：out of memory 是指程序在申请内存的时，没有足够的内存空间供其使用，出现out of memory。比如申请了一个int类型的空间，但是却存了一个long型的数据，就是内存溢出。当应用的堆资源超过了Dalvik虚拟机分配的内存就会内存溢出。

内存泄漏：memory leak，是指程序在申请内存后，无法释放已经申请的内存空间，一次内存泄漏危害可以忽略不计，但是内存泄漏堆积后果很严重，无论多少内存迟早会被占光，内存泄漏最终会导致内存溢出。为什么无法释放已经申请的空间，这是因为当一个对象已经不再需要使用了，本该被回收时，而又另外一个正在使用的对象持有他的引用，从而导致了对象不能被GC回收。

* 1. 常见的内存泄漏

1. 单例模式造成的内存泄漏

单例模式中传入的context是Activtiy的Context，只要这个单例模式存在，Activtiy就无法销毁，会引发内存泄漏；

解决办法就是调用Application的Context，因为Application的生命周期是整个进程的生命周期。

1. Handler引起的内存泄漏

Handler是在Looper线程中不断读取消息的，当Activity销毁时，MessageQUeue中还存在消息，那么消息就有Handler的引用，而handler又有Activtiy的引用，导致Activtiy无法回收，引发内存泄漏。

解决方式是在Activtiy的destory函数中，清除MessageQueue中的message；调用的是Handler的removeCallbackAndMessage方法。

1. 资源未正确关闭引发的内存泄漏

如BroadcastReceiver、File、Cursor、ContentObserve、Stream；

解决方式在Activtiy的destory函数中，关闭这些资源。

1. 线程造成的内存泄漏

异步任务和Runnable是一个匿名内部类，因此，他们都含有一个队当前Activtiy的隐式引用，当Activtiy在销毁之前，任务还未完成，那么将导致Activtiy的资源无法进行回收，造成内存泄漏。

解决方式是将线程的实现不同匿名内部而是使用静态内部类。

还有假如用AsyNCTask实现多线程时，在Activtiy销毁时也要对AsyncTask进行回收。

1. 如何进行内存泄漏检测
   1. 使用开源库LeadCanary；
   2. ANR Application Not Responding

产生原因：

* + - 1. 主线程（UI线程）响应用户操作事件时间超过5秒；
      2. BroadcastReceiver超过10秒未执行完毕

Android应用程序完全运行在一个独立的线程中，任何主线程中运行需要消耗大量的操作都会引发ANR，因此需要消耗大量时间的操作如访问网络或者数据库，都要放到子线程中或者使用异步方式来完成。

* 1. 强制关闭

产生原因：程序出现异常，一般像空指针、数组越界、类型转换异常等；

避免方法；编写程序时要思维缜密，异常出现时可以通过logcat查看抛出异常代码出现的位置，然后到程序中进行修改。

* 1. 项目中登录体系的实现

分为注册过程和登录过程：

注册过程：填写手机号，获取验证码，输入验证码，进行匹配验证；验证成功之后，用手机号和密码的方式进行注册；此时密码采用的是MD5加密的方式，传输到服务器；

再次就是登陆的设计：

实现了手动登陆和自动登录两种方式：

手动登陆的话：首先是填写手机号以及密码；同样密码也是采用的MD5的方式，加密后的密码然后跟服务器中存储的密码进行匹配验证；登录成功后返回一个token，保存在本地；

每次登录的时候，先判断本地是否含有存储token的对象，没有就直接手动登陆，有的话就验证本地的Token跟服务器的Token是否一致，一致的话就直接登录。

* 1. 基于Token的身份验证方法

使用基于Token的身份验证方法，在服务端不需要存储用户的登录记录；大概流程是这样子的：

1. 客户端使用用户名和密码请求登录；
2. 服务端收到请求后，去验证用户名和密码；
3. 验证成功后，服务端会签发一个Token，再把这个Token发送给客户端，
4. 客户端收到Token后，将Token存储起来，如放在本地或者是放在COOkie中；
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的Token；
6. 服务端收到请求，去验证客户端请求里面携带的Token，如果验证成功，就向客户返回请求的数据。
   1. MVP与MVC的区别
7. MVC指的是Model View Controller；MVP指的是Model View Presneter；他们都是用来分离UI层与具体的业务逻辑层的；Presenter中将复杂的逻辑处理代码给提取出来。
8. 最关键的差异是MVP中View不直接与Model交互，他们之间的交互是通过接口来实现的。但是MVC模式中，View与Model是直接交互的。
   1. 为什么要使用base64编码

因为计算机中数据都是按ASCII码进行存储的，而ASCII码的128-255之间的值是不可见字符，在网络上传输的时候，由于不同的路由对于字符的处理格式不同，这样不可见的字符在传输过程中就有可能出现错误，所以可以采用base64编码的方式，将数据先进行编码，然后再进行传输。

* 1. Activity的跳转形式

1. 使用显示Intent
2. 使用隐式Intent；
3. 使用显示Intent时，有两种情况，分别是返回数据的，不返回数据的；返回数据的用startActivtiyForResult；不返回数据的用startActivity；用setResult传递返回的数据，返回数据的用onActivityResult进行回调。
   1. APK的打包过程
4. 打包资源文件，生成R.java文件
5. 打包aidl文件，生成相应的Java文件
6. 编译工程源码，生成.class 文件
7. 转换所有的class文件，生成classes.dex文件
8. 打包生成apk文件
9. 对apk文件进行签名处理
10. 对签名后的apk进行对齐处理
    1. Android中的布局优化
11. 优化布局层级，降低View树的高度，官方文档中说明View树高度不宜超过10层。

使用相对布局代替线性布局

1. 避免嵌套过多无用的布局，使用<include>标签重用公用的Layout；
   1. Android内存优化
2. Bitmap的加载优化
   1. 将加载的图片进行压缩
3. 对常量使用static final
4. 对频繁更新的String 使用StringBUilder或者StringBUffer，因为String假如内存中没有，会在内存中创建一个新的String对象
5. 尽量不适用枚举
6. 对Cursor，File等注意他的创建与回收
7. 使用视图缓存，而不是每次都调用inflate（）方法
   1. Android DVM

Dalvik和JVM之间的区别是Dalvik是基于寄存器的，而JVM是基于栈的，基于寄存器可以对提前优化提供更好的支持，基于寄存器的虚拟机对于更大的程序来说，在编译时，花费的时间更短。

Dalvik和Java虚拟机之间另外一个区别就是Dalvik经过优化，允许在有限的内存中同时运行多个虚拟机实例，并且每一个Dalvik应用作为一个独立的Linux进程执行，独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

* 1. 方法数超过65535

原因：在Android系统中，一个App的所有代码都在一个Dex文件里面。Dex是一个类似Jar的存储了许多有Java编译字节码的归档文件。当Android系统启动一个应用的时候，有一步是对Dex进行优化，这个过程有一个专门的工具来处理，叫DexOpt。DexOpt的执行过程是在第一次加载Dex文件的时候执行的。这个过程会生成一个ODEX文件，即Optimised Dex。执行ODex的效率会比直接执行Dex文件的效率要高很多。但是在早期的Android系统中，DexOpt有一个问题， DexOpt会把每一个类的方法id检索起来，存在一个链表结构里面。但是这个链表的长度是用一个short类型来保存的，导致了方法id的数目不能够超过65536个。当一个项目足够大的时候，显然这个方法数的上限是不够的。

解决方法：使用MultiDexApplication；在gradle.builder中添加依赖；

然后再activity中重写

@Override

protected void attachBaseContext(Context base) {

super.attachBaseContext(base);

MultiDex.install(this);

}

* 1. Retrofit的使用
     1. Retrofit使用时将底层的代码封装起来，只暴露出我们业务中的数据模型和操作方法；
     2. 定义一个接口类，将Rest API封装成Java接口，根据我们的需要将其进行封装，使用到了retrofit的注解功能如Get POST注解；
     3. 获取一个retrofit对象，然后通过retrofit对象获得接口的实例。



可以在获得retrofit对象时，通过client方法将OKHttp作为网络请求的执行器。

* + 1. 然后再通过接口去调用相应的方法。如：



此时可以采用两种方式进行调用，

1. 同步的方式：call.execute()；
2. 异步的方式：call.enqueue();
   1. OKhttp的好处：
      1. 能自动维护socket连接池，减少握手次数，
      2. 有队列线程池，能够轻松实现并发；
      3. 支持socket自动选择最好路径，并支持自动重连。

        3、同上情况，使用Fragment，又当如何？与viewpager有什么区别

        4、ExpandableListView的Adapter怎么写

        5、在ListView的每个item中如果可能出现的view都不一样，如何处理？动态获取view种类数量的话是不是就不能使用viewHolder进行优化？固定显示view如果不存在该种view就不显示的方法是否太耗内存

        7、Activity之间传递大量的数据用什么方法

                Binder

                Bundle不够大

                Socket 在同一应用中没必要使用

                共享内存

                static变量——同一应用中static变量如何控制，会不会并发出问题，如果Activity由于内存不足被kill掉，再重启的时候static变量的值还可靠吗：设置专门的访问类，进行读写控制，以及Activity重启后的初始化工作。

        8、SQLite在使用时 假如ActivityA是用于修改SQLite，ActivityBCDEF。。。是用来展示SQLite的内容，如何在修改SQLite的同时使得BCDEF...都立刻获得最新的数据？SQLite有没有类似于ContentObserver的这么一种监视类可以监听到SQLite内容的改变？如果广播通知所有类，那么广播可以携带多大的数据？

        9、假设把一个ActivityB向右滑动可以回到上一个ActivityA，B渐渐透明，A渐渐清晰，要怎么编写页面？说说你对android:theme的理解，说说你对alpha值的理解

        10、ListView在数据量很大图片很多的情况下怎么优化？假如一个图片，轻轻的向上滑动一丢丢，那么需要重绘吗？（什么鬼。。。）

        11、

        12、在全屏显示高清大图的时候，网速慢的情况下如何显示？渐进式显示是怎么做到的？如果想显示下载进度圈圈，该如何实现？

        13、AsyncTask在4.x以后有什么改变？怎样改回并发执行好多个？如果一个AsyncTask结束取得结果之前Activity就因为内存原因被Destroy掉了，那会有什么情况发生？会内存泄露吗？会空指针吗？需要在Activity彻底死掉之前把AsyncTaskcancel掉吗？如果没有cancel掉，然后Activity重启了，那这个Asynctask又当如何呢

        19、说一下canvas

        20、dialog和popupwindow的应用场景 如果popupwindow在没有dismiss的时候Activity finish了，会出现什么情况？

        21、了解过什么三方库？为什么不用三方库？

        22、你觉得安卓的未来在哪里？你自己为什么都用IOS了(╯‵□′)╯︵┻━┻

        23、Service和推送通知，通知有没有可能出现有推送但是通知栏收不到通知？service被kill掉会如何？如何保证service不被kill掉

        25、dp sp px各自的应用场景是什么 换算关系是什么 使用sp的时候如果调整系统字体，显示字体会跟着调整吗？

        26、gallery为什么被废弃？为什么Google推荐使用viewpager代替gallery呢？有什么改进？

        27、handler.postAtTime不是延时post么 那handler怎么延时处理Message

美团这里问到的是我A打开了B，B打开了C，C的右上角有一个叉叉，那我怎样做到使我点击C的叉叉，就直接关闭了BC回到A，PS，不可以用startActivityForResult

假如，ViewPager里面一页的宽度超过了屏幕的宽度，ViewPager是怎么显示的，这还真没试过。

然后问到了app是怎么通知Android系统页面有刷新的，还有view的measure函数之类的，进而讨论到了View树，然后编程就让我写View树。

 编程题是，因为在有的性能比较差的机型上，view树深度过大会造成crash，那么我想做一个提醒，进入APP后首先求出这个APP的最大View树深度，如果深度过大则进行提醒。其实就是二叉树的最大深度只不过换成了View树嘛，并没有什么难度。

主要问了ListView的优化、HTTP与HTTPS、进程与线程，编程题是给定一个char[]，然后翻转，其实就是翻转字符串啦，然后进一步的就是给出一个句子，翻转这个句子但是不翻转其中的单词，就是剑指offer那个，也很简单。

，说有十层楼，每层楼有一个钻石，钻石大小不一，没有规律，电梯只上不下，只能选一层下电梯，拿走该层钻石，问什么样的策略可以尽量大的拿到钻石。

说有1000瓶水，其中一瓶有毒，有10条狗，狗喝了毒水以后会在1小时内死去，问怎样能最快发现哪瓶水有毒。

100层楼，两个瓶子，用你的策略在最坏情况下最少扔多少次能测出来从几楼开始扔瓶子会碎

.（2）JVM如何加载一个类的过程，双亲委派模型中有哪些方法？

（4）HashMap和Concurrent HashMap区别， Concurrent HashMap 线程安全吗， ConcurrentHashMap如何保证 线程安全？

（6）

（7）

（8）JVM如何GC，新生代，老年代，持久代，都存储哪些东西？

（9）GC用的引用可达性分析算法中，哪些对象可作为GC Roots对象？

（10）快速排序，过程，复杂度？

（11）什么是二叉平衡树，如何插入节点，删除节点，说出关键步骤。

（12）TCP如何保证可靠传输？三次握手过程？

（13）TCP和UDP区别？

（14）滑动窗口算法？

（15）Linux下如何进行进程调度的？

（16）Linux下你常用的命令有哪些？

（17）操作系统什么情况下会死锁？

（18）常用的hash算法有哪些？

（19）什么是一致性哈希？

（20）如何理解分布式锁？

（21）数据库中的范式有哪些？

（22）数据库中的索引的结构？什么情况下适合建索引？

（23）Java中的NIO，BIO，AIO分别是什么？

（24）用什么工具调试程序？JConsole，用过吗？

（25）现在JVM中有一个线程挂起了，如何用工具查出原因？

（26）线程同步与阻塞的关系？同步一定阻塞吗？阻塞一定同步吗？

（27）同步和异步有什么区别？

（28）线程池用过吗？

（29）如何创建单例模式？说了双重检查，他说不是线程安全的。如何高效的创建一个线程安全的单例？

（30）concurrent包下面，都用过什么？

（31）常用的数据库有哪些？redis用过吗？

（32）了解hadoop吗？说说hadoop的组件有哪些？hdfs，hive,hbase,zookeeper。说下mapreduce编程模型。

（33）你知道的开源协议有哪些？

（34）你知道的开源软件有哪些？

（35）你最近在看的书有哪些？

（36）你有什么问题要问我吗？

基本上就这样了，还有一些零星的问题想不起来了，从7点到8点多，持续了一个多小时。我想说，我的手好累啊。。下次一定要戴个耳机。

又想起来几个。

（37）了解哪些设计模式？说说都用过哪些设计模式

（38）如何判断一个单链表是否有环？

（39）操作系统如何进行分页调度？

（40）匿名内部类是什么？如何访问在其外面定义的变量？

作者：惟愿无事  
链接：<https://www.nowcoder.com/discuss/3378?type=2&order=0&pos=31&page=1>  
来源：牛客网  
  
1 ArrayList和LinkedList有什么区别，能否序列化  
2 如何判断链表是否有环  
3 concurrentHashMap如何实现  
4 集群服务器 如何application 共享  
5 四个表 记录成绩，每个大约十万条记录，如何找到成绩最好的同学  
6 AIO与BIO的区别  
7 DirectMemory是否了解  
8 对JVM了解多少？我当时提到了内存模型 ，他问我工作线程的内存如何释放  
  
10 是否了解红黑树  
11 session的生命周期是多久  
12 关于Mina框架了解多少？（因为我在项目里用到了Mina，所以提到了这个部分）  
13 是否了解Linux

作者：Frank Sun  
链接：<https://www.nowcoder.com/discuss/6888?type=2&order=0&pos=60&page=1>  
来源：牛客网

Java篇：

2）HashMap和Map的区别，底层实现是什么？

3）对二叉树了解不？我说了解啊。他问有几种遍历方式？我说三种。他就说哪三种。追问了中序遍历的实现原理是什么？

4）多线程了解不？我说还好。他说两个进程同时要求写或者读，能不能实现？如何防止进程的同步？

Android篇：

5）在项目中有没有用到图片加载库？网络加载库？

6）问我JAVA哪些先进的东西给你在Android开发带来方便？我说JSON库。知道几种JSON库，有什么区别？

作者：jimbo  
链接：<https://www.nowcoder.com/discuss/5732?type=2&order=0&pos=71&page=1>  
来源：牛客网

* 介绍一下自己做过什么项目 ps：这里会根据项目问一些问题
* 跨线程通信 主要涉及LooperMessageHandler以及MessageQueue
* 手写代码：给你一个无限大的数(用字符串表示)，计算这个数加一以后的结果 手写代码在[这个网站](http://collabedit.com/)进行 ps：面试官发给你一个链接 你写代码面试官会同步看到

二面

* 手写代码 两个有序链表合并成一个有序链表
* Object中有哪些公有方法 ps：clone();toString();wait();notify();getClass();finalize();equals();hashCode();
* view的绘制过程

#### 阿里

* 做过什么项目没有 讲一下项目优化的地方
* 跨线程通信
* 夸进程通信 主要是Android的AIDL

#### 蘑菇街

* 说说做了什么项目
* 自定义view的几个步骤 怎么刷新view ps：onLayout(); onMeasrue();onDraw();
* RecycleView的优化
* 几种context的区别
* 看过什么源码没有
* 讲一下binder类
* Runnable运行在哪个线程里面 ps：开启一个子线程的唯一方法就是new Thread().start();
* raw和assets文件夹的区别
* activity关闭后尚未运行完毕的thread会怎么样 ps：会变成空进程线程，优先级最低 很容易被终止回收
* mainfest文件的合并规则

#### 网易

网易内推后笔试，过了笔试后去杭州面试 ps：报销来回车费 报销上限是600元 这个回来没记 可能不全

一面：

* 自我介绍
* 跨线程通信
* 讲了其中一个项目是怎么实现的 问jsoup怎么拼 = =
* 用过哪些开源库 讲一下他的架构是怎样的
* 图片三级缓存 内存缓存满了怎么办 ps：优先级队列 满了后根据优先级主动删除一部分图片 根据LRU算法确定优先级
* 讲一下JNI开发的过程 java怎么找到c函数 ps：JNINativeMethod保存函数对应关系 有兴趣的小伙伴可以查看我的这篇[博客](http://bornbeauty.github.io/2016/04/05/jni.html)
* 平时有什么爱好没有 = = ps：潜泳 可以潜好几天 然后自己浮上来
* \* 项目中使用了哪些材料设计中的东西？
* \* 为什么使用CardView，好在哪里？
* \* 项目中如何做到后台监听的？（项目中BroadcastReceive和Service相关问题）
* \* 使用的是什么数据库？（答：Realm）为什么使用，为什么不用Sqlite？
* \* 项目中使用那么多开源框架，如何获取这些信息的？如何筛选决定要使用这些东西？
* \* 有没有看过开源的代码？（大概说了一下Rxjava）
* \* 你有什么问题要问？
* 基本上就是这些，一面整体感觉不错，当天晚上七点半接到二面电话，由于在做360笔试，拖到第二天早上；
* 阿里二面（24分钟）
* \* 自我介绍，同时问了项目中的细小的问题；
* \* 项目中如何让service常驻后台？（告诉他这是我项目中的Bug被杀掉后就没办法了，但是在说了网上提到的QQ的两种常驻方法）
* \* Activity的生命周期，每个周期伴随着什么？
* \* Activity的启动模式，四种方法？
* \* 广播注册的两种方式，区别，优先级？
* \* 问了Layout的优化的方法？（这个不清楚，答的是ListView和Activity的优化）
* \* 一个singleton如何实现线程的同步问题？
* \* 平时如何代码是如何练习的？
* \* 使用过那么多开源框架，有没有看过源代码？（答：结合解析看过Retrofit、Rxjava、Okhttp等）
* \* 你有什么问题要问？
* 二面问的比较基础，基本上也都答上来了。二面结束第二天的下午接到三面电话；
* 阿里三面（45分钟）
* \* 自我介绍，同时问了三个项目的大概情况，（对混合开发那个比较感兴趣，问了一下，其他的没有细问）
* \* 由于其中一个项目是负责人，所以问了一下项目的整个设计流程
* \* 又问了项目中service常驻的方法，（回答和二面一样，但是这次说的比较清晰）
* \* 有没有了解过Realm的底层数据库是什么？（没有了解过）
* \* 他问有没有读过开源的源码？（说了一下Retrofit和RxJava的内部主要实现，这一次也答的比上一次清楚多了）
* \* 使用哪种数据传输格式？答Json，问还有哪些？答Xml。还有哪些？答：不知道
* \* 怎么解析Json？（答的在线解析生成Java类，他表示很感兴趣）
* \* 怎么在非UI线程更改UI？（答handle、AsyncTask、RxJava，表示现在我基本不用handle，直接用RxJava。）
* \* Handle的工作原理和工作过程？（同时还说了Handle会造成内存泄漏的问题）
* \* 有没有了解过跨进程通信；（瞬间心虚，只是简单说了一下远程service的创建和AIDL实现远程通信，这里需要感谢CSDN的郭大神）
* \* 问了多线程的并发（说了两种重入锁的实现）
* \* 又问使用synchronized关键字的时候底层调用的是哪种锁？（没有了解过不清楚）
* \* 问了一个JavaEE的新特性，这个不知道也没听清楚；
* \* interface中定义的变量默认的类型是什么（不加任何修饰）？（说是default，他反问default，我说这是方法中变量中的类型，interface中应该也一样吧；后来查了一下才知道，这里是个坑，interface只能是public static final的常量）
* \* 父类的方法是public，子类重写后，改为protect，会不会报错？反过来呢？
* \* 算法：国际象棋8\*8的格子，马走“日”字，设计算法计算马如果走完全部格子？（答的回溯法说了一下思路，后来查了基本差不多）
* \* 你有什么问题要问？
* 作者：追忆陌路  
  链接：<https://www.nowcoder.com/discuss/3122?type=2&order=0&pos=89&page=2>  
  来源：牛客网
* 4、进程与线程的区别
* 5、线程池的概念、好处、常见的线程池举例
* 8、ConcurrentHashMap原理
* Android部分：
* 1、Android的存储方式
* 3、AsyncTask 为什么java已经有线程池的实现了，还要继续使用AsyncTask，AsyncTask相对于java自带的线程池的好处
* 4、onTouch事件的处理机制
* 5、说一下MVC以及其在Android中的应用
* 6、说一下OOM以及你怎么解决的
* 7、了解Android的Framework么
* 8、说一下一个Android APP从点击打开开始，是怎样展示在用户面前的（Zygote、Ams、Wms等）
* 二面：
* 杭州现场面，一个月了路费还没报销回来。。。现场面是一个看起来很嚣张的面试官，看你的眼神从头到尾都充满了鄙视，从一开始的感觉就很不好。。
* 1、viewpager里面只能嵌套view吗 可不可以嵌套Activity
* 2、假如viewpager里面的每一页都有有很大数据量的内容，那么在快速的左右滑动时，如果出现了内存被回收的情况，如何处理 假如出现了OOM，怎么处理
* 3、同上情况，使用Fragment，又当如何？与viewpager有什么区别
* 4、ExpandableListView的Adapter怎么写
* 5、在ListView的每个item中如果可能出现的view都不一样，如何处理？动态获取view种类数量的话是不是就不能使用viewHolder进行优化？固定显示view如果不存在该种view就不显示的方法是否太耗内存
* 6、ListView的Adapter的getView具体是什么机制？
* 7、Activity之间传递大量的数据用什么方法
* Binder
* Bundle不够大
* Socket 在同一应用中没必要使用
* 共享内存
* static变量——同一应用中static变量如何控制，会不会并发出问题，如果Activity由于内存不足被kill掉，再重启的时候static变量的值还可靠吗：设置专门的访问类，进行读写控制，以及Activity重启后的初始化工作。
* 8、SQLite在使用时 假如ActivityA是用于修改SQLite，ActivityBCDEF。。。是用来展示SQLite的内容，如何在修改SQLite的同时使得BCDEF...都立刻获得最新的数据？SQLite有没有类似于ContentObserver的这么一种监视类可以监听到SQLite内容的改变？如果广播通知所有类，那么广播可以携带多大的数据？
* 9、假设把一个ActivityB向右滑动可以回到上一个ActivityA，B渐渐透明，A渐渐清晰，要怎么编写页面？说说你对android:theme的理解，说说你对alpha值的理解
* 10、ListView在数据量很大图片很多的情况下怎么优化？假如一个图片，轻轻的向上滑动一丢丢，那么需要重绘吗？（什么鬼。。。）
* 11、详细展开说一下所有LaunchMode的应用场景
* 假如A-》B-》C，想让C后退直接到A，使用什么样的intentflag？
* 假如A-》B-》C，C使用singleTask，C后退，后退到什么地方呢？
* 12、在全屏显示高清大图的时候，网速慢的情况下如何显示？渐进式显示是怎么做到的？如果想显示下载进度圈圈，该如何实现？
* 13、AsyncTask在4.x以后有什么改变？怎样改回并发执行好多个？如果一个AsyncTask结束取得结果之前Activity就因为内存原因被Destroy掉了，那会有什么情况发生？会内存泄露吗？会空指针吗？需要在Activity彻底死掉之前把AsyncTaskcancel掉吗？如果没有cancel掉，然后Activity重启了，那这个Asynctask又当如何呢？
* 14、AsyncTask内部实现机理 与Thread+Handler有什么不同
* 16、你的APP里，是每个Activity都有一个Handler呢还是所有Activity共享一个Handler
* 17、基于事件监听、基于事件回调 假如一个touch事件发生，那是监听器先收到还是onTouch函数先收到
* 18、假如你要记录ListView滚动到的位置，要记录什么信息，view怎样获得坐标信息
* 19、说一下canvas
* 20、dialog和popupwindow的应用场景 如果popupwindow在没有dismiss的时候Activity finish了，会出现什么情况？
* 21、了解过什么三方库？为什么不用三方库？
* 22、你觉得安卓的未来在哪里？你自己为什么都用IOS了(╯‵□′)╯︵┻━┻
* 23、Service和推送通知，通知有没有可能出现有推送但是通知栏收不到通知？service被kill掉会如何？如何保证service不被kill掉
* 24、了不了解Scrollview scrollview和ListView有什么相似点 有什么不同 那如果这两个是继承关系 那应该是谁继承谁？
* 25、dp sp px各自的应用场景是什么 换算关系是什么 使用sp的时候如果调整系统字体，显示字体会跟着调整吗？
* 26、gallery为什么被废弃？为什么Google推荐使用viewpager代替gallery呢？有什么改进？
* 27、handler.postAtTime不是延时post么 那handler怎么延时处理Message
* 笔试（阿里游戏）（3月16日）   
  1. 基础[**数据结构**](http://lib.csdn.net/base/datastructure)与算法知识   
  2. looper，handler等原理   
  4. 与运算符   
    
  5. 设计模式与设计原则   
  设计模式：—   
  设计原则：   
  1）单一功能：对象仅具有一种单一功能概念   
  2）开闭原则：软件提对于拓展开放，但是对于修改封闭   
  3）里氏替换：程序中对象可以在不改变程序正确性的前提下被它的子类所替换的   
  4）接口隔离：多个特定客户端需要有宽泛用途的接口   
  5）依赖反转：依赖于抽象而不是一个实例   
  6. A向B发送消息，A能实时看到B的阅读时间，问A实现方式
* 阿里内推一面：（3月18日1个小时）   
  1. volley层次结构，原理，为什么选用volley，不用其他开源框架，对开源框架的看法   
  2. MVC与MVP的区别，activity，fragment扮演的角色，presentor之间的通信问题。   
  3. 进程间的通信，binder的原理，intent与aidl的应用场景及相关比较   
  4. 图片如何缓存处理，如何处理oom   
  5. 自定义view实现，注意事项
* 阿里内推二面：（3月18日1个多小时）   
  1. 发出的请求是否可以终止请求，pc网页刷新与停止的原理，跟移动浏览器的区别   
  2. 加载图片请求的超时时间设置，多少秒合适，原因，大图如何加载。   
  3. 线程间的通信，并发问题解决方案。   
  4. 快速排序为什么叫做快排，其算法复杂度，及其实现原理   
  5. 应用签名是为了什么，签名原理是什么   
  6. root之后有什么重大的变化，开发者可以操作什么   
  7. 给出九游android客户端程序，解释下相应的布局，listView做大面积的嵌套，下拉刷新的实现   
  8. 渠道提示不合法的问题，为何系统能识别渠道非法

1. 线程间通信，进程间通信，binder原理   
2. sharepreference原理   
3. 事件分发原理   
4. 设计模式，延迟加载实现单例模式，内部类来解决，加锁也不稳   
5. mvp与mvc   
6. 线程池调度实现，不能用executor（当时居然以为blockingqueue存线程呢=\_=)   
7. 动态加载，dex分包，热修复   
8. 缓存原理实现，内存如何分配   
9. 如何减小apk体积   
10. 性能优化从哪几方面考虑（内存，cpu，fps，ddms灵活dump)