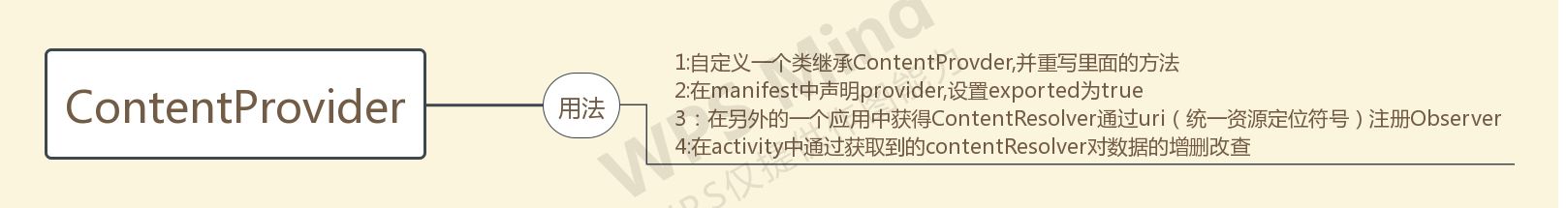
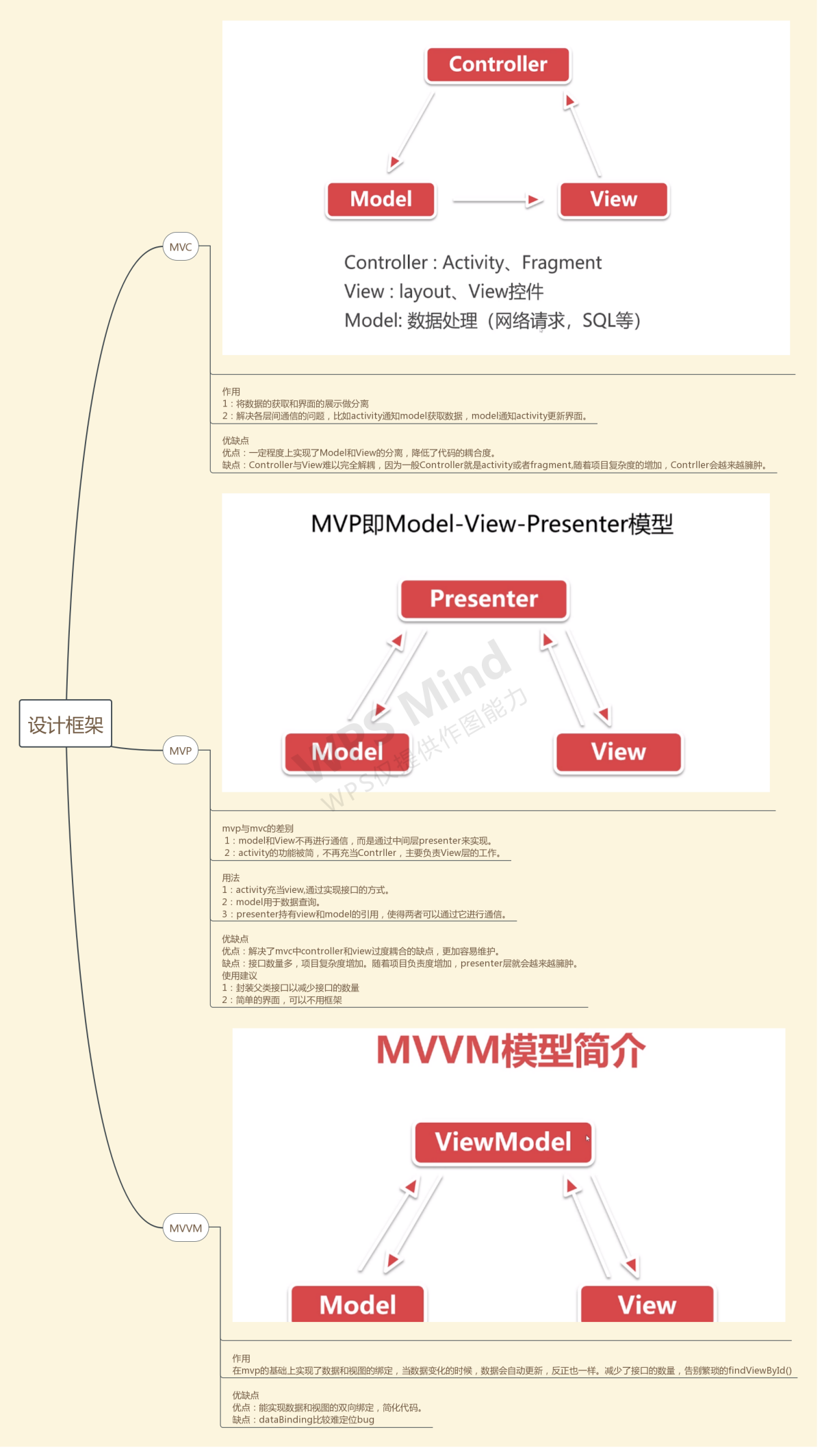


Ndk-jni









设计框架 mvc\mvp\mvv

线程同步的方法 ，原子性、有序性

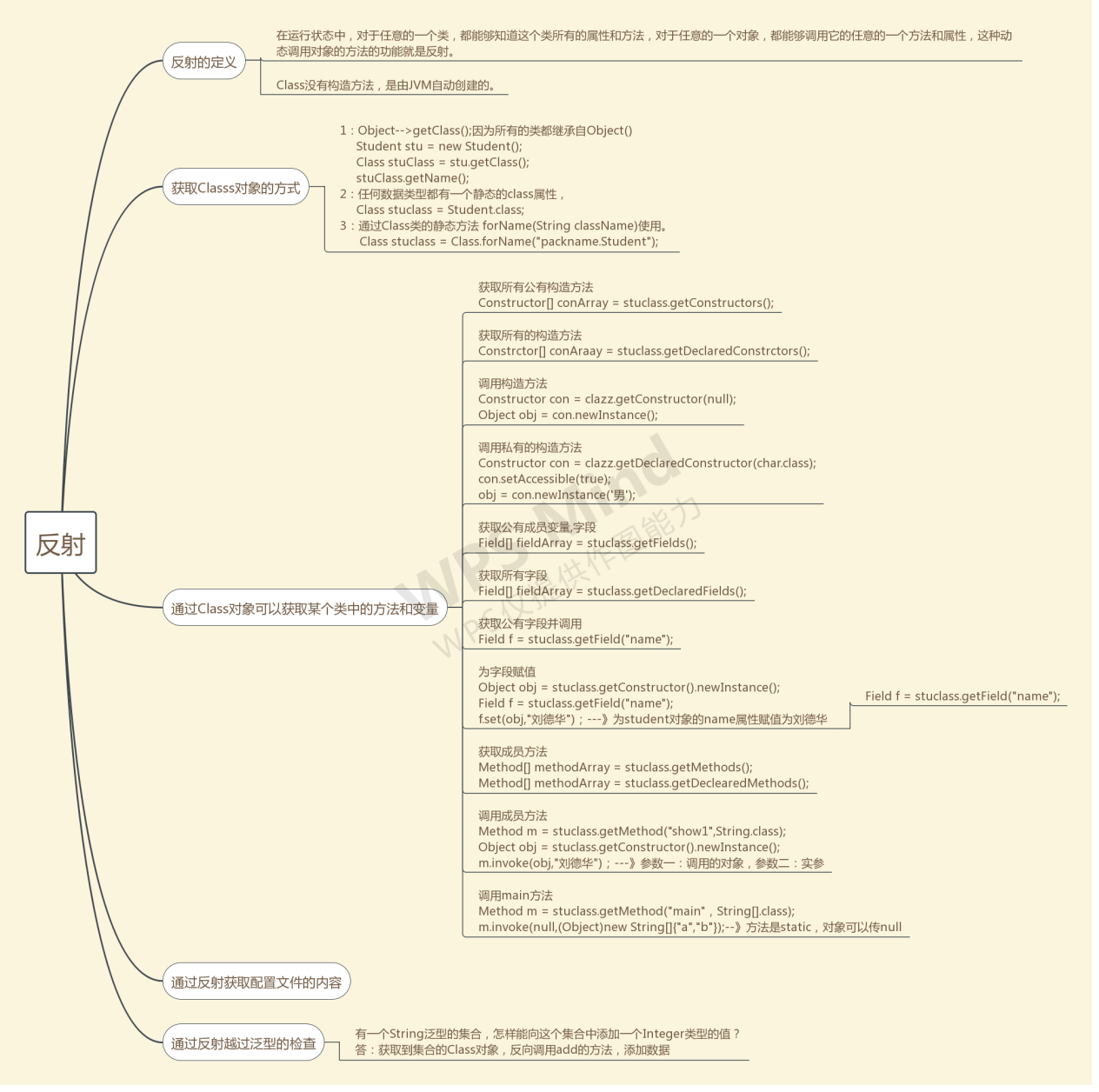


Android 五大存储方式

C++

梳理项目难点：送餐、im、升级、so、jni、sdk、封装

Sdk适配，sdk封装：泛型、反射、接口回调、设计模式：单例，工厂，代理，适配器，builder模式

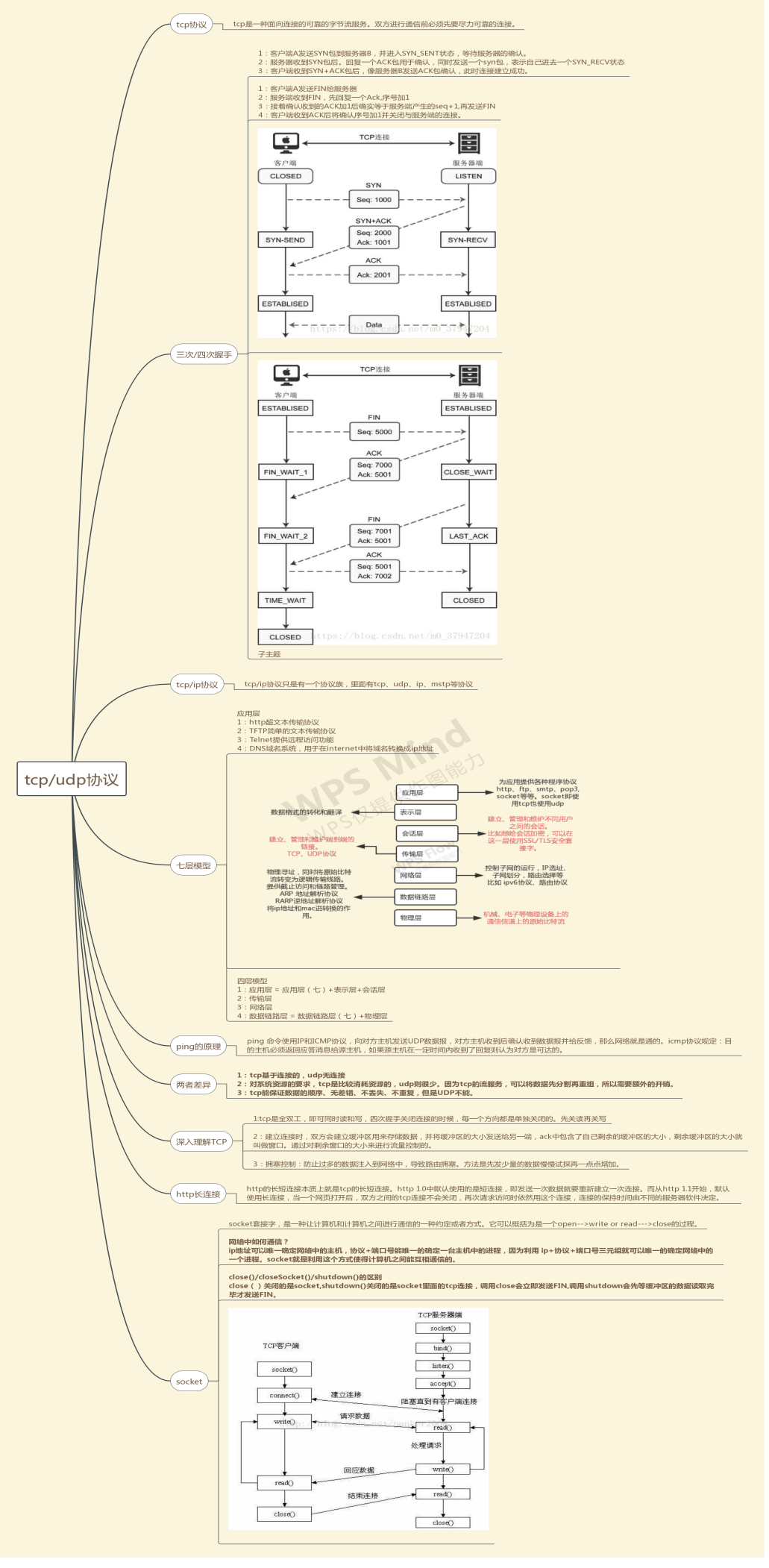


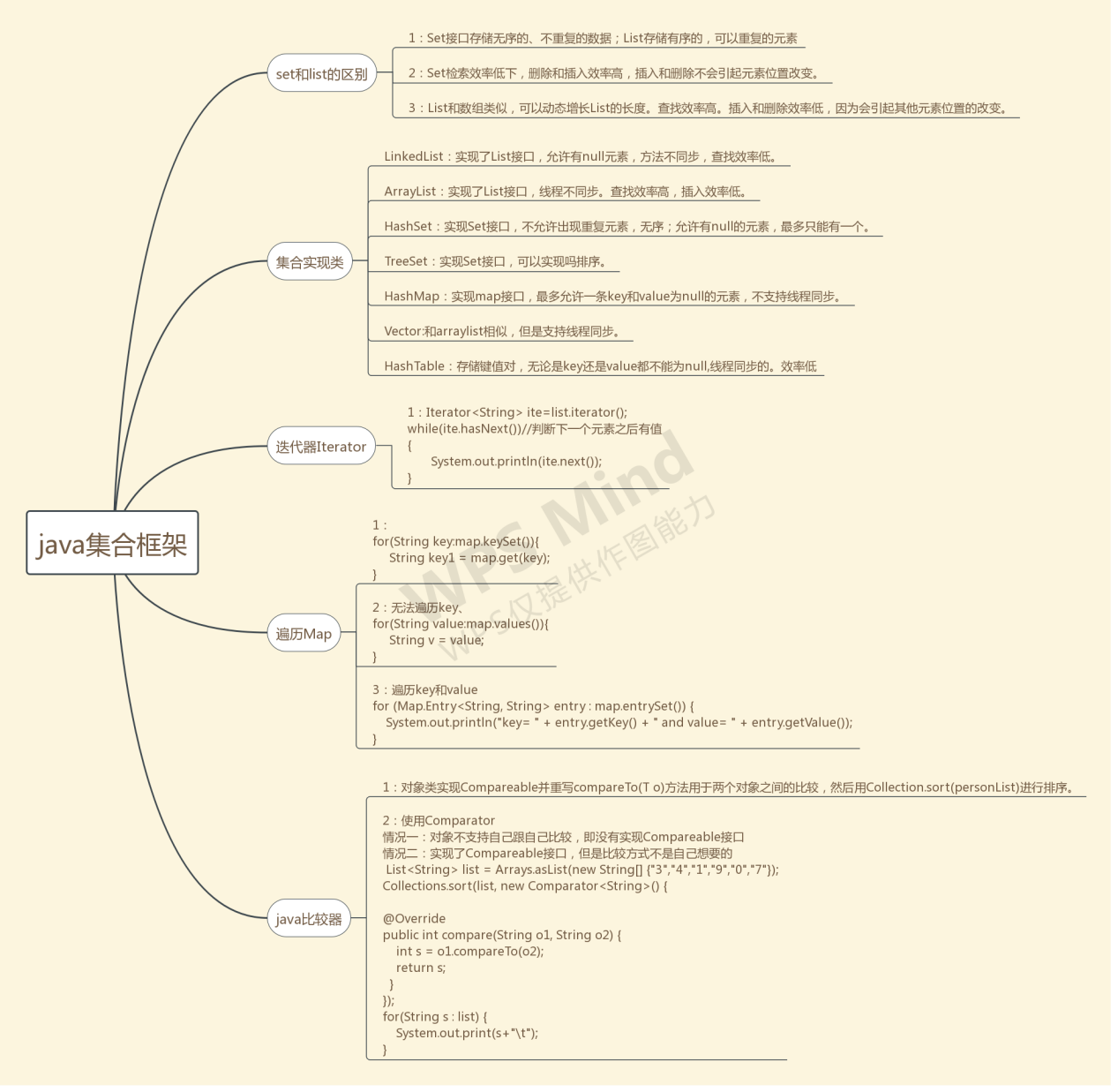
开源框架retrofit2+RxJAVA,resfult(幕课)

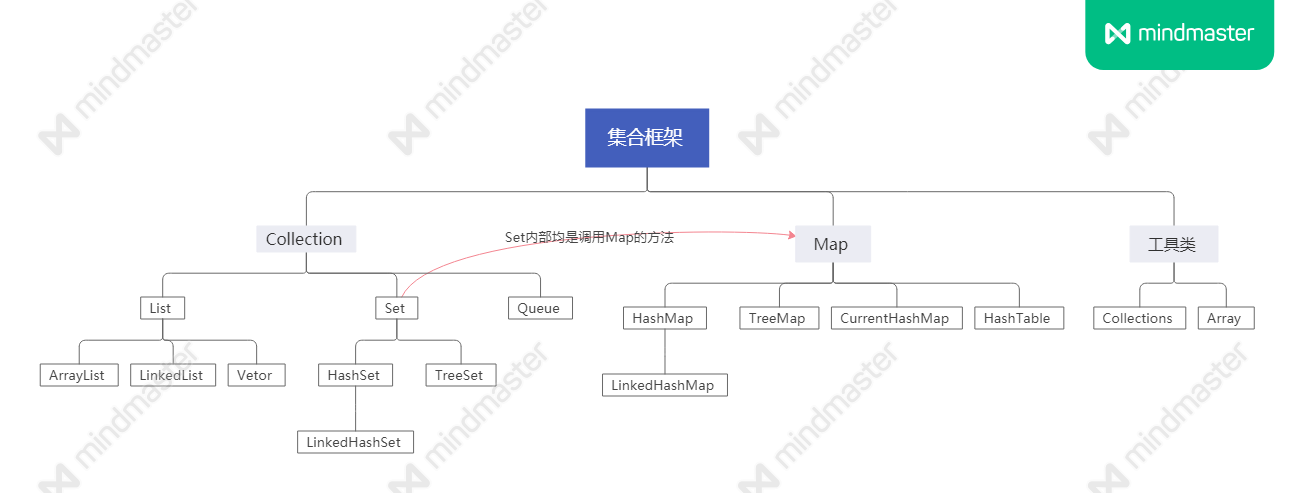
Listview的优化:commonViewHolder、commonAdater,checkBox抢占焦点问题、checkBox复用错乱问题（幕课）

数据库

http协议，通信协议,socket

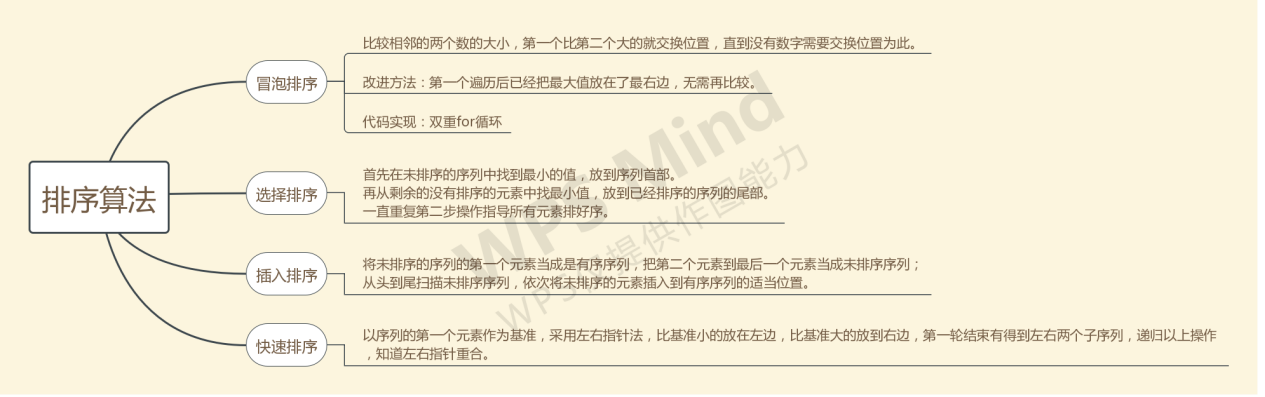




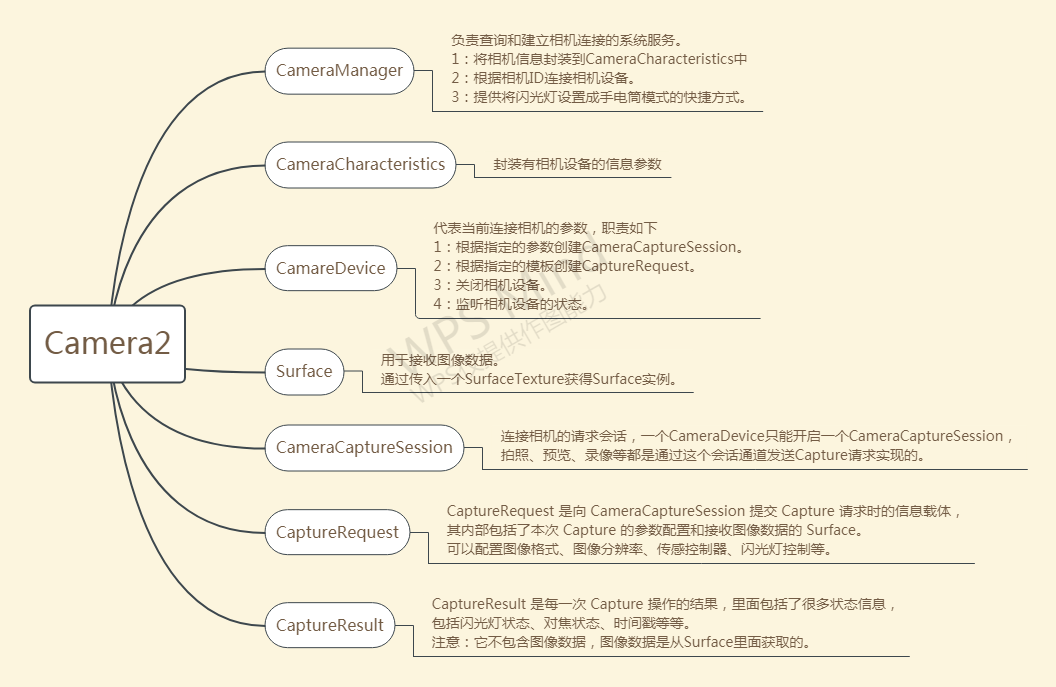


Git

几种简单的排序方式：快排、二分法、冒泡等



Activty的启动模式



1. **==、equals和hashcode的区别**

==对于基础数据类型来说比较的是数值是否相当，对于一般的引用对象来说比价的是内存地址是否相等，Object类的源码==比较的就是内存地址。

Object的equals比较的也是引用对象的内存地址，但是对于一些重写了equals方法的来说，比如String类、Date类等，他们的equals方法比较的是字符串的内容是否相等。

Hashcode主要用于集合对象的比较，如果一个集合中不存在某个对象的hashcode就往里存，如果存在了hashcode就使用equals进一步比较，这样可以提高效率。

如果重写了equals方法，就必须重写hashcode方法，因为equals默认比较的是对象的地址，而想要比较两个对象的内容是否相等，就必须保证这两个对象的hashcode是一致的，重写hashcode的方法：result = 31 \* result + filed.hashcode。

**2,int和Integer的区别**

Int是基本数据类型，integer是包装类，integer需要实例化才能使用，int的默认值是0，integer默认值是null。

**3，谈谈多态的理解**

思路：多态是什么？有什么作用？具体体现在哪？实现方式？

多态是对象多种表现形式的体现，同一个接口不同的实例对象执行表现出不同的结果。

多态能降低耦合度、具有可扩展性、灵活性和简化性等作用。具体体现在继承、重写、对象转型这几个主要方面。

实现方式有重写：子类继承父类，重写父类方法；定义接口，具体实现类实现接口；定义抽象类和抽象方法，子类继承抽象类实现抽象方法。

**4，String、StringBuffer、StringBuilder的区别**

String会创建一个新的对象指向改变的字符串，一旦创建了新的String对象，它的值就无法改变了；StringBuffer和StringBuilder的对象能被多次修改，并且不会产生新的对象，效率比较高；两者区别是StringBuffer是线程同步的，StringBuilder不是线程安全同步的，所以StringBuilder的速率比StringBuffer又更高。

**5，什么是内部类？作用？**

将一个类定义在一个类里面或者一个方法里面，称为内部类；分为成员内部类、局部内部类、匿名内部类和静态内部类。作用是内部类能独立继承或者实现接口，不受外部类的影响，使多继承的方案个更加完整；方便将类之间组织在一起，但是又可以对外界隐藏；方便事件驱动程序，是代码结构简洁清晰。

**6，抽象类和接口的区别**

1：抽象类可以有具体实现的方法，而接口只能定义方法的声明（降低耦合度）。

2：抽象类的成员变量可以是任何类型，而接口只能是public static finnal

3：抽象类可以有静态代码块和静态方法，而接口不能有，**jdk1.8后接口可以有静态方法，但是必须由方法体**。

4：一个类只能继承一个抽象类，但是可以实现多个接口。

5：抽象类不能被实例化，而接口可以。

**7，抽象类和接口的应用场景**

抽象类是一种对事物属性和构成的抽象，接口是对一种行为或者目的进行抽象。使用场景看业务和功能定，比如存在多个子类具有一些相同的属性，可以进行复用的，就可以定义抽象类；不同的子类需要实现不同的行为效果，那就使用接口。

**8，泛型中extends和super的区别**

比如<? extends Fruit>代表的是上界通配符，就是说这个泛型匹配的对象是Fruit和它的子类，可以访问Fruit和它的子类，适用于需要频繁读取的操作；

<? super Fruit>是下界通配符，匹配的是Fruit的父类，get的时候返回是的Fruit的父类，但是具体是哪个父类不知道；set的时候不需要确切的类型，因为都能向下转型为Fruit类。适用于需要频繁插入的操作。

**9，父类的静态方法能被重写吗？**

不能。因为静态方法是类加载的时候就已经存在了的，指向了一个固定的地址，只跟类有关，跟实例对象无关，就算子类里面存在一个跟父类一样的静态方法，它指向的也会是另外的一个地址，即那是完全单独的方法。

**10，进程和线程的区别？**

1：进程是资源分配的最小单位，线程是程序执行的最小单位。

2：进程有自己独立的地址空间，线程只是共享进程的数据，使用的是相同的地址空间。也就是一种包含的关系，一个进程包含有多个线程。

堆：是大家公有的空间，在系统对进程初始化的时候分配。

栈：是线程独有的，在线程初始化的时候创建。

**11，序列化的方式有哪些？**

序列化是将对象转化为二进制流，便于持久化保存。序列化方式有以下几种

1：实现Serializable接口，会自动序列化所有非static和非transient关键字修饰的成员变量。

2：实现Externalizable接口，必须实现writeExternal和readExternal方法，可以手动选择哪些字段需要序列化。

3：实现Serializable接口，添加writeObject和readObject方法，也可以实现手动选择序列化字段。

4：json序列化，fastjson等方式。

**12，Serializable 和Parcelable 的区别**

Parcelable 是Android特有的api,效率更高，内存开销更小。实现Parcelable 接口，重写writeToParcel方法实现序列化，通过静态成员变量CREATOR实现反序列化。主要用来activity、fragment间传递复杂对象以及用在AIDL数据传递。

**13，静态方法和静态属性是否可以被继承？是否可以被重写？**

可以被继承，不能被重写。原因同9

**14，静态内部类的设计意图**

1：静态内部类的创建不需要依赖外部类

2：静态内部类不能使用外部类的非静态属性和方法。

**15，java实现多态的机制是什么？**

多态简单说就是一个引用指向不同的实例化对象，变现出不同的行为。Java程序在运行过程中才决定让引用变量绑定到哪个实例化对象上，所以可以支持多态性。

**16，说一说对反射的理解**。

反射是指的是在运行的过程中，对于任意一个类都能知道它的全部属性和方法；对于任意一个对象，都能调用它的属性和方法。

**17，String为什么要设计成不可变的？**

1：基于安全考虑，不可变对象对于多线程环境来说是安全的。

2：可以节省堆空间，java提出了一个字符串池的概念，就是已经存在了一个字符串，就不会再去创建它的字符串对象，而是直接返回它的引用。

1. list、set、map的区别？hashMap实现原理？hashMap数据结构？hashMap如何put数据？ArrayMap和HashMap的区别？hashTable的实现原理？treeMap实现原理？hashMap和hashtable的区别？hashMap和hashSet区别？他们如何判断重复的元素？怎么防止碰撞？ArrayList和Linklist的区别？数组链表区别？二叉树遍历？堆的结构？堆和树的区别？

**19，开启线程的三种方式**

第一种：新建一个线程类继承Thread，并start()方法启动该线程类。

第二种：新建一个线程类实现Runnable接口，重写run（）方法，在new Thread（）中传入该线程类的实例化对象并使用start启动。

第三种：使用匿名内部类创建new Runable()接口。

**20,线程run()和start()的区别**

使用run()方法，系统会把它当成是普通的方法，立即就执行了，也不会真正的创建的一个子线程，还是会运行在主线程上；使用start()方法，系统会把run()方法当成一个线程来处理，会创建子线程从而能达到异步的作用。

**21,sleep和wait的区别**

Sleep：让当前线程休眠指定的时间；不释放已经获取的锁资源；可通过interrupt()方法唤醒休眠线程；作用于当前线程；静态方法

Wait:让当前线程进入等待状态并释放已经获取到的锁资源；当别的线程调用notify()或者notifyAll()，当前线程将进入就绪状态；只能在同步上下文中使用；作用于对象本省，是个实例方法。

**22，Wait/notify的用法**

Notify()和notifyAll()的作用是唤醒当前对象上的等待线程，前者唤醒单个，后者唤醒所有。（写代码）。

**23，为什么会线程阻塞？**

当某一时刻，一个线程在访问某一个资源时另外一个线程也需要访问该资源，在第一个线程访问结束前，第二个线程无法获得锁，这个时候称为线程阻塞。

出现阻塞的原因：一：线程睡眠且不释放锁；二：线程被优先级更改的线程插入进入阻塞状态；三：线程join()进入阻塞状态，直到另外一个线程执行完。

**24，停止线程方式**

一：设置停止标记位，检查到true就不再执行；

二：使用stop()废弃的方法。

三：使用thread.interrupt(),并不能立即中断依然会等当前线程执行完，所以在使用interrupt()前需要使用Thread.currentTread().isInterrupted()判断标志位，才能中断当前线程。

**25，synchronied修饰对象**

修饰实例方法，是对调用该方法的对象加锁。

修饰静态方法，是对该类对象加锁。

修饰代码块，是对调用该代码块的对象加锁。

**26，Comparable和Comparator比较**

Comparable是排序接口；若一个类实现了Comparable接口，意味着该类支持排序，相当于一个java的内部比较器。Comparator则是一个外部比较器，是一个比较接口，可以控制类的次序。

**27，static环境中是否可以访问非static变量，static方法是否可以被覆盖**

当java虚拟机对类进行加载的时候，就会对static变量进行初始化，非stati变量此时是还没有被创建出来的，所以不可以被访问到。

Static方法不能被覆盖，因为方法的覆盖是基于运行时，而静态方法是编译时就绑定了的，且static修饰的方法跟类的实例不相关，所以不能覆盖。

**28，泛型**

使用泛型的好处是代码重用和可以省去强制转换，在编译的时候就能检查类型安全。

常用的泛型类型：E(元素element)、K(关键字key)、N(数字number)、T(类型type)、V(值value)、？(不确定的类型)

泛型方法的定义：public static <T> T test(T t){……}

泛型接口的定义：public interface GenericIntercace<T> {

T getData();

}

<https://www.jianshu.com/p/986f732ed2f1>

**29，获取Class类的三种方式：**

1. 使用Object类的getClass()获取

Class cla = xxx.getClass();

1. 使用Class类的静态方法forName()获取

Class cls = Class.forName(“xx.xxx.xx”);

1. 直接Class cls = xxx.class;

**30,java内存模型**

**堆区：**存储对象实例，不存对象的引用。每一个JVM(java虚拟机)只有一个堆区。所有线程共享。GC发生的主要场所。

**栈区：**存储基本数据类型和对象的引用，每个线程有一个栈区，不共享。

**方法区：**所有线程共享，存储静态变量。常量池是方法区的一部分。

显式的字符串常量放在常量池里面，字符串对象放在堆中。（常量池从JDK1.7开始移到了堆里面，以前在方法区）

例子：

String a = "holten";

String b = "holten";

第一句代码执行后就在常量池中创建了一个值为holten的**String对象**；

第二句执行时，因为常量池中存在holten所以就不再创建新的**String对象**了。

此时该字符串的引用在虚拟机栈里面。

**31，final、finally、finalize的区别**

Final修饰类的时候表示该类不允许被继承、修饰方法的时候表示不允许子类对该方法进行重写、修饰变量表示该变量不可更改。

Finally是异常处理的一部分，在try语句执行了的情况下，finally最终会执行，可用于释放资源操作。

Finalize（）是Object的一个方法，在必要的时候程序员可以使用它来启动GC操作。

**32，序列化的方式有哪些**

使用场景：在进行Android开发的时候，没办法将对象的引用传递给Activity或者fragment,需要将对象放到Intent或者Bundle里面，然后再进行传递。

一：实现Serializable接口，注意这种方式不能序列化静态成员变量和transient修饰的变量。需要使用到ObjectOutputStream和ObjectInputStream

以及writeObject/readObejct方法。注意，当进行反序列的时候，JVM会把字节流的seriaVersionUID与本地实体类的seriaVersionUID进行比较，如果不一样的会造成反序列化失败，所以需要手动生成系列号。

二：实现Parcelable接口,Android独有，效率更高。作用是可以永久性保存对象、可以把序列化对象在网络中传播、可以在进程间传递对象。Activity可以通过putExtra()传入序列化好的对象，通过getParcelableExtra()获取对象。

两者区别：第一种方式能很好地将数据保存在磁盘上，但是会引起频繁的GC；第二种方式效率高。

**33，java异常体系**

运行时异常不需要程序员捕获，比如空指针异常、数组越界异常、强制转换异常、数据格式异常等；编译时异常需要程序员做异常处理，否则编译不通过，比如FileNotFound、IOExcepetion、SqlExcepation。

**34，说一下多态（什么是多态？优点？存在的必要条件？实现方式？向上/向下转型->代码）**

多态是指一个引用变量在运行期间才决定指向那个类的对象，从而体现为一个接口具有多种不同的变现形式。优点：解耦、可扩充、灵活

存在的必要条件：继承、重写、父类引用指向子类对象。

实现方式：重写、接口、抽象类和抽象方法

**35，说一说反射机制（代码实现）**

反射是在运行状态中，对于任何一个类或者对象可以知道他们的所有属性和方法，还能调用。

1. **List,Set,Map的区别**

List接口实例存储的是有序的，允许重复的元素；Set接口实例存储的是无序的，不允许重复的元素；Map存储的是键值对，key值不允许重复，不允许为null；

1. **List和Map的实现方式和存储方式**

List的实现方式有两种，一种是基于一维数组实现，另一种是基于链表实现。

Map是基于hash表实现的，以键值对的方式存储。

1. **HashMap的实现原理**

HashMap的实现原理是散列表，即数组+链表的形式进行数据的存储。

首先：利用key的hashcode计算出当前元素在数组中的下标；

存储时，如果出现hash值相同的两个key,那么进行判断，如果存在的key与要存储的key相同，那么就直接覆盖value;如果没有存在的key与当前key相同，那么将当前的key-value放入链表中。

读取时，根据hashcode找到对应的数组下标，再判断key值，如果是链表就遍历找node，如果是红黑树就遍历红黑树找node,从未找打value;

当出现hash冲突的时候，会进行resize数组扩容优化。

1. **hashMap如果put数据，从源码的角度解释**

1：根据hash(key)算法得到hash值；（源码是用hashcode与hashcode右移16位得到的值进行异或运算从而降低hash冲突的概率）

2：判断table数组是否是null或者长度为0，如果是就resize()进行数组扩容（扩容原则是默认大小为16，当前size>16\*加载因子了，则进行扩容乘以2）。

3：如果数组的第i位是null,直接将新建的结点插入，插入后判断当前存在的键值对数量是否超过了阈值，超过则进行扩容；

4：如果数组的第i位不是null,则遍历判断key值是否存在相同，相同则覆盖value,不同话继续判断是否需要转换为红黑树；

5：如果链表的长度大于8了就转换为红黑树，在红黑树中插入；如果不需要转换为红黑树，则添加到链表末尾。

1. **ConCurrentHashMap的实现原理**

<https://www.jianshu.com/p/865c813f2726>

ConcurrentHashMap的实现原理

jdk 1.7:segment+数组+链表（分段锁机制，即每个segment都有一组或多组散列表）

缺点：需要进行两次hash计算，效率肯定比hashmap低

优点：只需要对元素所在的Segment进行加锁，不影响其他Segment

JDK 1.8:CAS（比较再交换）+数组+链表、红黑树

使用CAS+Synchronized保证线程安全，对数组进行加锁。

1. **如何提高ConcurrentHashMap的插入效率？**

将大批量数据保存到map中有两个地方的消耗将会是比较大的：第一个是扩容操作，第二个是锁资源的争夺。第一个扩容的问题，主要还是要通过配置合理的容量大小和扩容因子，尽可能减少扩容事件的发生；第二个锁资源的争夺，在put方法中会使用synchonized对头节点进行加锁，而锁本身也是分等级的，因此我们的主要思路就是尽可能的避免锁等级。所以，针对第二点，我们可以将数据通过通过ConcurrentHashMap的spread方法进行预处理，这样我们可以将存在hash冲突的数据放在一个组里面，每个组都使用单线程进行put操作，这样的话可以保证锁仅停留在偏向锁这个级别，不会升级，从而提升效率

1. **ArrayMap和HashMap的区别**

<https://blog.csdn.net/qq_28845393/article/details/87601753>

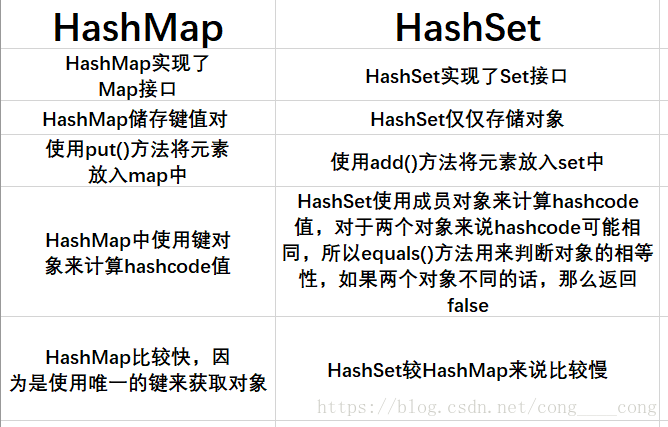
ArrayMap:两个一维数组结构，android独有api

优点：占用更少的内存，扩容时因为只需要拷贝，所以扩容更高效

缺点：数据量大的时候查询效率比hashMap低，存取效率低，不可序列化

1. **HashMap与HashSet的区别**

HashSet内部还是通过hashMap来实现。Hashset仅仅存储对象,用对象来计算hashcode,效率上比hashMap低。



1. **ArrayList和LinkedList和Vetor的区别，以及应用场景**

| **类型** | **线程安全** | **内部结构** |  | **扩容规则** | **执行效率** | **序列化** |
| --- | --- | --- | --- | --- | --- | --- |
| ArrayList | 否 | 数组Object[] | 10 | 数组足够最小长度\*1.5 | 高 | 是 |
| Vector | 是 | 数组Object[] | 10 | 默认数组足够最小长度\*2，可自定义每次扩容数量 | 低 | 是 |

| **类型** | **内部结构** | **插入效率(正常情况)** | **删除效率(正常情况)** | **顺序遍历效率** | **随机遍历效率** | **占用内存** | **序列化** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ArrayList | 数组Object[] | 低 | 低 | 高 | 高 | 低 | 是 |
| LinkedList | 双向链表Node | 高 | 高 | 高 | 低 | 高 | 是 |

问：插入效率为何说正常情况下ArrayList低，LinkedList高呢？

答：我们清楚ArrayList之所以插入效率低，有两个原因会造成时间的消耗。

​ 第一，当底层数组空间不足时需要扩容，扩容后需进行数组拷贝

​ 第二，当不在数组末尾插入数据，那么就需要移动数组元素

​ 知道了其插入效率低的原因后，那么很明显，数据扩容及拷贝只有在数组空间不足时才发生，如果我们正确使用，就像《阿里巴巴Java开发手册》中提到我们在创建集合对象时，就传递参数预先设置好数组大小，那么插入效率是非常高的；而90%的情况下我们在添加元素时都调用的是add(E e)，直接在末尾添加元素，很少调用add(int index, E e)在数组中部添加元素，这样其实移动数组元素就很少发生，因此插入效率也很高。

问：删除效率为何说正常情况下ArrayList低，LinkedList高呢？

答：因为删除效率高、低不是绝对的。其实删除操作可以分为两部分。

​ 第一：找到要删除的元素，这个通过索引找，ArrayList的执行效率要远高于LinkedList的执行效率；通过equals找则需要遍历整个集合，ArrayList和LinkedList执行效率基本一致。

​ 第二：删除元素及后续操作，这个如果删除是最后一个元素，执行效率基本一致；如果是删除的中间元素，那么ArrayList需进行数组元素移动，而LinkedList只需搭建起该元素的上一个节点和下一个节点的关系即可，LinkedList执行效率高于ArrayList。

​ 因此，需根据实际情况才可判断实际的执行效率。

问：遍历效率这个问题怎么说？

答：ArrayList通过数组实现，天然可以通过数组下标读取数据，顺序遍历、随机遍历效率都非常高；LinkedList通过双向链表实现，顺序遍历时，可直接通过本节点.next()直接找到相关联的下一个节点，效率很高，而如果LinkedList随机遍历时，首先需判断（传递的索引值与集合长度/2）的大小，来确定接下来是应该从第一个节点开始找还是最后节点开始找，越是靠近集合中部、集合越大，随机遍历执行效率越低。

1. **LinkedList随机遍历的机制**

比较传入的index索引与集合的长度size/2，如果index小，那么从第一个顺序循环，直到找到为此；如果index大，那么从最后一个元素开始找，找到为此。所以越靠近中间的元素，遍历次数越多，效率也越低；

1. **什么是浅拷贝，什么是深拷贝**

浅拷贝：会创建一个新的对象，有着原始对象属性值的一份拷贝，如果是基本类型，那拷贝的是数值；如果是引用类型，拷贝的就是内存地址，因此如果一个对象改变了这个地址，就会影响到另一个对象；

浅拷贝实现方式:实现Cloneable接口，并覆写clone()方法，通过super.clone。

深拷贝：会创建新对象，对于基本数据类型会拷贝数值；如果是引用类型，会开辟一段新的内存空间，然后拷贝内容；对于多层对象嵌套的，每个对象都需要实现Cloneable对象并重写clone（）方法。速度慢，内存花销比较大。

深拷贝实现方法:每一层实现浅拷贝，最后再顶层的类中的clone方法中调用所有类的clone()方法即可实现深拷贝。

1. **synchronized和ReentrantLock的区别**

（1）synchronized是独占锁，加锁解锁过程都是自动进行；reentrantLoack也是独占锁，但是加锁和解锁过程都是手动的。

（2）Synchronized可重入，因为加锁解锁是自动，所以不用担心最后锁不会被释放；reentrantLoack因为是手动进行的，且次数需要一样，否则其他线程无法获取锁。

（3）Synchronized不可以响应中断，获取不到锁的时候就一直等待；reentrantloack可以响应中断。

（4）reentrantLock可以实现公平锁机制，也就是在锁上等待时间最长的锁将优先获得锁的使用权。

<https://baijiahao.baidu.com/s?id=1648624077736116382&wfr=spider&for=pc>

1. **为什么要用进程而不是仅仅用线程？**

1：因为进程在同一时刻只能干一件事，不能同时干多件事。

2：进程如果被阻塞就会被挂起，即使有一些不用依赖某个让进程挂起的操作，也无法执行。

1. **wait和sleep的区别**

1：sleep是线程中的方法，wait是Object的方法。

2：sleep方法不会释放锁，wait会释放锁，而且会加入到等到队列中。

3：sleep方法依赖于synchronized来实现同步，wait不需要

4：sleep不需要被唤醒，wait需要被唤醒。

1. **wait()方法一定要使用sycronized进行同步吗？不用sycronized修饰会有什么问题？wait()方法会释放对象锁，那么这里指的锁是什么？ wait()会释放对象锁，而sleep()不会释放对象锁，这在实际情况中有什么区别？**

<https://www.jianshu.com/p/1dafbf42cc54>

1：wait一定要使用synchronized修饰，因为wait会释放锁，没有synchronized修饰的话就没有锁，没有锁还释放锁就会报错；

2：锁指的是synchronized修饰的方法、对象或者代码块等

3：wait释放锁，其他线程可以自行synchronized修饰的内筒。

**51：如何关闭线程？**

1：当线程被调用sleep或者wait等方法进入阻塞状态的时候，若此时调用线程的interrupt()将线程的中断标志设为true,由于处于阻塞状态，中断标记会立即被设置为false,同时产生一个InterruptException异常，例如在while循环中不断的执行任务，调用interrupt()会产生异常中断，这时异常在while循环外捕获，这样就退出了(**当线程处于阻塞状态的时候可以用这种方法**)。

2：使用中断标记。isInterrupt（）是判断线程的中断标记是不是true,当线程处于**运行状态**，调用interrupt()方法后，isInterrupt()会返回true,此时就可以退出while循环。（**当线程处于运行状态的时候可以用这个方法**）

3：添加额外的标记。通过一个flag标记，默认为false,并提供一个stopTask()方法，当调用stopTask()方法的时候，修改flag的值从而退出循环。

**51，如何保证线程的安全性？**

线程的安全性体现在三个方面

1原子性：提供互斥访问，同一个时刻只能有一个线程对数据进行操作。（atomic,synchronized，reentrant）

2可见性：一个线程对主存的修改可以及时的被其他线程看到。（synchronize，volatile）

3有序性：执行执行顺序。

**52,java中对象的生命周期**

<https://blog.csdn.net/yanliguoyifang/article/details/80964237>

1. **synchronized的用法和原理**

<https://www.cnblogs.com/lukelook/p/9946065.html>

<https://www.cnblogs.com/greatLong/articles/10768814.html>

线程同步是多同一把锁而言的。

修饰普通方法：锁的是调用该方法的对象，每个对象只有一把锁，不同的对象是不同的锁。

Public synchronized void method(){…………}

修饰静态方法：此时锁住的是类，类锁所有对象一把锁

Public static synchronized void run(){…………}

修饰代码块：锁住的是类

Public static void test(){

synchronized(A.class){

}

}

修饰类：锁住的是对象

Public void test1(){

synchronized(this){

}

}

补充：当没有明确的对象来作为锁的时候，只是想让一段代码同步时，可以创建一个特殊的对象类充当锁，如下：

class Test implements Runnable

{

private byte[] lock = new byte[0]; // 特殊的instance变量

public void method()

{

synchronized(lock) {

// todo 同步代码块

}

}

public void run() {

}

}

原理：

乐观锁：

　　乐观锁是一种乐观思想，即认为读多写少，遇到并发写的可能性低，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，采取在写时先读出当前版本号，然后加锁操作（比较跟上一次的版本号，如果一样则更新），如果失败则要重复读-比较-写的操作。java中的乐观锁基本都是通过CAS操作实现的，CAS是一种更新的原子操作，比较当前值跟传入值是否一样，一样则更新，否则失败。

悲观锁：

　　悲观锁是就是悲观思想，即认为写多，遇到并发写的可能性高，每次去拿数据的时候都认为别人会修改，所以每次在读写数据的时候都会上锁，这样别人想读写这个数据就会block直到拿到锁。java中的悲观锁就是Synchronized,AQS框架下的锁则是先尝试cas乐观锁去获取锁，获取不到，才会转换为悲观锁，如RetreenLock。

自旋锁（CAS）：

自旋锁就是让不满足条件的线程等待一段时间，而不是立即挂起。看持有锁的线程是否能够很快释放锁。怎么自旋呢？其实就是一段没有任何意义的循环。虽然它通过占用处理器的时间来避免线程切换带来的开销，但是如果持有锁的线程不能在很快释放锁，那么自旋的线程就会浪费处理器的资源，因为它不会做任何有意义的工作。所以，自旋等待的时间或者次数是有一个限度的，如果自旋超过了定义的时间仍然没有获取到锁，则该线程应该被挂起。JDK1.6中-XX:+UseSpinning开启； -XX:PreBlockSpin=10 为自旋次数； JDK1.7后，去掉此参数，由jvm控制；

1. **volatile的原理和作用**

<https://www.cnblogs.com/paddix/p/5428507.html>

为什么单例模式下singleton要用volatile修饰？

volatile是如何实现有序性和可见性的？

Volatile关键字的用法

1. **synchronized和volatile的区别**

<https://blog.csdn.net/xiaoming100001/article/details/79781680>

1. **Synchronized和lock的区别**

<https://blog.csdn.net/u012403290/article/details/64910926?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.channel_param>

| **类别** | **synchronized** | **Lock** |
| --- | --- | --- |
| 存在层次 | Java的关键字，在jvm层面上 | 是一个类 |
| 锁的释放 | 1、以获取锁的线程执行完同步代码，释放锁 2、线程执行发生异常，jvm会让线程释放锁 | 在finally中必须释放锁，不然容易造成线程死锁 |
| 锁的获取 | 假设A线程获得锁，B线程等待。如果A线程阻塞，B线程会一直等待 | 分情况而定，Lock有多个锁获取的方式，具体下面会说道，大致就是可以尝试获得锁，线程可以不用一直等待 |
| 锁状态 | 无法判断 | 可以判断 |
| 锁类型 | 可重入 不可中断 非公平 | 可重入 可判断 可公平（两者皆可） |
| 性能 | 少量同步 | 大量同步 |
| 原理 |  | CAS+AQS队列，通过一个state的值判断 |

Lock是显式锁，通过队列来实现。synchronized是隐式锁，通过CAS实现

1. **线程池**

<https://www.jianshu.com/p/7726c70cdc40>

<https://www.cnblogs.com/superfj/p/7544971.html> 棒

1. **多线程断点续传原理**

**RandomAccess** <https://blog.csdn.net/qq_40208605/article/details/80647803>

**59,同步工具类**

1：CountDownLatch用来协调多个线程之间的同步，它能够使一个线程在等待另外一个线程完成工作后继续执行。使用一个计数器实现的，计数器的初始值和线程数量一样，当一个线程完成自己的任务之后，计数器的值就会减1.计数器的值变为0的时候，等待的线程就可以恢复执行任务。

<https://www.cnblogs.com/Lee_xy_z/p/10470181.html>

2：CycliBarrier的作用是让所有线程都等待，直到都完成后再继续下一步执行。

比如去吃饭，有的朋友早到有的朋友晚到，那么就需要等所有人都到齐了才开吃。

<https://www.jianshu.com/p/333fd8faa56e>

**60,Activity四种启动模式**

**注意onNewIntent的使用**

<https://www.jianshu.com/p/ebde48a0a52c>

**61,Activity之间的通信方式**

Intent携带bundle数据、通过static变量、通过sharepreferrance、通过File或者通过SQLite

<https://blog.csdn.net/a_running_wolf/article/details/48813995>

1. **横竖屏切换的时候Activity的生命周期变化**

不同的Android系统版本表现不一样，不同的横竖屏设置表现也不一样

<https://blog.csdn.net/qq_36713816/article/details/80538467?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-3.channel_param>

1. **Activity和Fragment之间生命周期的比较**

<https://www.jianshu.com/p/9f538c3a1918> frgament的生命周期

1. **activity上有dialog时按下Home键时的生命周期**

非activity主题的dialog是不会影响activity的生命周期的，所以在这种情况下的生命周期是onPause--->onStop。

1. **activity状态保存与恢复**

<https://blog.csdn.net/sinat_33921105/article/details/78631823>

在这几种情况下需要对数据进行保存：点击了返回键、锁屏、点击home键、其他app进入前台比如电话、启动了新的activity、屏幕方向发生变化、app被杀死等。EditText是不需要特意做保存操作的，系统默认已经实现了。

在onSaveInstanceSate()中以Bundle数据包的形式保存数据，在onRestoreInstanceSate()或者onCraete()中从Bundle中读取数据进行恢复。

1. **fragment在各种情况下生命周期的变化**

<https://blog.csdn.net/ya1139569539/article/details/78192112>

1. **frgament的状态保存**

与activity类似

<https://blog.csdn.net/zephyr_g/article/details/53516568>

1. **如何绑定activity和Service**

<https://blog.csdn.net/liyuchong2537631/article/details/48440911>

<https://blog.csdn.net/weixin_41101173/article/details/79718718?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.channel_param>

1. **Service和activity如何通信？**

<https://www.jianshu.com/p/4b0cbe068555>

<https://www.cnblogs.com/jiangzhaowei/p/11341890.html>

<https://www.jb51.net/article/137607.htm>

第一种：通过MyBinder方式调用Service的方法

第二种：通过接口

第三种：广播、handler

1. **Service的启动方式**

<https://www.jianshu.com/p/239a866cf85d>

1. **broadcastReceiver广播**

<https://www.jianshu.com/p/7b44d983c01a>

1. **ContentProvider**

<https://www.jianshu.com/p/5e13d1fec9c9>

1. **AlertDialog,popupWindow和activity的区别**

第一种非阻塞，弹出后背景是灰色的，后台程序依然在执行；

第二种是阻塞式的，弹出后没有背景，后台停止运行，直到使用dismiss

第三种是伪activity弹框，适用于比较负责的弹框，其实就是一个activity修改它的theme 主题而已。

<https://blog.csdn.net/wenxustqiang/article/details/79758165>

<https://www.cnblogs.com/liangstudyhome/p/3724590.html>

<https://www.cnblogs.com/wuyou/p/3581836.html> activity伪弹框

1. **Application 和 Activity 的 Context 对象的区别**

<https://blog.csdn.net/zx_android/article/details/79483748>

1. **Android属性动画特性**

**76，LinearLayout、RelativeLayout、FrameLayout的特性及对比，并介绍使用场景。**

1：RelativeLayout和LinearLayout的layout和draw过程相差无几，但是measure的时候，RelativeLayout和LinearLayout慢一大截。

2：FrameLayout和LinearLayout基本上性能一样。

1.RelativeLayout会让子View调用2次onMeasure，LinearLayout 在有weight时，也会调用子View2次onMeasure

2.RelativeLayout的子View如果高度和RelativeLayout不同，则会引发效率问题，当子View很复杂时，这个问题会更加严重。如果可以，尽量使用padding代替margin。

3.在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。

3.在不影响层级深度的情况下,使用LinearLayout和FrameLayout而不是RelativeLayout。

最后再思考一下文章开头那个矛盾的问题，为什么Google给开发者默认新建了个RelativeLayout，而自己却在DecorView中用了个LinearLayout。因为DecorView的层级深度是已知而且固定的，上面一个标题栏，下面一个内容栏。采用RelativeLayout并不会降低层级深度，所以此时在根节点上用LinearLayout是效率最高的。而之所以给开发者默认新建了个RelativeLayout是希望开发者能采用尽量少的View层级来表达布局以实现性能最优，因为复杂的View嵌套对性能的影响会更大一些。

1. 能用两层LinearLayout，尽量用一个RelativeLayout，在时间上此时RelativeLayout耗时更小。另外LinearLayout慎用layout\_weight,也将会增加一倍耗时操作。由于使用LinearLayout的layout\_weight,大多数时间是不一样的，这会降低测量的速度。这只是一个如何合理使用Layout的案例，必要的时候，你要小心考虑是否用layout weight。总之减少层级结构，才是王道，让onMeasure做延迟加载，用viewStub，include等一些技巧。
2. **接口回调的原理**

回调方类A的一个方法a通过调用类B的一个方法b去调用类A的另一个方法c。

<https://blog.csdn.net/IT_XF/article/details/65058367?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.channel_param>

<https://blog.csdn.net/xiaanming/article/details/8703708>

1. **写一个接口回调的demo**
2. **Android常用的多线程**

1：继承Thread

2：实现Runable

3：Handler

4：AsyncTask

<https://www.jianshu.com/p/ee1342fcf5e7>

5：HandlerThread

<https://www.jianshu.com/p/9c10beaa1c95>

Handler内存泄漏原因以及解决办法

<https://www.jianshu.com/p/ed9e15eff47a>

Android切换UI线程的方法有哪些？