***Android的消息机制：***

Android中消息机制主要指Handler的运行机制，Handler的运行需要底层的MessageQueue和Looper的支撑，MessageQueue是消息列队，内部存储一组消息，以队列的形式对外提供插入和删除的工作，是采用单链表的数据结构来存储消息列表的。Looper消息列队，ThreadLocal作用是在每个线程中存储数据，在不同的线程中互不干扰的存储并提供数据，通过ThreadLocal可以轻松获取每个线程的looper。

主线程（ActivityThead）被创建时就会初始化looper（Looper.prepareMainLooper();）也就是主线程中默认可以使用Handler的原因。

Android系统为什么不允许在子线程中访问ui呢？因为Android的UI控件不是线程安全的，如果在多线程中并发访问可能会导致UI控件处于不可预期的状态。

为什么不给枷锁呢？

加锁机制会让ui访问的逻辑变得更加复杂，其次减低ui的访问效率，因为锁机制会阻塞默写线程的执行，最简单的方式就是采用单线程模型处理ui操作（通过handler切换一下就好）

ThreadLocal：一个线程内部的数据存储类，通过它可以在指定的线程中存储数据，在指定线程中可以获取到存储的数据，对于其他线程来说无法获取到的数据。

ThreadLocal另一个使用场景是复杂逻辑下的对象传递，比如监听器的传递，线程中任务过于复杂，使用ThreadLocal可以让监听器作为线程内的全局对象而存在，在线程内部只有通过get方法可以获取到监听器，如果不使用ThreadLocal,想到两种方法，第一种就是将监听器通过参数的方式在函数调用栈中进行传递，第二种寄宿将监听器作为静态变量供线程访问。

ThreadLocal的set和get方法可以得出，他们所有的操作的对象都是当前线程的localValues对象的table数组，因此在不同线程中访问同一个ThreadLocal的set和get方法，他们对threadlocal所做的读写操作仅限于各自线程内部。

如何为一个线程创建一个Looper？

通过Looper.prepare()为当前线程创建一个Looper，通过Looper.loop()开启消息循环。

Looper.prepare();

Handler handler = new Handler();

Looper.loop();

在子线程中创建Handler为什么抛出异常：

因为Handler对象是基于Looper的，每个Handler必须有一个looper,这个Looper是在调用Looper.prepare()的时候创建的，Looper会以Map的形式存储在当前线程的ThreadLocal中，当调用Looper.myLooper（）方法就是去当前线程的ThreadLocal中拿到当前线程的Looper线程，没有拿到就会抛出异常。

如何直接获取UI线程中的Handler?

Handler handler = new Handler(Looper.getMainLooper());

Android 线程和线程池：

**Android线程保活：**现进程的保活，暂时实现了双进程守护，JobService的检测与拉起，

进程死亡AlarmManager定时拉起，广播监听（网络变化，开机等），同时通过Timer和TimerTask实现心跳服务：1.0双进程守护：双进程守护即本地进程和远程进程，两个类：AbsHeartBeanService：本地进程，抽象的心跳服务：DaemonService:远程进程，即守护进程：启动本地服务后会启动远程进程的服务并绑定远程的进程服务，同时远程服务也会绑定本地进程的服务，任何一个服务停止都会得到另一个进程binder通知，即刻被拉起，实现进程保活的方式。

2.0 JobService检测与拉起：JobService做定时任务，定时检测本地进程的服务是否运行。

3.0 进程死亡AlarmManager定时拉起

4.0 广播监听：

动态和静态：

动态：网络变化，开屏，锁屏，点击home键。

静态： 开机，连接电源，断开电源，安装应用，卸载等~

**Android 框架原理（框架解决的问题，框架大体框架结构，核心点的实现）:**

**设计模式：如何理解设计模式的基本原则（单一原则，开闭原则等），平时用过哪几种，列举例子，解决了什么问题。**

****

优点：高内聚，低耦合。

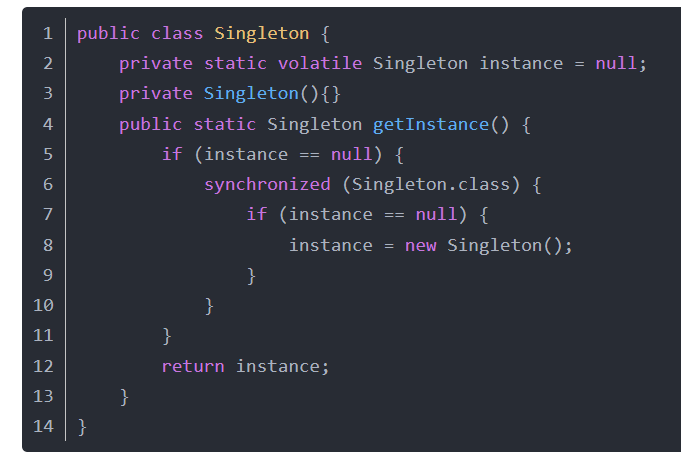
使用案例：创建型的模式

比如单例和建造者模式：

单例模式：一个类只有一个实例，并且唯一的实例。

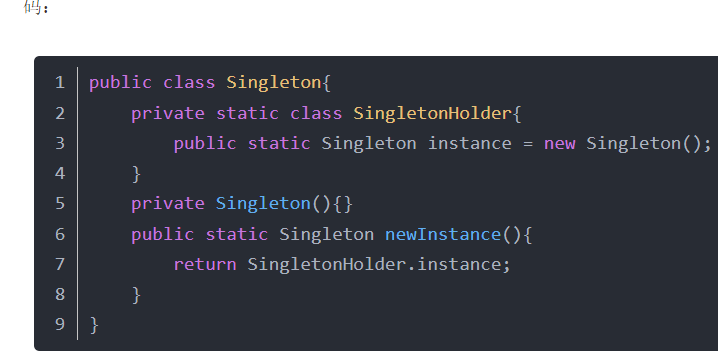
优势：避免实例对象的重复创建，减少每次创建对象的时间开销，节约内存空间，避免由于过多操作多个实例导致的逻辑错误。

书写：



**双重校验锁**

**静态内部类**

****

**设计模式https://mp.weixin.qq.com/s/T6ZCQRydzFgVUezlywC8Zw**

Android中的使用实例

比如Application,和Gson,网络请求的相关管理类（Retrofit对象，ServiceManager等），Android提供的各种Manager( NotiificationManager).

Builder的链式调用：（建造者模式）：

使用场景：

当面临一个复杂对象的创建工作，通常由各个部分的子对象用一定的算法构成，由于需求的变化，这个复杂对象的各个部分经常面临剧烈的变化，但是将他们组合在一起的算法相对稳定。

Android的实例：AlterDialog的创建使用：链式调用

new AlterDialog

.Buillder(this)

.setTitle(“标题”)

.setMessage(“内容”)

.setNegativeButton(“取消”，new DialogInterce.OnClickListener(){

public void onClick(){

}

}

Java中的建造者模式：StringBuilder和StringBuffer的源码中append（方法

）



单一原则：一个类只负责一个功能领域中相应的职责。

开闭原则：一个软件应对扩展开发，对修改关闭。

Adapter模式：使原本由于接口不兼容而不能一起工作的那些类可以一起工作。

RecyclerView：适配器模式（recyclerView.Adapter）

Retrofit：外观模式与动态代理：

迪米特法则：最少知识原则：一个软件实体应当尽可能少的与其他实体发生作用。

Public interface BlogService{

}

Android适配相关内容（总结）

<https://www.jianshu.com/p/55e0fca23b4f>

https://github.com/JessYanCoding/AndroidAutoSize