Android面经

Android的四大组件

- Activity： 用户与程序进行交互的界面
- Service：适合不需要和用户交互，后台长时间耗时的服务
- Broadcast Receiver： 接收一个或多个Intent作触发事件
- Content Provider： 提供的第三方应用程序的访问方案

常用的布局

- FrameLayout
- LinearLayout
- RelativeLayout
- AbsoluteLayout
- TableLayout

Activity的生命周期

> onCreate()、onStart()、onReStart()、onResume()、onPause()、onStop()、onDestory()；

- 可见生命周期：从onStart()直到系统调用onStop()
- 前台生命周期：从onResume()直到系统调用onPause()

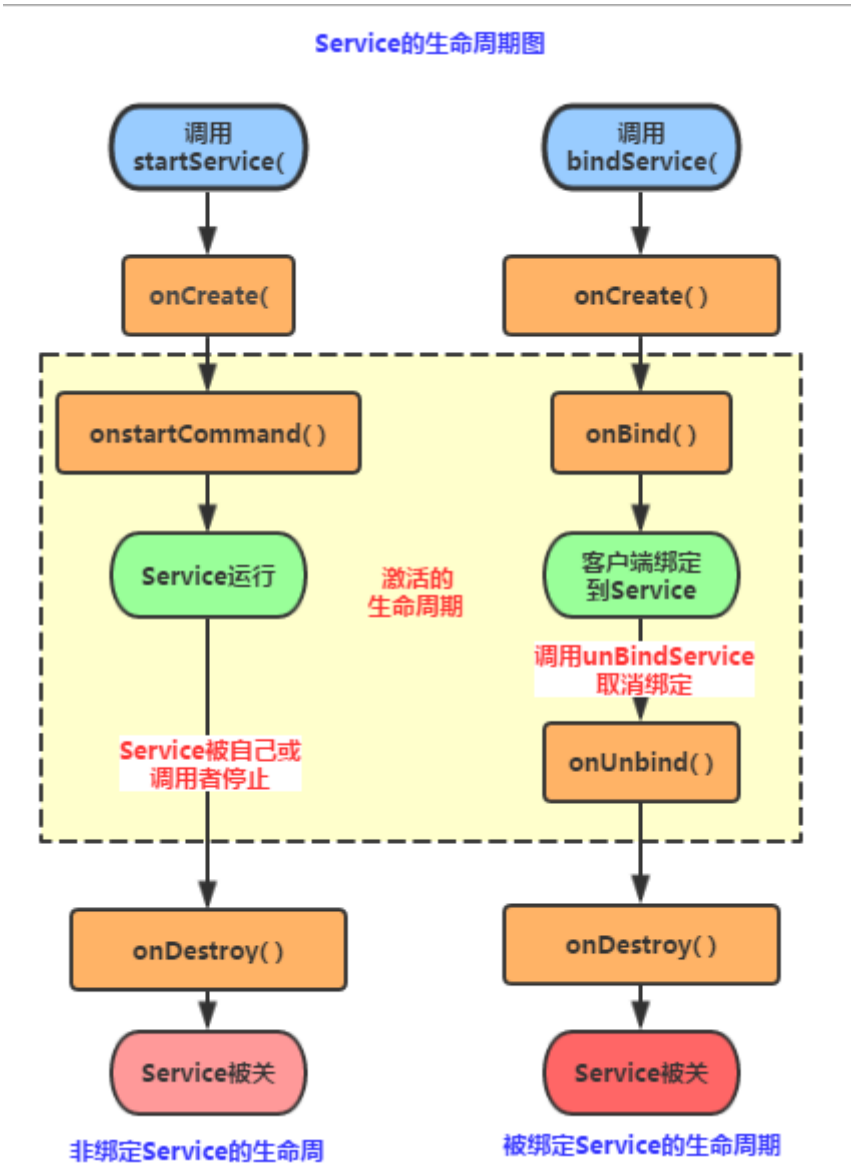
如何在Android中执行耗时操作

> 将耗时操作放到非UI线程执行，常用的包括AsyncTask、Handler配合Thread等

activity在屏幕旋转时的生命周期

> 不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次；设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次；设置Activity的android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

Service的生命周期图



Service和IntentService的区别

- Service:默认运行在主线程，不可直接执行耗时操作，不会自动退出
- IntentService：默认运行在子线程，可以直接执行耗时操作，可以自动退出

Activity、Service、app间的通信方式

Activity间

- 基于消息的通讯机制：Intent。缺点：只能传递基本数据类型或序列化数据
- Broadcast广播
- EventBus
- 数据存储（SharePreference、SQLite、ContentProvider)
- 静态变量

Activity与Service之间

- Binder对象
- Broadcast广播
- Intent
- EventBus

Fragment之间

- 拿到另一个Fragment实例
- 通过Actively来操作
- 广播

两个app之间

- Activity（Intent）
- Broadcast
- ContentProvider
- Service跨进程通信
- 利用ShareUserId共享数据，两个ShareUserId相同，则共享对方的`data`目录下的文件

使用Service的方式

> AndroidManifest.xml完成Service注册

- StartService()启动Service
  - 首次启动会创建一个Service实例,依次调用onCreate()和onStartCommand()方法,此时Service进入运行状态,如果再次调用StartService启动Service,将不会再创建新的Service对象, 系统会直接复用前面创建的Service对象,调用它的onStartCommand()方法!
  - 但这样的Service与它的调用者无必然的联系,就是说当调用者结束了自己的生命周期, 但是只要不调用stopService,那么Service还是会继续运行的!
  - 无论启动了多少次Service,只需调用一次StopService即可停掉Service
- BindService()启动Service
- 启动Service后绑定Service
  - 如果Service已经由某个客户端通过StartService()启动,接下来由其他客户端 再调用bindService() 绑定到该Service后调用unbindService()解除绑定最后在 调用bindService()绑定到Service的话,此时所触发的生命周期方法如下:onCreate( )->onStartCommand( )->onBind( )->onUnbind( )->onRebind( )
  - 使用bindService来绑定一个启动的Service,注意是已经启动的Service!!! 系统只是将Service的内部IBinder对象传递给Activity,并不会将Service的生命周期 与Activity绑定,因此调用unBindService( )方法取消绑定时,Service也不会被销毁!

Fragment

什么是Fragment

- 一种可以嵌入在Activity中的UI片段
- 可以将多个Fragment组合，Activity可以复用

Fragment的生命周期

- Fragment 的生命周期与 Activity 的生命周期协调一致
- Activity 的每次生命周期回调都会引发每个片段的类似回调
- 片段还有额外的生命周期，用于处理与Activity的唯一交互
  - onAttach: 与Activity关联时调用
  - onCreatView：创建Fragment的视图层次结构
  - onActivityCreate： Activity的onCreate方法已返回时调用
  - onDestoryView：移除与Fragment片段关联的视图层次结构时调用
  - onDetach：取消与Activity关联时调用

View

> UI组件的基本模块，一个View占据屏幕一部分，负责绘制和事件处理

Activity、View和Window的关系

- 每个Activity包含了唯一的一个PhoneWindow，实现了Window接口
- View是Actively中的视图单元，必须依附于Window存在
- View和Window之间通过ViewRootImol进行关联，ViewRootImpl 可以看作 ViewTree 的管理者

ViewGroup

- 一个放置View的容器，继承自View，是一种特殊的View
- 功能：为childView计算出建议的宽、高、测量模式；决定hildView的位置
- View的功能：根据测量模式和 ViewGroup 建议的宽高计算出自己的宽和高；在 ViewGroup 为其指定的区域内绘制自己的形态

View.GONE 和 View.INVISIBLE 之间的区别。

- View.GONE：控件不可见，在布局中不占据空间
- View.INVISIBLE：控件不可见，但在布局中还占据空间

Intent

- 一个消息传递对象，可以启动Activity、Service、Broadcast，同时携带必要的数据
- 分为显示和隐式Intent
  - 显示按照名称指定要启动的组件
  - 隐式Intent则声明要执行的操作，允许其他应用组件来处理
- 基于Binder机制，Intent可通过BUndle携带可序列化的对象

AsyncTask

> 一个轻量级的用于处理异步任务的类:AsyncTask

- 有四个回调方法,不能在UI线程中手动调用
  - `onPreExecute`：在执行后台之前调用，运行在主线程
  - `doInBackground`：必须实现的核心方法，可以执行后台耗时操作，运行在子线程
  - `onProgressUpdate` 在 doInBackground 方法中调用publishProgress 方法时会进行回调，可以进行后台操作状态的展示更新，运行在主线程
  - `onPostExecute` 在后台操作完成后调用，运行在主线程

AsyncTAsk的原理

- 首先调用 AsyncTask 的构造方法，构造时对 Handler、WorkerRunnabl（Callable） 和 FutureTask 进行初始化
- 然后调用 AsyncTask 的 execute 方法（可以手动设置 Executor，不设置则使用系统默认的 SerialExecutor）
- 首先判断当前 AsyncTask 状态，正在运行或者已经运行过就退出
- 调用 onPreExecute 执行准备工作
- 由 Executor 调用 FutureTask 的 run 方法，在 WorkerRunnable 中执行了 doInBackground
- 依旧是在 WorkerRunnable 中，调用 postResult，将执行结果通过 Handler 发送给主线程；调用 publishProgress 时，也是通过 Handler 将消息发送到主线程的消息队列中

生命周期

- 不随Activity的销毁而销毁，一直执行doInBackGround直到结束

导致的问题

- 使用非静态内部AsyncTask时，持有外部类Activity的引用，造成Activity结束时对象无法回收，内存泄漏

解决

- 如果 doInBackground 方法内有循环操作时，应使用 isCancelled 来进行判断，避免后续无用的循环操作
- 销毁 Activity 实例时，调用 AsyncTask 的 cancel 方法
- 使用静态内部类，持有外部类的弱引用

Support Libary

> 安卓支持库提供了许多没有内置到框架中的功能

- 向下兼容，保证高版本SDK开发的向下兼容
- 提供了布局模式、界面元素，可以避免再做重复的工作
- 支持不同形态的设备

Android的虚拟机

> Dalvik虚拟机是Google设计用于Android平台的虚拟机
> 有自己的字节码，并不使用Java的字节码
> 从2.2开始支持即时编译

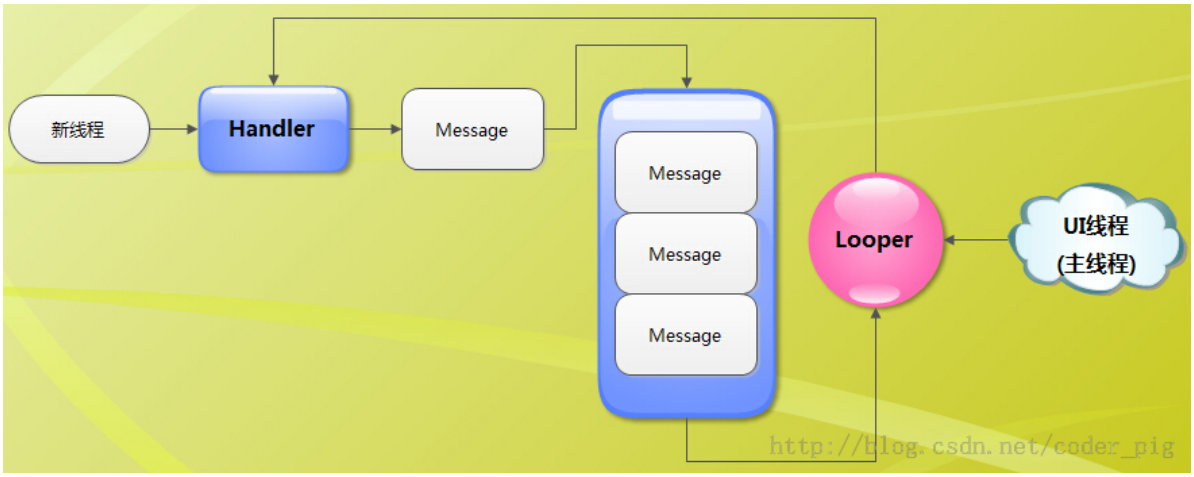
ANR是什么？怎么避免？

> Application Not Responding.

- 活动管理器和窗口管理器这两个系统服务负责监视应用程序的响应，当用户操作的在5s内应用程序没能做出反应，BroadcastReceiver在10秒内没有执行完毕，就会出现应用程序无响应对话框，这既是ANR。
- 避免方法：Activity应该在它的关键生命周期方法（如onCreate()和onResume()）里尽可能少的去做创建操作。潜在的耗时操作，例如网络或数据库操作，或者高耗时的计算如改变位图尺寸，应该在子线程里（或者异步方式）来完成。主线程应该为子线程提供一个Handler，以便完成时能够提交给主线程。

Handler

> Android为了线程安全，并不允许我们在UI线程外操作UI；很多时候我们做界面刷新都需要通过Handler来通知UI组件更新
> Handler的执行流程图



主线程

```
final Handler myHandler = new Handler()
 {
   @Override
   //重写handleMessage方法,根据msg中what的值判断是否执行后续操作
   public void handleMessage(Message msg) {
     if(msg.what == 0x123)
        {
         imgchange.setImageResource(imgids[imgstart++ % 8]);
        }
     }
 };

 myHandler.sendEmptyMessage(0x123);
```

子线程

```
Looper.prepare();
//实现Handler
Looper.loop();
```

post与sendMassage的区别

> handler.post和handler.sendMessage本质上是没有区别的，都是发送一个消息到消息队列中，而且消息队列和handler都是依赖于同一个线程的。

```
/*post方法解决UI更新，直接在runnable里面完成更新操作，这个任务会被添加到handler所在线程的消息队列中，即主线程的消息队列中*/
handler_post.post(new Runnable() {
          @Override
          public void run() {
            tv_up.setText(new_str);
          }
        });
```

HandlerThread和Thread的区别

> HandlerThread继承于Thread，所以它本质就是个Thread。与普通Thread的差别就在于，然后在内部直接实现了Looper的实现，这是Handler消息机制必不可少的。有了自己的looper,可以让我们在自己的线程中分发和处理消息。如果不用HandlerThread的话，需要手动去调用Looper.prepare()和Looper.loop()这些方法。

Activity 的四种启动模式对比

- standard:默认的启动模式，每启动一个Activity就会在栈顶创建一个新的实例
- singleTop：该模式会判断要启动的Activity实例是否位于栈顶，如果位于栈顶直接复用，否则创建新的实例。如果Activity并未处于栈顶位置，则可能还会创建多个实例
- singleTask：使Activity在整个应用程序中只有一个实例。每次启动Activity时系统首先检查栈中是否存在当前Activity实例，如果存在则直接复用，并把当前Activity之上所有实例全部出栈
- singleInstance：在整个系统里只有一个实例

onNewIntent()的调用时机

> 在该Activity的实例已经存在于Task和Back stack中(或者通俗的说可以通过按返回键返回到该Activity)时,当使用intent来再次启动该Activity的时候,如果此次启动不创建该Activity的新实例,则系统会调用原有实例的onNewIntent()方法来处理此intent.

广播的两种启动方式

动态注册

- Context.registerReceiver()方法来注册；
- 自定义一个BroadcastReceiver，在onReceive()方法中完成广播要处理的事务
- 指定IntentFilter，添加不同的Action即可，一定要调用unregisterReceiver让广播取消注册

静态注册

- AndroidManifest.xml中通过

Worker

WorkManager

如何做应用适配

- 使用 wrap_content 和 match_parent 而非硬编码尺寸，以适应各种屏幕尺寸和方向
- 使用相对布局，禁用绝对布局
- 使用限定符（尺寸限定符、最小宽度限定符、屏幕方向限定符）
- 使用点九图
- 使用密度无关像素单位
- 图片文件放置到合适的 drawable 目录下

px、dp、sp、ppi、dpi 有什么区别，如何换算，给出公式

- px 为像素点数
- ppi 与 dpi 均代表像素密度，即每英寸上的像素点数
- dp 为像素无关密度，以 160 ppi 为基准，1dp = 1px
- sp 与 dp 类似，用于描述字体大小，以 160 ppi 为基准，当字体大小为 100% 时，1sp = 1px

ListView和RecyclerView

效率上

- RecyclerView更高
- RecyclerView的ViewHolder模式更加规范

功能

- 新增LayoutManager，新增瀑布、网格、横向布局
- RecyclerView 高度解耦，LayoutManager 负责布局显示，ItemDecoration 负责 item 分割线，ItemAnimator 负责 item 增删动画。

周期地更新页面的最好方式是什么?

- 使用 Alarm 机制
- 使用定时器 Timer 类

Handler

定义

> 用于线程间通信，主要负责Android消息机制中消息的发送和接收

发送

> 通过向消息队列插入一条消息实现发送，
> 使用方法：发送消息可以通过 Handler 对象的 sendMessage 的一系列方法和 post 一系列方法来实现。

MVC,MVP和MVVM

MVC-Model，View，Controller

- View：XML布局文件。 Model：实体模型（数据的获取、存储、数据状态变化）。 Controller：对应于Activity，处理数据、业务和UI。
- 但是Android中纯粹作为View的XML视图功能太弱，我们大量处理View的逻辑只能写在Activity中，这样Activity就充当了View和Controller两个角色，直接导致Activity中的代码大爆炸

MVP-Model，View，Prsenter

- **View:** 对应于Activity和XML，负责View的绘制以及与用户的交互。 **Model:** 依然是实体模型。 **Presenter:** 负责完成View与Model间的交互和业务逻辑。
- Persenter持有该View接口，对该接口进行操作，而不是直接操作View层。这样就可以把视图操作和业务逻辑解耦，从而让Activity成为真正的View层。

MVVM-Model，View，ViewModel

- **View:** 对应于Activity和XML，负责View的绘制以及与用户交互。 **Model:** 实体模型。 **ViewModel:** 负责完成View与Model间的交互，负责业务逻辑。
- 利用数据绑定(Data Binding)、依赖属性(Dependency Property)、命令(Command)、路由事件(Routed Event)等新特性，打造了一个更加灵活高效的架构。

View事件分发机制

MotionEvent事件

- ACTION_DOWN：手指刚接触屏幕，按下去的那一瞬间产生该事件
- ACTION_MOVE：手指在屏幕上移动时候产生该事件
- ACTION_UP：手指从屏幕上松开的瞬间产生该事件

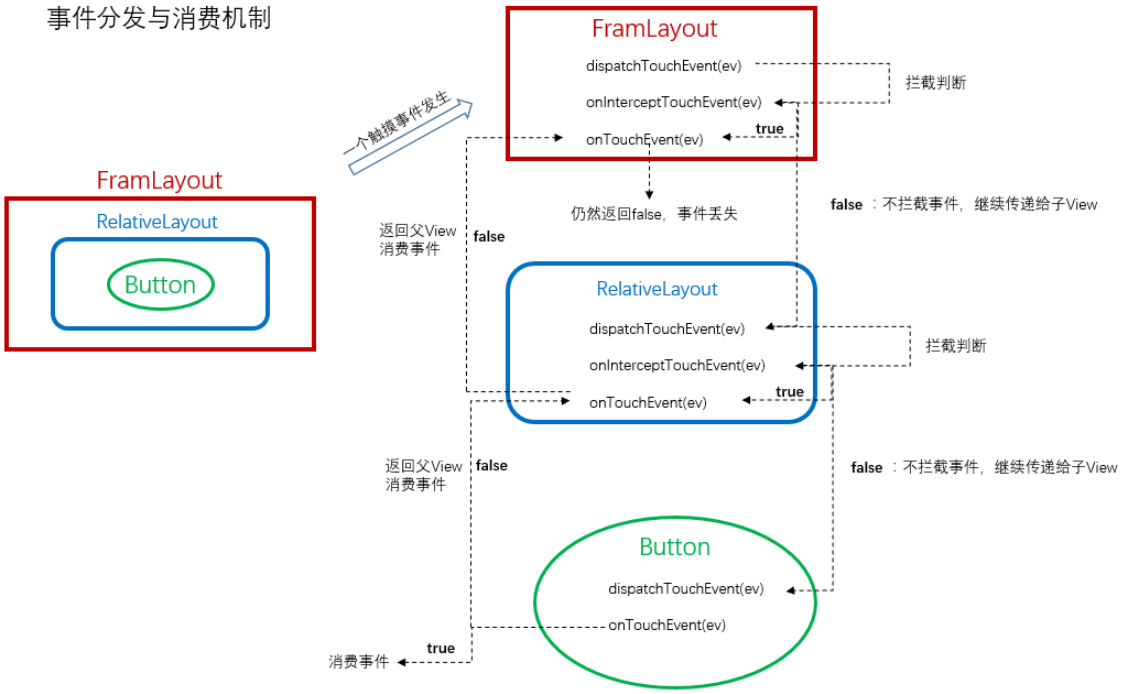
> 从ACTION_DOWN开始到ACTION_UP结束我们称为一个事件序列

事件分发

> 当一个MotionEvent产生了以后，就是你的手指在屏幕上做一系列动作的时候，系统需要把这一系列的MotionEvent分发给一个具体的View。通过三个重要方法完成

- `public boolean dispatchTouchEvent(MotionEvent event)`：事件的分发就是当一个触摸事件发生的时候，会按照Activity -> Window -> View的顺序依次往下传递。也就是说系统会把这个事件传递给一个具体的View，从而来执行或者说响应这个事件。返回值：表示是否消费了当前事件。可能是View本身的onTouchEvent方法消费，也可能是子View的dispatchTouchEvent方法中消费。返回true表示事件被消费，本次的事件终止。返回false表示View以及子View均没有消费事件，将调用父View的onTouchEvent方法
- `public boolean onInterceptTouchEvent(MotionEvent ev)`：ViewGroup特有的方法，View并没有拦截方法
  返回值：是否拦截事件传递，返回true表示拦截了事件，那么事件将不再向下分发而是调用View本身的onTouchEvent方法。返回false表示不做拦截，事件将向下分发到子View的dispatchTouchEvent方法。
- `public boolean onTouchEvent(MotionEvent ev)`：真正对MotionEvent进行处理或者说消费（调用点击事件）的方法。在dispatchTouchEvent进行调用。
  返回值：返回true表示事件被消费，本次的事件终止。返回false表示事件没有被消费，将调用父View的onTouchEvent方法

事件分发与消费机制



```
    public boolean dispatchTouchEvent(MotionEvent ev) {
        boolean consume = false;//事件是否被消费
        if (onInterceptTouchEvent(ev)){//调用onInterceptTouchEvent判断是否拦截事件
            consume = onTouchEvent(ev);//如果拦截则调用自身的onTouchEvent方法
        }else{
            consume = child.dispatchTouchEvent(ev);//不拦截调用子View的
dispatchTouchEvent方法
        }
        return consume;//返回值表示事件是否被消费，true事件终止，false调用父View的
onTouchEvent方法
    }
```

okHttp3

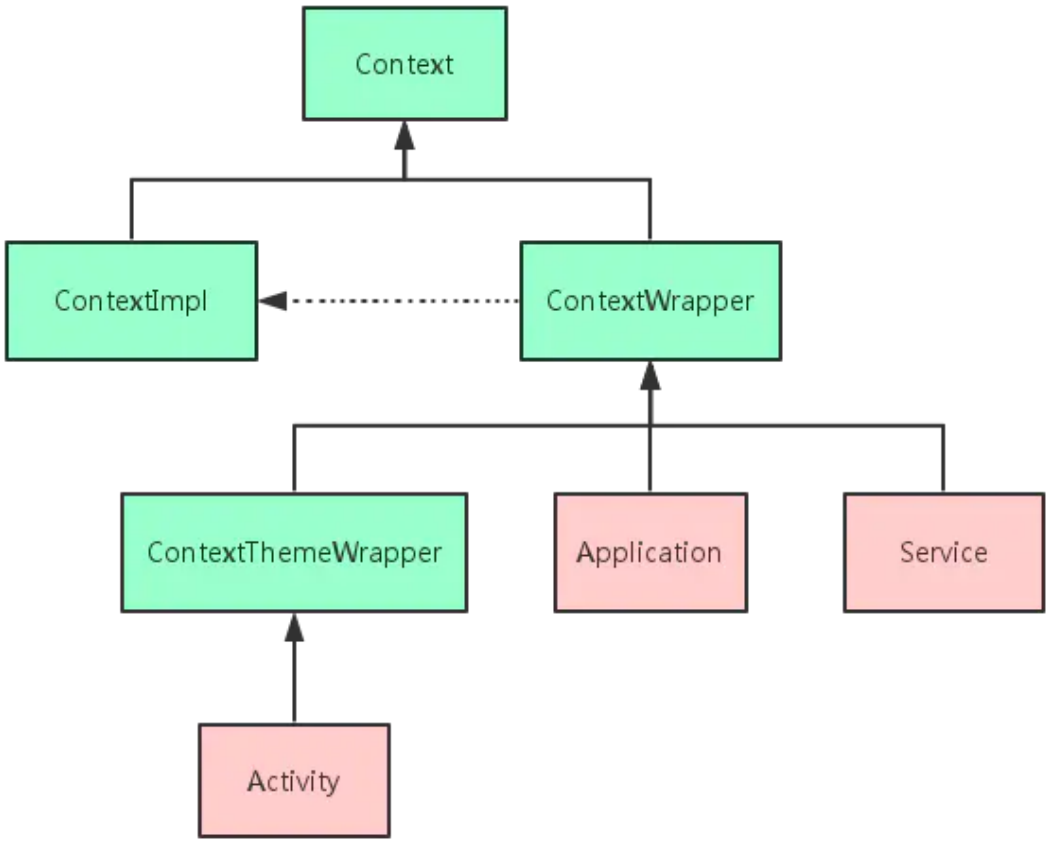
简介

- 一个非常高效的Http客户端，近年来几乎所有的Android应用都会使用它作为网络访问的框架
- 使用构造器模式builders来设计，它支持阻塞式的同步请求和带回调的异步请求

优点

- okhttp支持SPDY，允许所有的访问统一主机的请求共享一个socket
- 利用连接池减少请求延迟
- 支持GZIP压缩
- 响应缓存减少重复的请求

Context



> 与有关应用程序环境的全局信息的接口。 这是一个抽象类，其实现由Android系统提供。 它允许访问特定于应用程序的资源和类，以及向上调用应用程序级操作，例如启动活动、广播和接收意图等。

EventBus

> 一种用于Android的事件发布-订阅机制，简化了各个组件之间通信的复杂度，避免由于使用广播通信带来的不便

四种线程模型

- Posting：默认的，事件处理函数的线程跟发布的在同一个线程
- MAIN：
- BackGround：
- ASYNC

发布订阅流程

- 创建一个事件类型，可以是int，可以是String，可以是自定义的
- 在需要订阅的模块中，注册eventBus

```
    @Override
    protected void onStart() {
        super.onStart();
        EventBus.getDefault().register(this);
    }

    @Override
    protected void onStop() {
        super.onStop();

    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        EventBus.getDefault().unregister(this);
    }
```

- 注册后创建一个方法来接受消息

```
@Subscribe(threadMode = ThreadMode.MAIN)
    public void onReceiveMsg(EventMessage message) {
        Log.e(TAG, "onReceiveMsg: " + message.toString());
    }
```

- 在需要发送事件的地方调用post发送

```
    EventMessage msg = new EventMessage(1,"Hello MainActivity");
    EventBus.getDefault().post(msg);
```

使用SQLite

- 创建xxHelper类，继承SQLiteOpenHelper
  - `onCreat()`创建数据库，第一次建表时使用
  - `onUpgrade()`更新数据库表结构，数据库版本发生变化的时候回调，必须传入一个version参数
- 创建xxOperator
  - 获得之前的xxHelper的实例， SQLiteDatabase db = xxHelper.getWritableDatabase()/getReadableDatabase();
  - 更新可以用contentValues

Maven 和 Gradle 的区别

> Gradle 的优势：依赖管理、多模块构建、Maven 基于 XML 配置繁琐，阅读性差，Gradle 基于 Groovy，简化了构建代码的行数，易于阅读

- 依赖管理方面：Gradle 支持依赖动态版本管理，解决依赖冲突机制更明确
- 多模块构建方面：Gradle 使用 allprojects 和 subprojects 来定义里面的配置是应用于所有项目还是子项目，更加灵活
- 构建周期方面：Gradle 本身与项目构建周期是解耦的，可以灵活的增删 task