# 【安卓 R 源码】从源码角度看Context

原创 蜘蛛侠不会飞 于 2021-11-22 16:25:35 发布 阅读量2.1k 收藏 2 点赞数 版权

分类专栏： 安卓源码解析 安卓开发 文章标签： 安卓 安卓源码 Context 源码

安卓源码解析 同时被 2 个专栏收录 5 订阅 22 篇文章 订阅专栏

目录



## 1. Context 是什么?

Context，中文直译为“上下文”，它描述的是一个应用程序环境的信息，从程序的角度上来理解：Context是个抽象类，而Activity、Service、Application等都是该类的一个实现。SDK中对其说明如下：

> Interface to global information about an application environment. This is an abstract class whose implementation is provided by the Android system. It allows access to application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc

中文意思为：

> 1、它描述的是一个应用程序环境的信息，即上下文。
> 2、该类是一个抽象( abstract class)类，android提供了该抽象类的具体实现类(ContextIml类)。
> 3、通过它我们可以获取应用程序的资源和类，也包括一些应用级别操作，例如：启动一个Activity，发送广播，接受Intent信息 等，有如下功能：
> - 启动Activity
> - 启动和停止Service
> - 发送广播消息(Intent)
> - 注册广播消息(Intent)接收者
> - 可以访问APK中各种资源(如Resources和AssetManager等)
> - 可以访问Package的相关信息
> - APK的各种权限管理

在android中context可以作很多操作，但是最主要的功能是加载和访问资源。在android中有两种context，一种是 application context，一种是activity context，通常我们在各种类和方法间传递的是activity context。

有些函数调用时需要一个Context参数，比如Toast.makeText，**因为函数需要知道是在哪个界面中显示的Toast。**一般在Activity中我们直接用this代替，代表调用者的实例为Activity。再比如，Button myButton = new Button(this); **这里也需要Context参数（this），表示这个按钮是在“this”这个屏幕中显示的。**

Context体现到代码上来说，是个抽象类，其实调用的是 ContextImpl，其主要方法如下：

| Context行为分类 | 常用函数 |
| --- | --- |
| 使用系统提供的服务 | getPackageManager()、getSystemService()… |
| 基本功能 | startActivity()、sendBroadcast()、registerReceiver()、startService()、bindService()、getContentResolver()…【内部基本上是和AMS打交道】 |
| 访问资源 | getAssets()、getResources()、getString()、getColor()、getClassLoader()… |
| 和当前所寄身的进程之间的联系 | getMainLooper()、getApplicationContext()… |
| 和信息存储相关 | getSharedPreferences()、openFileInput()、openFileOutput()、deleteFile()、openOrCreateDatabase()、deleteDatabase()… |

CSDN @蜘蛛侠不会飞

常用的一些方法：

```
TextView tv = new TextView(getContext());

ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), …);

AudioManager am = (AudioManager) getContext().
                      getSystemService(Context.AUDIO_SERVICE);
getApplicationContext().getSharedPreferences(name, mode);
```

```
8
   9  getApplicationContext().getContentResolver().query(uri, …);
10
11 getContext().getResources().getDisplayMetrics().widthPixels * 5 / 8;
12
13 getContext().startActivity(intent);
14
15 getContext().startService(intent);
16
17 getContext().sendBroadcast(intent);
```

## 2. 获取 Context 的 3 种方式

- **1. mContext = getApplicationContext();**

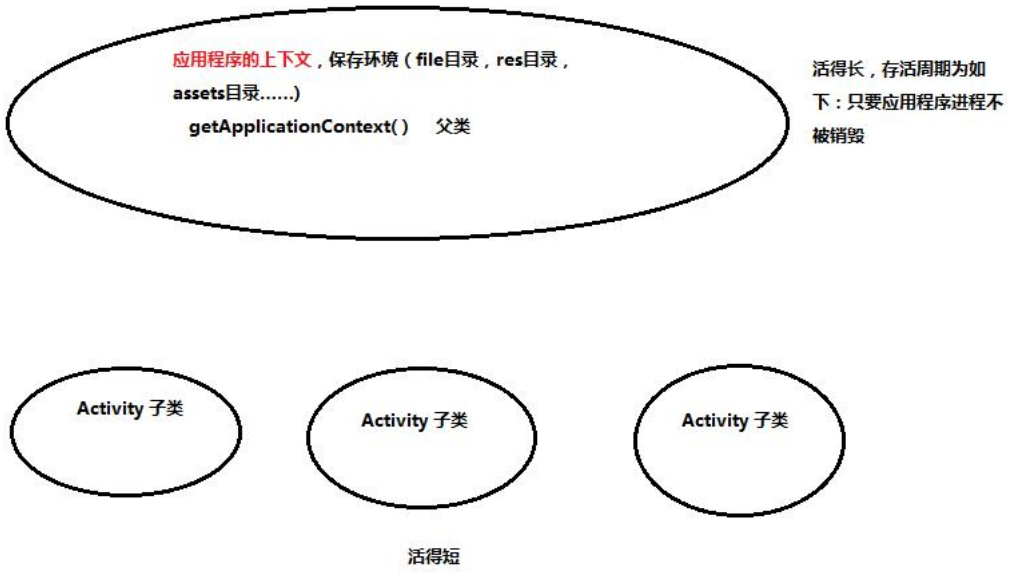
这种方式获得的context是全局context，整个项目的生命中期中是唯一的且一直存在的，代表了所有activities的context

- **2. mContext = getContext()**

这种方式获得的context当activity销毁时，context也会跟着销毁了

- **3. mContext = getBaseContext();**

获取的是 ContextImpl




应用程序创建Context实例的情况有如下几种情况：

> 1、创建Application 对象时， 而且**整个App共一个Application对象**
> 2、创建Service对象时
> 3、创建Activity对象时

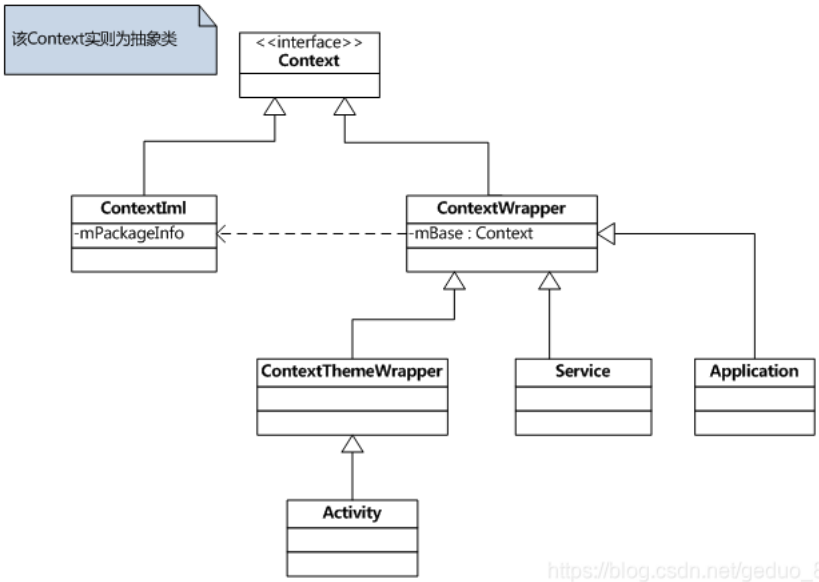
因此应用程序App共有的Context数目公式为：**总Context实例个数 = Service个数 + Activity个数 + 1（Application对应的Context实例）**
Broadcast Receiver，Content Provider并不是Context的子类，他们所持有的Context都是其他地方传过去的，所以并不计入Context总数。

## 3. Context 及其相关类源码

Context相关类的继承关系如下：



C 笔记






Context的直接子类是ContextImpl和ContextWrapper（后文分析二者的作用），Service和Application二者相似，都是继承于ContextWrapper，而Activity是继承于ContextThemeWrapper，因为Activity是有主题的

## 1）. Context相关的类

- **frameworks/base/core/java/android/content/Context.java**

```
// Context 是抽象类
public abstract class Context {
     ...
     public abstract Object getSystemService(String name);  //获得系统级服务
     public abstract void startActivity(Intent intent);     //通过一个Intent启动Activity
     public abstract boolean bindService(@RequiresPermission Intent service,
            @NonNull ServiceConnection conn, @BindServiceFlags int flags);  //绑定Service
     //根据文件名得到SharedPreferences对象
     public abstract SharedPreferences getSharedPreferences(String name,int mode);
     ...
}
```

- **frameworks/base/core/java/android/app/ContextImpl.java**

**Context 的具体的实现类ContextImpl：**

```
class ContextImpl extends Context {
    private final static String TAG = "ContextImpl";
    private final static boolean DEBUG = false;

// 绑定服务，具体的实现
    @Override
    public boolean bindService(Intent service, ServiceConnection conn, int flags) {
        warnIfCallingFromSystemProcess();
        return bindServiceCommon(service, conn, flags, null, mMainThread.getHandler(), null,
                getUser());
    }

// 启动一个activity
    @Override
    public void startActivity(Intent intent) {
        warnIfCallingFromSystemProcess();
        startActivity(intent, null);
    }
```

- **frameworks/base/core/java/android/content/ContextWrapper.java**

ContextWrapper 对Context类的一种包装，该类的构造函数包含了一个真正的Context引用，即ContextIml对象

```
public class ContextWrapper extends Context {
```

```
    @UnsupportedAppUsage
    Context mBase; // Context 实际是实现类 ContextImpl

    public ContextWrapper(Context base) {
        mBase = base;
    }

// 传入 Context 对象
    protected void attachBaseContext(Context base) {
        if (mBase != null) {
            throw new IllegalStateException("Base context already set");
        }
        mBase = base;
    }

// 获取Context 对象
    /**
     * @return the base context as set by the constructor or setBaseContext
     */
    public Context getBaseContext() {
        return mBase;
    }

// 调用 ContextImpl方法去启动 activity
    @Override
    public void startActivity(Intent intent) {
        mBase.startActivity(intent);
    }
```

- **frameworks/base/core/java/android/view/ContextThemeWrapper.java**

包含了主题(Theme)相关的接口，即android:theme属性指定的。只有Activity需要主题

```
public class ContextThemeWrapper extends ContextWrapper {
    @UnsupportedAppUsage
    private int mThemeResource;
    @UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.P, trackingBug = 123768723)
    private Resources.Theme mTheme;

    public ContextThemeWrapper(Context base, @StyleRes int themeResId) {
        super(base);
        mThemeResource = themeResId;
    }

    public ContextThemeWrapper(Context base, Resources.Theme theme) {
        super(base);
        mTheme = theme;
    }

    @Override
    protected void attachBaseContext(Context newBase) {
        super.attachBaseContext(newBase);
    }
```

## 2. Activity的 Context 创建

Activity 的启动流程可以看这篇文章：【安卓 R 源码】Activity 启动流程及其生命周期源码分析

- **frameworks/base/core/java/android/app/ActivityThread.java**

```
    /**  Core implementation of activity launch. */
    private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {
        ActivityInfo aInfo = r.activityInfo;
        if (r.packageInfo == null) {
            r.packageInfo = getPackageInfo(aInfo.applicationInfo, r.compatInfo,
                    Context.CONTEXT_INCLUDE_CODE);
        }

        ComponentName component = r.intent.getComponent();
        if (component == null) {
```

目录



C 笔记

```
            component = r.intent.resolveActivity(
                    mInitialApplication.getPackageManager());
            r.intent.setComponent(component);
        }

        if (r.activityInfo.targetActivity != null) {
            component = new ComponentName(r.activityInfo.packageName,
                    r.activityInfo.targetActivity);
        }

// 1. 此处创建了 context 对象，ContextImpl
        ContextImpl appContext = createBaseContextForActivity(r);
        Activity activity = null;
        try {
            java.lang.ClassLoader cl = appContext.getClassLoader();

// 2. 通过反射创建 activity 对象
            activity = mInstrumentation.newActivity(
                    cl, component.getClassName(), r.intent);
            StrictMode.incrementExpectedActivityCount(activity.getClass());
            r.intent.setExtrasClassLoader(cl);
            r.intent.prepareToEnterProcess(isProtectedComponent(r.activityInfo),
                    appContext.getAttributionSource());
            if (r.state != null) {
                r.state.setClassLoader(cl);
            }
        } catch (Exception e) {
            if (!mInstrumentation.onException(activity, e)) {
                throw new RuntimeException(
                    "Unable to instantiate activity " + component
                    + ": " + e.toString(), e);
            }
        }

        try {

// 3. 创建 Application对象
            Application app = r.packageInfo.makeApplication(false, mInstrumentation);

            if (localLOGV) Slog.v(TAG, "Performing launch of " + r);
            if (localLOGV) Slog.v(
                    TAG, r + ": app=" + app
                    + ", appName=" + app.getPackageName()
                    + ", pkg=" + r.packageInfo.getPackageName()
                    + ", comp=" + r.intent.getComponent().toShortString()
                    + ", dir=" + r.packageInfo.getAppDir());

            if (activity != null) {
                CharSequence title = r.activityInfo.loadLabel(appContext.getPackageManager());
                Configuration config =
                        new Configuration(mConfigurationController.getCompatConfiguration());
                if (r.overrideConfig != null) {
                    config.updateFrom(r.overrideConfig);
                }
                if (DEBUG_CONFIGURATION) Slog.v(TAG, "Launching activity "
                        + r.activityInfo.name + " with config " + config);
                Window window = null;
                if (r.mPendingRemoveWindow != null && r.mPreserveWindow) {
                    window = r.mPendingRemoveWindow;
                    r.mPendingRemoveWindow = null;
                    r.mPendingRemoveWindowManager = null;
                }

                // Activity resources must be initialized with the same loaders as the
                // application context.
                appContext.getResources().addLoaders(
                        app.getResources().getLoaders().toArray(new ResourcesLoader[0]));

// 4. 设置外部context，setOuterContext，参数为activity
                appContext.setOuterContext(activity);

// 5. appContext作为参数传入到 activity 方法attach
```

目录



C 笔记

```
                activity.attach(appContext, this, getInstrumentation(), r.token,
                        r.ident, app, r.intent, r.activityInfo, title, r.parent,
                        r.embeddedID, r.lastNonConfigurationInstances, config,
                        r.referrer, r.voiceInteractor, window, r.configCallback,
                        r.assistToken, r.shareableActivityToken);
```

**1. 此处创建了 context 对象，ContextImpl appContext = createBaseContextForActivity**

```
// 调用其私有方法设置
    private ContextImpl createBaseContextForActivity(ActivityClientRecord r) {
        final int displayId = ActivityClient.getInstance().getDisplayId(r.token);
        ContextImpl appContext = ContextImpl.createActivityContext(
                this, r.packageInfo, r.activityInfo, r.token, displayId, r.overrideConfig);

```

- frameworks/base/core/java/android/app/ContextImpl.java

```
    @UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.R, trackingBug = 170729553)
    static ContextImpl createActivityContext(ActivityThread mainThread,
            LoadedApk packageInfo, ActivityInfo activityInfo, IBinder activityToken, int displayId,
            Configuration overrideConfiguration) {
        if (packageInfo == null) throw new IllegalArgumentException("packageInfo");
        String[] splitDirs = packageInfo.getSplitResDirs();
        ClassLoader classLoader = packageInfo.getClassLoader();

// 创建了ContextImpl 对象

        ContextImpl context = new ContextImpl(null, mainThread, packageInfo, ContextParams.EMPTY,
                attributionTag, null, activityInfo.splitName, activityToken, null, 0, classLoader,
                null);
// 设置了resource
        context.setResources(resourcesManager.createBaseTokenResources(activityToken,
                packageInfo.getResDir(),
                splitDirs,
                packageInfo.getOverlayDirs(),
                packageInfo.getOverlayPaths(),
                packageInfo.getApplicationInfo().sharedLibraryFiles,
                displayId,
                overrideConfiguration,
                compatInfo,
                classLoader,
                packageInfo.getApplication() == null ? null
                        : packageInfo.getApplication().getResources().getLoaders()));
        context.mDisplay = resourcesManager.getAdjustedDisplay(displayId,
                context.getResources());
        return context;
    }

// 可以通过 getResources获取对应 Resources
    @Override
    public Resources getResources() {
        return mResources;
    }
```

注释4中， 设置外部context，setOuterContext，参数为activity

- 作用：把Activity的实例传递给ContextImpl，这样ContextImpl中的mOuterContext就可以调用Activity的变量和方法

```
    @UnsupportedAppUsage
    final void setOuterContext(@NonNull Context context) {
        mOuterContext = context;
        // TODO(b/149463653): check if we still need this method after migrating IMS to
        //  WindowContext.
        if (mOuterContext.isUiContext() && mContextType <= CONTEXT_TYPE_DISPLAY_CONTEXT) {
            mContextType = CONTEXT_TYPE_WINDOW_CONTEXT;
            mIsConfigurationBasedContext = true;
        }
```

目录



C 笔记

```
    }11
    @UnsupportedAppUsage
    final Context getOuterContext() {
        return mOuterContext;
    }
```

### 5. appContext作为参数传入到 activity 方法attach

新创建的ContextImpl对象传递到Activity的attach方法

- frameworks/base/core/java/android/app/Activity.java

```
    @UnsupportedAppUsage(maxTargetSdk = Build.VERSION_CODES.R, trackingBug = 170729553)
    final void attach(Context context, ActivityThread aThread,
            Instrumentation instr, IBinder token, int ident,
            Application application, Intent intent, ActivityInfo info,
            CharSequence title, Activity parent, String id,
            NonConfigurationInstances lastNonConfigurationInstances,
            Configuration config, String referrer, IVoiceInteractor voiceInteractor,
            Window window, ActivityConfigCallback activityConfigCallback, IBinder assistToken,
            IBinder shareableActivityToken) {
        attachBaseContext(context);

        mFragments.attachHost(null /*parent*/);

        mWindow = new PhoneWindow(this, window, activityConfigCallback);
        mWindow.setWindowControllerCallback(mWindowControllerCallback);
        mWindow.setCallback(this);
        mWindow.setOnWindowDismissedCallback(this);
        mWindow.getLayoutInflater().setPrivateFactory(this);
        if (info.softInputMode != WindowManager.LayoutParams.SOFT_INPUT_STATE_UNSPECIFIED) {
            mWindow.setSoftInputMode(info.softInputMode);
        }
        if (info.uiOptions != 0) {
            mWindow.setUiOptions(info.uiOptions);
        }
        mUiThread = Thread.currentThread();

        mMainThread = aThread;
        mInstrumentation = instr;
        mToken = token;
        mAssistToken = assistToken;
        mShareableActivityToken = shareableActivityToken;
        mIdent = ident;
        mApplication = application;
        mIntent = intent;
```

此处调用父类ContextThemeWrapper的attachBaseContext方法并最终调用ContextWrapper类的attachBaseContext方法，将新创建的ContextImpl对象赋值给ContextWrapper的成员变量mBase，这样ContextWrapper及其子类的mBase成员变量就被实例化为ContextImpl对象。

```
public class Activity extends ContextThemeWrapper

    @Override
    protected void attachBaseContext(Context newBase) {
// 调用其父类的方法
        super.attachBaseContext(newBase);
        if (newBase != null) {
            newBase.setAutofillClient(this);
            newBase.setContentCaptureOptions(getContentCaptureOptions());
        }
    }

frameworks/base/core/java/android/view/ContextThemeWrapper.java
public class ContextThemeWrapper extends ContextWrapper {
// 也是调用父类方法
    @Override
    protected void attachBaseContext(Context newBase) {
        super.attachBaseContext(newBase);
```

目录



C 笔记

```
    }
```

Activity通过ContextWrapper的成员mBase来引用ContextImpl对象，即Activity组件可通过这个ContextImpl对象来执行一些具体的操作（启动Service等）。同时ContextImpl类又通过自己的成员变量mOuterContext引用了与它关联的Activity，这样ContextImpl类也可以操作Activity。

因此说明一个Activity就有一个Context，而且生命周期和Activity类相同。

## 3. Application 的 Context 创建

如上述代码注释 3.中，创建 Application对象，Application app = r.packageInfo.makeApplication(false, mInstrumentation);

- **frameworks/base/core/java/android/app/LoadedApk.java**

```
    @UnsupportedAppUsage
    public Application makeApplication(boolean forceDefaultAppClass,
            Instrumentation instrumentation) {
        if (mApplication != null) {
            return mApplication;
        }

        Application app = null;

        String appClass = mApplicationInfo.className;
        if (forceDefaultAppClass || (appClass == null)) {
            appClass = "android.app.Application";
        }

        try {
            final java.lang.ClassLoader cl = getClassLoader();
            if (!mPackageName.equals("android")) {
                Trace.traceBegin(Trace.TRACE_TAG_ACTIVITY_MANAGER,
                        "initializeJavaContextClassLoader");
                initializeJavaContextClassLoader();
                Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
            }

            // Rewrite the R 'constants' for all library apks.
            SparseArray<String> packageIdentifiers = getAssets().getAssignedPackageIdentifiers(
                    false, false);
            for (int i = 0, n = packageIdentifiers.size(); i < n; i++) {
                final int id = packageIdentifiers.keyAt(i);
                if (id == 0x01 || id == 0x7f) {
                    continue;
                }

                rewriteRValues(cl, packageIdentifiers.valueAt(i), id);
            }
// 1. 创建 ContextImpl 对象
            ContextImpl appContext = ContextImpl.createAppContext(mActivityThread, this);
            // The network security config needs to be aware of multiple
            // applications in the same process to handle discrepancies
            NetworkSecurityConfigProvider.handleNewApplication(appContext);
// 2. 创建 Application 对象
            app = mActivityThread.mInstrumentation.newApplication(
                    cl, appClass, appContext);
// 3. 设置 外部的context
            appContext.setOuterContext(app);
        } catch (Exception e) {
            if (!mActivityThread.mInstrumentation.onException(app, e)) {
                Trace.traceEnd(Trace.TRACE_TAG_ACTIVITY_MANAGER);
                throw new RuntimeException(
                    "Unable to instantiate application " + appClass
                    + " package " + mPackageName + ": " + e.toString(), e);
            }
        }
        mActivityThread.mAllApplications.add(app);
        mApplication = app;
```

**// 1. 创建 ContextImpl 对象**

目录



C 笔记

```
    @UnsupportedAppUsage
    static ContextImpl createAppContext(ActivityThread mainThread, LoadedApk packageInfo) {
        return createAppContext(mainThread, packageInfo, null);
    }

    static ContextImpl createAppContext(ActivityThread mainThread, LoadedApk packageInfo,
            String opPackageName) {
        if (packageInfo == null) throw new IllegalArgumentException("packageInfo");
        ContextImpl context = new ContextImpl(null, mainThread, packageInfo,
            ContextParams.EMPTY, null, null, null, null, null, 0, null, opPackageName);
        context.setResources(packageInfo.getResources());
        context.mContextType = isSystemOrSystemUI(context) ? CONTEXT_TYPE_SYSTEM_OR_SYSTEM_UI
                : CONTEXT_TYPE_NON_UI;
        return context;
    }
```

**// 2. 创建 Application 对象**

调用Instrumentation对象newApplication方法来创建Application对象

- frameworks/base/core/java/android/app/Instrumentation.java

```
    public Application newApplication(ClassLoader cl, String className, Context context)
            throws InstantiationException, IllegalAccessException,
            ClassNotFoundException {
        Application app = getFactory(context.getPackageName())
                .instantiateApplication(cl, className);
        app.attach(context);
        return app;
    }

frameworks/base/core/java/android/app/Application.java

public class Application extends ContextWrapper implements ComponentCallbacks2 {
    private static final String TAG = "Application";

    @UnsupportedAppUsage
    /* package */ final void attach(Context context) {
// 调用父类的 attachBaseContext方法
        attachBaseContext(context);
        mLoadedApk = ContextImpl.getImpl(context).mPackageInfo;
    }

frameworks/base/core/java/android/content/ContextWrapper.java

    protected void attachBaseContext(Context base) {
        if (mBase != null) {
            throw new IllegalStateException("Base context already set");
        }
        mBase = base;
    }

```

和之前一样，通过种种方法的传递，调用Application的父类方法attachBaseContext将ContextImpl对象赋值给ContextWrapper类的成员变量mBase，最终使ContextWrapper及其子类包含ContextImpl对象的引用。

注释：3. 设置 外部的context，将新创建的Application对象赋值给ContextImpl对象的成员变量mOuterContext，既让ContextImpl内部持有Application对象的引用。

这样就让二者之间都持有互相的对象，另外可说明一个Application就包含一个Context，而且生命周期和Application类相同，且于应用程序的生命周期相同。

## 4. Service 的 Context 创建

启动Service一般是通过startService和bindService方式，但是无论通过哪种方式来创建新的Service其最终都是通过ActivityThread类的handleCreateService()方法来执行操作

绑定服务的流程可以看：【安卓 R 源码】 bindService 源码分析



C 笔记

- **frameworks/base/core/java/android/app/ActivityThread.java**

```
    private void handleCreateService(CreateServiceData data) {
        // If we are getting ready to gc after going to the background, well
        // we are back active so skip it.
        unscheduleGcIdler();

        LoadedApk packageInfo = getPackageInfoNoCheck(
                data.info.applicationInfo, data.compatInfo);
        Service service = null;
        try {
            if (localLOGV) Slog.v(TAG, "Creating service " + data.info.name);

// 创建 Application 对象
            Application app = packageInfo.makeApplication(false, mInstrumentation);

            final java.lang.ClassLoader cl;
            if (data.info.splitName != null) {
                cl = packageInfo.getSplitClassLoader(data.info.splitName);
            } else {
                cl = packageInfo.getClassLoader();
            }
// 1. 通过反射获取到 service 对象
            service = packageInfo.getAppFactory()
                    .instantiateService(cl, data.info.name, data.intent);

// 2. 获取 context对象
            ContextImpl context = ContextImpl.getImpl(service
                    .createServiceBaseContext(this, packageInfo));
            if (data.info.splitName != null) {
                context = (ContextImpl) context.createContextForSplit(data.info.splitName);
            }
            if (data.info.attributionTags != null && data.info.attributionTags.length > 0) {
                final String attributionTag = data.info.attributionTags[0];
                context = (ContextImpl) context.createAttributionContext(attributionTag);
            }
            // Service resources must be initialized with the same loaders as the application
            // context.
            context.getResources().addLoaders(
                    app.getResources().getLoaders().toArray(new ResourcesLoader[0]));
 // 双向绑定context 和service
            context.setOuterContext(service);
// 3. 与 context 绑定
            service.attach(context, this, data.info.name, data.token, app,
                    ActivityManager.getService());
// 4. 回调 onCreate 方法
            service.onCreate();
            mServicesData.put(data.token, data);
            mServices.put(data.token, service);
            try {
                ActivityManager.getService().serviceDoneExecuting(
                        data.token, SERVICE_DONE_EXECUTING_ANON, 0, 0);
            } catch (RemoteException e) {
                throw e.rethrowFromSystemServer();
```

### 2. 获取 context对象

和 【 Application 的 Context 创建】，也是调用 createAppContext创建 ContextImpl 对象

```
    @UnsupportedAppUsage
    static ContextImpl createAppContext(ActivityThread mainThread, LoadedApk packageInfo) {
        return createAppContext(mainThread, packageInfo, null);
    }

    static ContextImpl createAppContext(ActivityThread mainThread, LoadedApk packageInfo,
            String opPackageName) {
        if (packageInfo == null) throw new IllegalArgumentException("packageInfo");
        ContextImpl context = new ContextImpl(null, mainThread, packageInfo,
            ContextParams.EMPTY, null, null, null, null, null, 0, null, opPackageName);
```

目录



C 笔记

```
        context.setResources(packageInfo.getResources());
        context.mContextType = isSystemOrSystemUI(context) ? CONTEXT_TYPE_SYSTEM_OR_SYSTEM_UI
                : CONTEXT_TYPE_NON_UI;
        return context;
    }
```

3. 与 context 绑定

- frameworks/base/core/java/android/app/Service.java

```
public abstract class Service extends ContextWrapper implements ComponentCallbacks2,
        ContentCaptureManager.ContentCaptureClient {
    private static final String TAG = "Service";

    public final void attach(
            Context context,
            ActivityThread thread, String className, IBinder token,
            Application application, Object activityManager) {
        attachBaseContext(context);
        mThread = thread;           // NOTE:  unused - remove?
        mClassName = className;
        mToken = token;
        mApplication = application;
        mActivityManager = (IActivityManager)activityManager;
        mStartCompatibility = getApplicationInfo().targetSdkVersion
                < Build.VERSION_CODES.ECLAIR;

        setContentCaptureOptions(application.getContentCaptureOptions());
    }

// 相同地，也是调用父类方法
    @Override
    protected void attachBaseContext(Context newBase) {
        super.attachBaseContext(newBase);
        if (newBase != null) {
            newBase.setContentCaptureOptions(getContentCaptureOptions());
        }
    }
```

## 4. Context 导致内存泄漏问题：

**1. 错误方式获取 Context**

```

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        TextView tv = new TextView(MainActivity.this);

        TextView tv2 = new TextView(getApplicationContext()); //这里不能使用getApplicationContext()
    }
```

TextView tv = new TextView(MainActivity.this)；tv 这个view 是依赖Activity（界面而存在的）；Activity销毁，tv也会销毁

如果使用TextView tv = new TextView(getApplicationContext())，可能Activity销毁了，但是整个应用程序还没有销毁，这样这个tv会变成空指针，导致内存泄露。

**2. View持有activity 的引用：**

```
public class MainActivity extends Activity {
    private static Drawable mDrawable;

    @Override
    protected void onCreate(Bundle saveInstanceState) {
        super.onCreate(saveInstanceState);
```

目录



C 笔记

```
        setContentView(R.layout.activity_main);
        ImageView iv = new ImageView(this);
        mDrawable = getResources().getDrawable(R.drawable.ic_launcher);
        iv.setImageDrawable(mDrawable);
    }
}
```

Drawable是一个静态的对象，当ImageView设置这个Drawable时，ImageView保存了mDrawable的引用，而ImageView传入的this是MainActivity的mContext，因为被static修饰的mDrawable是常驻内存的，MainActivity是它的间接引用，MainActivity被销毁时，也不能被GC掉，所以造成内存泄漏。

**在使用 Context应该尽量注意如下几点：**

1. 如果不涉及Activity的主题样式，尽量使用Application的Context。
2. 不要让生命周期长于Activity的对象持有其的引用。
3. 尽量不要在Activity中使用非静态内部类，因为非静态内部类会隐式持有外部类示例的引用，如果使用静态内部类，将外部实例引用作为弱引用持有。

参考：

Android中Context源码分析

Android 10 源码理解上下文Context

Android Context的设计思想及源码分析

Android源码解析--Context详解


Android context源码详解及深入分析 08-31
主要介绍了Android context源码详解及深入分析的相关资料,这里对Android Context 如何使用进行了详细介绍，需要的朋友可以参考下

【安卓 R 源码】 bindService 源码分析 mike_jun的博客 2482
使用bindService主要分两种情形: 1. Service的调用者client与Service在同一个App中； 2. Service的调用者client是App1中的一个Activity，而…

context.getContentResolver().query()的使用 1-2
文章浏览阅读454次。当使用SQLite数据库进行查询操作时,可以使用选择条件(selection)和选择条件参数(selectionArgs)来过滤结果。在这个…

ContentResolver.query流程分析 12-28
文章浏览阅读1k次。ContentProvider查询流程

context.getContentResolver().query()详细用法详解 awodefengduanwu的博客 5048
1.获取联系人姓名 一个简单的例子，这个函数获取设备上所有的联系人ID和联系人NAME。 [java] view plain copy public void fetchAllContac…

android contentresolver权限,求助关于getcontentresolver（）.query（） weixin_28871097的博客 1000
该楼层疑似违规已被系统折叠隐藏此楼查看此楼今天搞了一天，用getcontentresolver().query()l来获取音乐列表，却啥也没有，是不是需要…

Android ContentProvider(跨进程) 1-2
文章浏览阅读2k次。1.内容提供者ContentProviderContentProvider是android四大组件之一,它为不同的应用之间实现数据共享,提供统一的接…

在执行context.getContentResolver.query()方法时出现错误。 12-21
文章浏览阅读1.5k次。1. 在执行context.getContentResolver.query()方法时出现错误。 07-15 18:46:13.470: E/AndroidRuntime(13624): FAT…

Android 13 有线网变更(用到的可以收藏) wenzhi的博客 2024
有线网设置新路径：try {try {try {从上面看，主要是api加了限制： maxTargetSdk = Build.VERSION_CODES.R //Android11maxTargetSdk 表…

Android 10.0 系统framework禁止访问应用信息页 安卓兼职framework和app工程师的博客 1779
在定制化开发中，在app崩溃后，会弹出弹窗提醒奔回了，然后弹窗可以通过点击应用信息按钮进入应用信息页这对于权限的管控不好处理…

Android7.0版本后 Uri和文件路径互相转换封装类,实现系统分享功能及 Fil... 1-2
文章浏览阅读1.7w次,点赞7次,收藏26次。在调用系统相机、相册时,经常需要进行Uri和File路径的互相转换,并且在项目中遇到按照百度查到…

剖析ContentProvider的query操作_contentprovider query 1-1
文章浏览阅读1.4k次。年终(农历年)的一篇任务文档。本篇剖析ContentProvider的query操作,包括了以下内容:获取ContentProvider对象Serv…

Context-Menu.Android源码 10-16
Context-Menu.Android源码，是一个很不错的C++代码，有兴趣的伙伴们抽时间可以看一下把。

Android项目源码安卓程序猿面试复习APP 04-23
注意：本项目使用android stdio开发，eclipse可能无法直接导入。 一、项目简介 为了更好地准备面试Android开发这一职位，于是就到应用…

...什么原因导致Android的ContentResolver.query()返回null? 12-30


目录



C 笔记