转载请注明链接:

Context详解

版本号:2019-03-31(14:24)

理解

1、Context是什么?

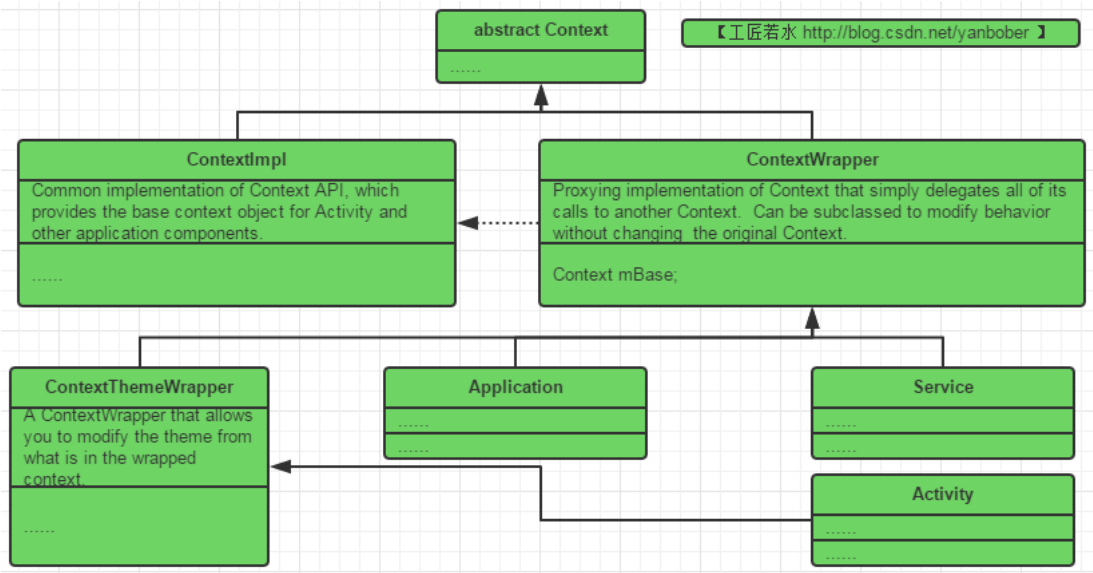
> 1. 一种抽象类, 提供app环境的信息，也就是上下文。

```
public abstract class Context {
}
```

2、Context有什么用?

> 1. 能用于访问app特定的资源和类
> 2. 能系统级别的操作，例如:
>   1. 启动activity
>   2. 发广播
>   3. 接收Intent

3、Context的继承关系



> Context具有4个相关的类:
>
> 1. 抽象类Context
> 2. ContextImpl
> 3. ContextWrapper
> 4. ContextThemeWrapper

4、ContextImpl的作用?

> 1. 继承自Context，实现了Context中所有的API

5、ContextWrapper的作用?

> 1. Context的包装类，能让继承自ContextWrapper的子类修改行为，而不会影响到Context。
> 2. ContextWrapper内部有局部变量mBase，用于持有ContextImpl对象(context实现类对象)
> 3. attachBaseContext()方法在 `Activity`启动过程中的`handleLaunchActivity()->performLaucnActivity()->activity.attach()` 方法中，就是创建了ContextImpl对象，然后建立的ContextImpl和Activity的联系

```
public class ContextWrapper extends Context {
    Context mBase;

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

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

6、为什么有了ContextImpl(实现类)还要有一个ContextWrapper?

> 1. 这是一种 装饰者模式 ： ContextWrapper内部持有ContextImpl的引用
> 2. 目的是，向现有的对象(ContextImpl对象)添加新功能，却不改变其结构。
> 3. ContextThemeWrapper就是在ContextWrapper的基础上添加主题相关的功能

7、为什么Application和Service继承自ContextWrapper？而Activity继承自 `ContextThemeWrapper` ？

> 1. Activity有主题
> 2. Application和Service不需要界面，也就不需要主题

拓展

深度拓展

1、Context是什么?

2、Context历史背景?

> 2006年，Android开源项目建立之初，就有Context了

3、为什么要有Context?

4、Context的作者?

Context源码解析

1、Context有哪些重要方法？进行分类

权限相关

2、Context具有权限相关的方法

```
 // 判断特定的process和user id，是否具有指定的权限
@PackageManager.PermissionResult
public abstract int checkPermission(@NonNull String permission, int pid, int uid);

// 检查自己的权限
@PackageManager.PermissionResult
public abstract int checkSelfPermission(@NonNull String permission);
```

> 1. 判断是否具有目标权限
> 2. 返回结果有两种: PackageManager的 `PERMISSION_GRANTED` (允许)和 `PERMISSION_DENIED` (拒绝)

四大组件相关

3、Context中Service相关方法

> 1. Service的启动和停止
> 2. Service的绑定和解绑

```
// 1. 启动Service
public abstract ComponentName startService(Intent service);
// 2. 启动前台Service
public abstract ComponentName startForegroundService(Intent service);
// 3. 停止Service
public abstract boolean stopService(Intent service);

// 4. 绑定Service
public abstract boolean bindService(Intent service, ServiceConnection conn, int flags);
// 5. 解绑Service
public abstract void unbindService(@NonNull ServiceConnection conn);
```

4、广播接收器的动态注册和解注册，以及广播的发送

> 1. 广播接收器的注册和解注册
> 2. 三种广播的发送：普通、有序、粘性广播

```
// 1. 注册广播接收器
public abstract Intent registerReceiver(BroadcastReceiver receiver,IntentFilter filter);
// 2. 解注册广播接收器
public abstract void unregisterReceiver(BroadcastReceiver receiver);
// 3. 发送普通广播
public abstract void sendBroadcast(Intent intent);
// 4. 发送有序广播
public abstract void sendOrderedBroadcast(Intent intent, receiverPermission);
// 5. 发送粘性广播，已经废弃！有安全隐患。
public abstract void sendStickyBroadcast(@RequiresPermission Intent intent);
```

5、启动Activity

> 1. startActivity
> 2. startActivityForResult

```
// 1. 启动Activity
public abstract void startActivity(@RequiresPermission Intent intent);

// 2. startActivityForResult
public void startActivityForResult(String who, Intent intent, int requestCode, Bundle options) {
  //
}
```

6、ContentProvider

```
// 获取到ContentProvider的ContentResolver
public abstract ContentResolver getContentResolver();
```

数据库、文件、目录

1、数据库相关

```
// 1. 创建APP私有的SQlite数据库
public abstract SQLiteDatabase openOrCreateDatabase(String name,int mode, CursorFactory factory);
// 2. 返回openOrCreateDatabase创建的SQlite数据库的绝对路径
public abstract File getDatabasePath(String name);
```

2、外部缓存相关

```
// 2. 外部缓存
public abstract File getExternalCacheDir();
public abstract File[] getExternalCacheDirs();
public abstract File getExternalFilesDir(@Nullable String type);
public abstract File[] getExternalFilesDirs(String type);
// 外部存储，存放多媒体文件，能够被其他app通过MediaStore访问到
public abstract File[] getExternalMediaDirs();
```

3、内部缓存相关

```
// 1. 缓存目录，磁盘空间不足时，系统会自动删除
public abstract File getCacheDir();
// 2. 缓存app运行时动态生成的代码，app升级和系统升级时，这些文件都会被删除
public abstract File getCodeCacheDir();
// 3. App私有的文件目录，不建议直接使用，应该用其他的API{@link #getFilesDir()}, {@link #getCacheDir()}, {@link #getDir(String, int)}
public abstract File getDataDir();


// 1. 返回openFileOutput调用时存储文件的目录路径
public abstract File getFilesDir();
// 2. 打开文件输出流，文件不存在时创建该文件。
public abstract FileOutputStream openFileOutput(String name, @FileMode int mode);
// 3. 获取到openFileOutput锁创建的文件路径
public abstract File getFileStreamPath(String name);

// 1. 创建自定义的目录，用于存储自己的数据
public abstract File getDir(String name, @FileMode int mode);
```

资源类、sp、系统服务

1、资源类相关的API

```
// 1. 返回App包的AssetManager
public abstract AssetManager getAssets();
// 2. 获取到Resources-Resources必须要是AssetManager返回的
public abstract Resources getResources();
// 3. 获取到主题
public int getThemeResId() {
    return 0;
}
public abstract Resources.Theme getTheme();

// 4. 获取颜色
public final int getColor(@ColorRes int id) {
    return getResources().getColor(id, getTheme());
}

// 5. 获取到Drawable
public final Drawable getDrawable(@DrawableRes int id) {
    return getResources().getDrawable(id, getTheme());
}


// 6. String
public final String getString(@StringRes int resId) {
    return getResources().getString(resId);
}
// 7. Text
public final CharSequence getText(@StringRes int resId) {
    return getResources().getText(resId);
}


// 8. 墙纸相关都已经废弃
public abstract Drawable getWallpaper();
public abstract void setWallpaper(Bitmap bitmap) throws IOException;
```

2、SharedPreference

```
// 8. SharePreference
public abstract SharedPreferences getSharedPreferences(String name, @PreferencesMode int mode);
```

系统服务

1、获取到系统服务

```
public abstract @Nullable Object getSystemService(@ServiceName String name);
```

2、可以获取的系统服务有下面

Context字符串	系统服务	作用
WINDOW_SERVICE	android.view.WindowManager	最顶层的WindowManager可以放置自定义的Windows
LAYOUT_INFLATER_SERVICE	android.view.LayoutInflater	在当前Context中展开布局资源
ACTIVITY_SERVICE	android.app.ActivityManager	和系统的activity状态进行交互
POWER_SERVICE	android.os.PowerManager	能量管理
ALARM_SERVICE	android.app.AlarmManager	在规定的时间点接收到通知(intents)
NOTIFICATION_SERVICE	android.app.NotificationManager	通知栏，通知用户关于后台的事件
KEYGUARD_SERVICE	android.app.KeyguardManager	控制键盘保护
LOCATION_SERVICE	android.location.LocationManager	控制位置更新(Gps)
SEARCH_SERVICE	android.app.SearchManager	处理搜索
VIBRATOR_SERVICE	android.os.Vibrator	“震动器”
CONNECTIVITY_SERVICE	android.net.ConnectivityManager	网络连接的管理
IPSEC_SERVICE	android.net.IpSecManager	管理Socket和Networks的IPSec
WIFI_SERVICE	android.net.wifi.WifiManager	管理WIFI连接
WIFI_AWARE_SERVICE	android.net.wifi.aware.WifiAwareManager	管理WIFI的感知发现和连接

Context字符串	系统服务	作用
WIFI_P2P_SERVICE	android.net.wifi.p2p.WifiP2pManager	管理WIFI的直接连接
INPUT_METHOD_SERVICE	android.view.inputmethod.InputMethodManager	管理输入
UI_MODE_SERVICE	android.app.UiModeManager	控制UI模式
DOWNLOAD_SERVICE	android.app.DownloadManager	用于请求HTTP下载
BATTERY_SERVICE	android.os.BatteryManager	管理电源状态
JOB_SCHEDULER_SERVICE	android.app.job.JobScheduler	处理计划的任务
NETWORK_STATS_SERVICE	android.app.usage.NetworkStatsManager	进行网络使用分析
HARDWARE_PROPERTIES_SERVICE	android.os.HardwarePropertiesManager	访问硬件属性

Handler

1、获取到主线程的Looper

```
public abstract Looper getMainLooper();
```

Package相关

1、Package相关的

```
// 1. 获取该包的ClassLoader，可以用于派生Class
public abstract ClassLoader getClassLoader();
// 2. App的包名
public abstract String getPackageName();
// 3. PackageManager
public abstract PackageManager getPackageManager();
// 4. Package的资源路径
public abstract String getPackageResourcePath();
```

Application相关

1、Application相关的

```
public abstract Context getApplicationContext();
public abstract ApplicationInfo getApplicationInfo();
```

ContextImpl源码解析

1、ContextImpl直接继承自Context

```
class ContextImpl extends Context {
}
```

2、ContextImpl具有的重要属性

> 除了LoadedApk和ActivitThread是default，其他属性都是private-也就是无法继承到子类中

```
// 1. SharedPreferences相关
private static ArrayMap<String, ArrayMap<File, SharedPreferencesImpl>> sSharedPrefsCache;
private ArrayMap<String, File> mSharedPrefsPaths;

// 2. 主线程
final @NonNull ActivityThread mMainThread;

// 3. LoadedApk
final @NonNull LoadedApk mPackageInfo;

// 4. ClassLoader
private @Nullable ClassLoader mClassLoader;
// LoadeApk中获取到ClassLoader
@Override
public ClassLoader getClassLoader() {
    return mClassLoader != null ? mClassLoader : (mPackageInfo != null ? mPackageInfo.getClassLoader() : ClassLoader.getSystemClassLoader());
}

// 5. IBinder mActivityToken
private final @Nullable IBinder mActivityToken;

// 6. Resources、ResourcesManager、Theme
private final @NonNull ResourcesManager mResourcesManager;
private @NonNull Resources mResources;
private Resources.Theme mTheme = null;

// 7. PackageManager-能获取到app的信息
private PackageManager mPackageManager;
```

ContextWrapper、ContextThemeWrapper

1、ContextWrapper仅仅是持有ContextImpl对象的引用，其他的方法调用都是直接转交给ContextImpl处理

2、ContextThemeWrapper在ContextWrapper的基础上扩展了4个UI相关属性

> 1. 主题和Resources
> 2. LayoutInflater

```
private Resources.Theme mTheme;
private LayoutInflater mInflater;
private Configuration mOverrideConfiguration;
private Resources mResources;
```

Context的创建

Activity

1、启动Activity的过程中创造Activity所属的Context

> 1. ActivityThread的performLaunchActivity，构造ContextImpl
> 2. 利用ContextImpl的createActivityContext()构造ContextImpl对象，本质就是用ContextImpl新增的五个重要字段赋值：ClassLoader、activityToken、主线程ActivityThread、Resources、LoadedApk
> 3. 让ContextImpl持有Activity的引用
> 4. Activity.attach()方法中通过ContextWrapper新增attachBaseContext()方法，让内部的mBase持有ContextImpl的引用

```
// ActivityThread.java-创建ContextImpl，ContextImpl内部持有Activity，Activity内部的mBase持有ContextImpl(继承自ContextThemeWrapper <- ContextW
private Activity performLaunchActivity(ActivityClientRecord r, Intent customIntent) {
    // ......
    // 1. 创建一个ContextImpl对象
    ContextImpl appContext = createBaseContextForActivity(r);
    // 2.
    appContext.setOuterContext(activity);
    activity.attach(appContext, xxx);
}

// ActivityThread.java-调用ContextImpl静态方法构建属于Activity的Context
private ContextImpl createBaseContextForActivity(ActivityClientRecord r) {
    ContextImpl appContext = ContextImpl.createActivityContext(
            this, r.packageInfo, r.activityInfo, r.token, displayId, r.overrideConfig);
    // ....
    return appContext;
}

// ContextImpl.java-构造出属于Activity的ContextImpl，本质就是给ContextImpl新增的字段赋值。
static ContextImpl createActivityContext(ActivityThread mainThread,
                                         LoadedApk packageInfo, ActivityInfo activityInfo, IBinder activityToken, int displayId,
                                         Configuration overrideConfiguration) {
    // 1. 获取到LoadedApk中的ClassLoader
    ClassLoader classLoader = packageInfo.getClassLoader();
    if (packageInfo.getApplicationInfo().requestsIsolatedSplitLoading()) {
        classLoader = packageInfo.getSplitClassLoader(activityInfo.splitName);
    }

    // 2. 使用mainThread、LoadedApk、activityToken、classLoader构建ContextImpl
    ContextImpl context = new ContextImpl(null, mainThread, packageInfo, activityInfo.splitName,
            activityToken, null, 0, classLoader);
    // 3. 同过ResourcesManager构造出属于Activity的Resources，设置到ContextImpl之中
    final ResourcesManager resourcesManager = ResourcesManager.getInstance();
    context.setResources(resourcesManager.createBaseActivityResources(activityToken,
            packageInfo.getResDir(),
            splitDirs,
            packageInfo.getOverlayDirs(),
            packageInfo.getApplicationInfo().sharedLibraryFiles,
            displayId,
            overrideConfiguration,
            compatInfo,
            classLoader));

    return context;
}
```

横向拓展

1、getApplication与getApplicationContext有什么区别?

2、一个App到底有几个Context?

> 1. App中Context的总数 = Application的Context(1个)+Activity的数量+Service的数量
> 2. 广播没有Context
> 3. ContentProvider本身没有Context，但是其对应的应用肯定有一个自身的ApplicationContext

3、Context和内存泄漏

4、App中有多个Context对象，那么context.getResources()得到的资源是同一份吗

创造性拓展

纠错

应用

问题汇总

参考资料

1. Android应用Context详解及源码解析