[**Android Context与GetApplicationContext的区别及使用**](http://blog.csdn.net/u014722744/article/details/43309349)

标签： [android](http://www.csdn.net/tag/android)[context](http://www.csdn.net/tag/context)[内存泄漏](http://www.csdn.net/tag/%e5%86%85%e5%ad%98%e6%b3%84%e6%bc%8f)

2015-01-30 17:21 3325人阅读 [评论](http://blog.csdn.net/u014722744/article/details/43309349#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/u014722744/article/details/43309349#report)

分类：

android工具*（9）*

目录[(?)[+]](http://blog.csdn.net/u014722744/article/details/43309349)

转载请标明出处：<http://blog.csdn.net/lmj623565791/article/details/40481055>，本文出自：【[张鸿洋的博客](http://blog.csdn.net/lmj623565791/)】

本文大多数内容翻译自：<http://www.doubleencore.com/2013/06/context/>  我重新组织了下内容以及结构，建议大家尽可能看下原文。

**1、Context概念**

其实一直想写一篇关于Context的文章，但是又怕技术不如而误人子弟，于是参考了些资料，今天准备整理下写出来，如有不足，请指出，参考资料会在醒目地方标明。

Context，相信不管是第一天开发[**Android**](http://lib.csdn.net/base/android)，还是开发[**android**](http://lib.csdn.net/base/android)的各种老鸟，对于Context的使用一定不陌生~~你在加载资源、启动一个新的Activity、获取系统服务、获取内部文件（夹）路径、创建View操作时等都需要Context的参与，可见Context的常见性。大家可能会问到底什么是Context，Context字面意思上下文，或者叫做场景，也就是用户与[**操作系统**](http://lib.csdn.net/base/operatingsystem)操作的一个过程，比如你打电话，场景包括电话程序对应的界面，以及隐藏在背后的数据；

但是在程序的角度Context又是什么呢？在程序的角度，我们可以有比较权威的答案，Context是个抽象类，我们可以直接通过看其类结构来说明答案：

可以看到Activity、Service、Application都是Context的子类；

也就是说，Android系统的角度来理解：Context是一个场景，代表与操作系统的交互的一种过程。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。

在仔细看一下上图：Activity、Service、Application都是继承自ContextWrapper，而ContextWrapper内部会包含一个base context，由这个base context去实现了绝大多数的方法。

先扯这么多，有能力了会从别的角度去审视Context，加油~

**2、Context与ApplicationContext**

看了标题，千万不要被误解，ApplicationContext并没有这个类，其实更应该叫做：Activity与Application在作为Context时的区别。嗯，的确是这样的，大家在需要Context的时候，如果是在Activity中，大多直接传个this，当在匿名内部类的时候，因为this不能用，需要写XXXActivity.this，很多哥们会偷懒，直接就来个getApplicationContext。那么大家有没有想过，XXXActivity.this和getApplicationContext的区别呢？

XXXActivity和getApplicationContext返回的肯定不是一个对象，一个是当前Activity的实例，一个是项目的Application的实例。既然区别这么明显，那么各自的使用场景肯定不同，乱使用可能会带来一些问题。

下面开始介绍在使用Context时，需要注意的问题。

**3、引用的保持**

大家在编写一些类时，例如工具类，可能会编写成单例的方式，这些工具类大多需要去访问资源，也就说需要Context的参与。

在这样的情况下，就需要注意Context的引用问题。

例如以下的写法：

package com.mooc.shader.roundimageview;

import android.content.Context;

public class CustomManager

{

private static CustomManager sInstance;

private Context mContext;

private CustomManager(Context context)

{

this.mContext = context;

}

public static synchronized CustomManager getInstance(Context context)

{

if (sInstance == null)

{

sInstance = new CustomManager(context);

}

return sInstance;

}

//some methods

private void someOtherMethodNeedContext()

{

}

}

对于上述的单例，大家应该都不陌生（请别计较getInstance的效率问题），内部保持了一个Context的引用；

这么写是没有问题的，问题在于，这个Context哪来的我们不能确定，很大的可能性，你在某个Activity里面为了方便，直接传了个this;这样问题就来了，我们的这个类中的sInstance是一个static且强引用的，在其内部引用了一个Activity作为Context，也就是说，我们的这个Activity只要我们的项目活着，就没有办法进行内存回收。而我们的Activity的生命周期肯定没这么长，所以造成了内存泄漏。

那么，我们如何才能避免这样的问题呢？

有人会说，我们可以软引用，嗯，软引用，假如被回收了，你不怕NullPointException么。

把上述代码做下修改：

public static synchronized CustomManager getInstance(Context context)

{

if (sInstance == null)

{

sInstance = new CustomManager(context.getApplicationContext());

}

return sInstance;

}

这样，我们就解决了内存泄漏的问题，因为我们引用的是一个ApplicationContext，它的生命周期和我们的单例对象一致。

这样的话，可能有人会说，早说嘛，那我们以后都这么用不就行了，很遗憾的说，不行。上面我们已经说过，Context和Application Context的区别是很大的，也就是说，他们的应用场景（你也可以认为是能力）是不同的，并非所有Activity为Context的场景，Application Context都能搞定。

下面就开始介绍各种Context的应用场景。

**4、Context的应用场景**

大家注意看到有一些NO上添加了一些数字，其实这些从能力上来说是YES，但是为什么说是NO呢？下面一个一个解释：

数字1：启动Activity在这些类中是可以的，但是需要创建一个新的task。一般情况不推荐。

数字2：在这些类中去layout inflate是合法的，但是会使用系统默认的主题样式，如果你自定义了某些样式可能不会被使用。

数字3：在receiver为null时允许，在4.2或以上的版本中，用于获取黏性广播的当前值。（可以无视）

注：ContentProvider、BroadcastReceiver之所以在上述表格中，是因为在其内部方法中都有一个context用于使用。

好了，这里我们看下表格，重点看Activity和Application，可以看到，和UI相关的方法基本都不建议或者不可使用Application，并且，前三个操作基本不可能在Application中出现。实际上，只要把握住一点，凡是跟UI相关的，都应该使用Activity做为Context来处理；其他的一些操作，Service,Activity,Application等实例都可以，当然了，注意Context引用的持有，防止内存泄漏。

**5、总结**

好了，到此，Context的分析基本完成了，希望大家在以后的使用过程中，能够稍微考虑下，这里使用Activity合适吗？会不会造成内存泄漏？这里传入Application work吗？

由于参考内容过多，本文改为译文咯~~