Author: Bobby Wang, [wbo4958@gmail.com](mailto:wbo4958@gmail.com)

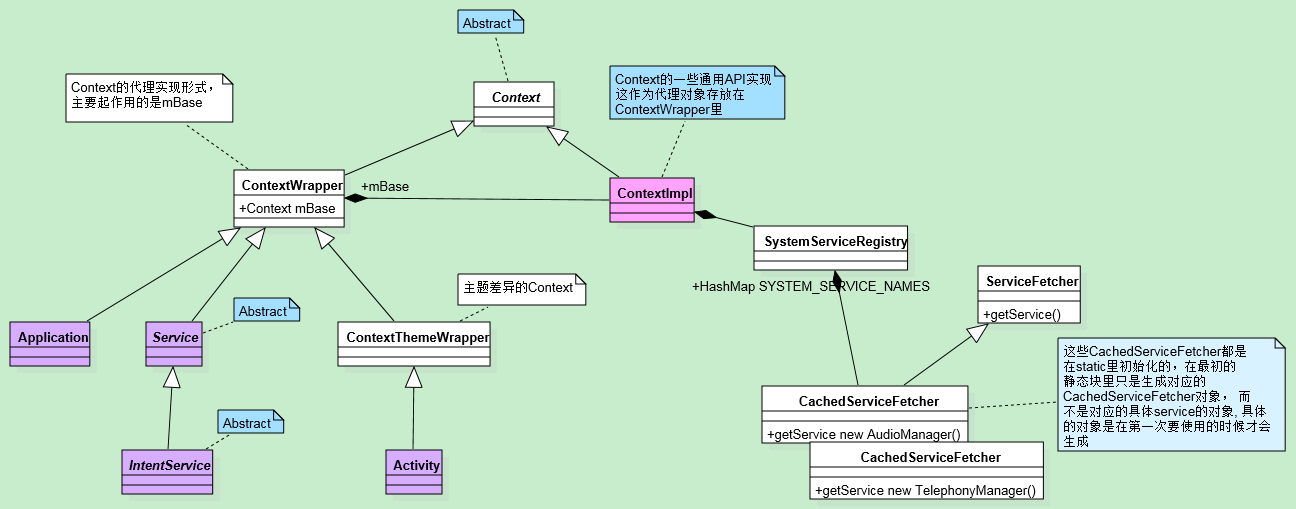
在做APP开发时，经常会使用到系统的一些服务， 如获得Wifi的信息，Telephony相关的系统，这时都会通过Context.getSystemService(String name)去获取对应的Service. 那么从Context里获得的Service和ServiceManager里getService()获得的Service是一样的么？它们是一个整样的关系的呢？

首先应该是概念上的混淆， ServiceManager类是hide的，因此如果直接通过官方的SDK是不能访问到的，那么能通过SDK访问到的那一定是Context的getSystemService()

以获得Telephony的Service为例

TelephonyManager telephonyManager = (TelephonyManager) **this**.getSystemService(Context.*TELEPHONY\_SERVICE*);

“this”是Context的实例，那么哪些类是Context的呢？从图可以看出Application/Service/Activity都是间接继承于Context



注意： 每个Context子类的实例中的mBase是不一样的，即Application/Activity/Service的实例中的具体的ContextImpl不是同一个，是不同的对象，可以通过打印 this.getBaseContext()来验证。这块内容后续会讲。

**一、ContextImpl的实例化**

当Launch一个新的Activity时，具体可以从ActivityThread中performLaunchActivity() -> createBaseContextForActivity() -> ContextImpl. createActivityContext()

**new** ContextImpl(…); 在得到一个类的对象之前，首先会初始化该类的成员变量(有个类初始化顺序… ), 这里假设是第一次创建ContextImpl实例， 实际上第一次初始化ContextImpl是在创建系统Context时，即createSystemContext()

*// The system service cache for the system services that are cached per-ContextImpl.***final** Object[] **mServiceCache** = SystemServiceRegistry.*createServiceCache*();

//通过调用SystemServiceRegistry中的静态方法createServiceCache()，该方法会先触发SystemServiceRegistry中静态方法/静态块的初始化，然后才会调用到createServiceCache()

Part 1 =====================================================================

//首先初始化SystemServiceRegistry里的静态变量/执行静态块

**final class** SystemServiceRegistry {  
**private static final** HashMap<Class<?>, String> ***SYSTEM\_SERVICE\_NAMES*** =  
 **new** HashMap<Class<?>, String>();  
 **private static final** HashMap<String, ServiceFetcher<?>> ***SYSTEM\_SERVICE\_FETCHERS*** =  
 **new** HashMap<String, ServiceFetcher<?>>();  
 **private static int** *sServiceCacheSize*;  
  
 *// Not instantiable.* **private** SystemServiceRegistry() { }  
  
 **static** {

…  
*registerService*(Context.***TELEPHONY\_SERVICE***, TelephonyManager.**class**,  
 **new** CachedServiceFetcher<TelephonyManager>() {  
 @Override  
 **public** TelephonyManager createService(ContextImpl ctx) {  
 **return new** TelephonyManager(ctx.getOuterContext());  
 }});

*…*

*}*

以Telephony Service为，展开泛型类CachedServiceFetcher

**static abstract class** CachedServiceFetcher<TelephonyManager> **implements** ServiceFetcher<TelephonyManager> {  
 **private final int** mCacheIndex;

// mCacheIndex用在 ContextImpl里的mServiceCache数组中。  
  
 **public** CachedServiceFetcher() {  
 mCacheIndex = *sServiceCacheSize*++;

//静态变量自动累加作为对应 service的指引  
 }  
  
 @Override  
 @SuppressWarnings("unchecked")  
 **public final** TelephonyManager getService(ContextImpl ctx) {  
 **final** Object[] cache = ctx.mServiceCache;

//ContextImpl里mServiceCache数组对象  
 synchronized (cache) {  
 // Fetch or create the service.  
 Object service = cache[mCacheIndex];  
 if (service == null) {  
 service = createService(ctx);

//从这里可以看出，如果该Context第一次使用TelephonyManager,在通过Activity.getSystemService(Context.TELEPHONY\_SERVICE)获得Telephony service时，是即时生成的TelephonyManager对象, 见上面registerService, 这么做是很有必要的，因为如果一开始就将所有的service都初始化，那么1, 每个Context将会非常庞大; 2, 在一个Context中，也基本上不会使用到所有的service, 一般会用到其中几个而已，完全没有必要把所有的service全部初始化出来。

cache[mCacheIndex] = service;

//缓存该service 到ContextImpl里的mServiceCache中  
 }  
 **return** (TelephonyManager)service;  
 }  
 }  
  
 **public abstract** TelephonyManager createService(ContextImpl ctx);

}

//将所有的ServiceFetcher放到 HashMap里

**private static** <T> **void** registerService(String serviceName, Class<T> serviceClass, ServiceFetcher<T> serviceFetcher) {  
 ***SYSTEM\_SERVICE\_NAMES***.put(serviceClass, serviceName);  
 ***SYSTEM\_SERVICE\_FETCHERS***.put(serviceName, serviceFetcher);  
}

Part 2 =====================================================================

初始化ContextImpl里的成员变量 mServiceCache.

**public static** Object[] createServiceCache() {  
 **return new** Object[*sServiceCacheSize*]; //用来保存所有的Service  
}

**二、获得SystemService**

Context.getSystemService(String name), 该方法最终都是通过 getBaseContext().getSystemService(String name)来获得的，即

**public** Object getSystemService(String name) {  
 **return** SystemServiceRegistry.*getSystemService*(**this**, name);  
}

**public static** Object getSystemService(ContextImpl ctx, String name) {  
 ServiceFetcher<?> fetcher = ***SYSTEM\_SERVICE\_FETCHERS***.get(name);  
 **return** fetcher != **null** ? fetcher.getService(ctx) : **null**;  
}

最终会进入part 1中的

**public final** TelephonyManager getService(ContextImpl ctx) {}

**三、通过Context得到的SystemService与ServiceManager中的Service关系？**

ServiceManager是系统级的类，是hide的，SDK是不能访问的，它的作用是作为Binder的辅助管理使用，是一个单独的进程。即提供服务的Server将自己注册给ServiceManager, 其它Client(处于不同的进程)通过ServiceManager获得Server对应的代理进行进程间通信，获Server提供的服务。

Android将ServiceManager隐藏起来不让上层APP直接访问，可能是因为

1，系统级的service是支撑起整个android系统的关键基石，如果把ServiceManager全部开放给给用户，可能会导致系统不稳定。

2，ServiceManager里也可以注册不同的服务，如果暴露给用户，用户完全可以将不停的向ServiceManger加入服务，这会导致很严重的系统性能issue.

因此为了能让用户使用到一些核心的服务，又不能让android处于不可控的状态，这时就出现了android 的系统级服务，即通过 Context 获得的系统级服务。这些Context获得的系统级服务大多是通过ServiceManager来获得具体的真正的系统服务来提供所需服务的。

如：

TelephonyManager telephonyManager = (TelephonyManager) **this**.getSystemService(Context.*TELEPHONY\_SERVICE*);

Android将所有的需要提供给用户的Telephony的相关信息通过TelephonyManager的管理，而android的telephony模块比较庞大，业务流程比较复杂，android又将telephony细分到具体的业务模块，如ITelephony.Stub, ITelecomService.Stub, IPhoneSubInfo.Stub, IIccPhoneBook.Stub, ISms.Stub, ITelephnyRegistry.Stub 等等，这些不同的模块负责不同的功能，它们作为service，将自己通过向ServiceManager注册对系统公开(注：并不是对所有上层APP)。 Telephony作为android手机中重要的一个模块，自然而然会提供相应的 对上层应用开发某些具体的服务。如果ServiceManager不是隐藏的，那么上层App将会获得所有的上述所说的这些服务，然后进行各种调用等等，这会让系统不稳定且也不安全。

因此Android将某些必要的API通过TelephonyManager来管理并提供在SDK里。

如：

**public** String getDeviceId() {  
 **try** {  
 ITelephony telephony = getITelephony();  
 **if** (telephony == **null**)  
 **return null**;  
 **return** telephony.getDeviceId(mContext.getOpPackageName());  
 } **catch** (RemoteException ex) {  
 **return null**;  
 } **catch** (NullPointerException ex) {  
 **return null**;  
 }  
}

/\*\*  
 \* **@hide** \*/  
 **private** ITelephony getITelephony() {  
 **return** ITelephony.Stub.*asInterface*(ServiceManager.*getService*(Context.***TELEPHONY\_SERVICE***));  
 }

TelephonyManager将getITelephony隐藏起来了，只提供必要的API，如getDeviceId()，同理还有

/\*\*  
\* **@hide**\*/  
**private** ITelecomService getTelecomService() {  
 **return** ITelecomService.Stub.*asInterface*(ServiceManager.*getService*(Context.***TELECOM\_SERVICE***));  
}

/\*\*  
 \* **@hide** \*/  
**private** IPhoneSubInfo getSubscriberInfo() {  
 *// get it each time because that process crashes a lot* **return** IPhoneSubInfo.Stub.*asInterface*(ServiceManager.*getService*("iphonesubinfo"));  
 }

总结：

系统级的service仅仅起了一个router的功能，它将对应的请求通过系统中的ServiceManager中的service去真正的请求对应的服务。