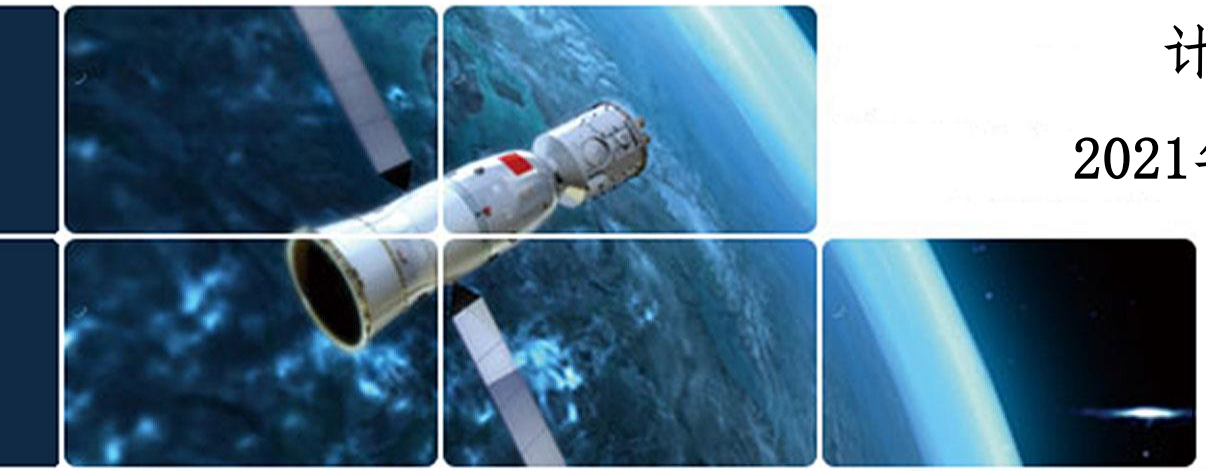
第9讲 移动互联网终端应用服务



计算学部

2021年12月14日

案例中的服务问题

应用可以不被
系统杀死吗?

应用不启动能时
能感知数据的实时变化吗?

Q1：能确保系统不被杀死吗?

A1：能，但需要孪生的服务或应用实现进程守护。

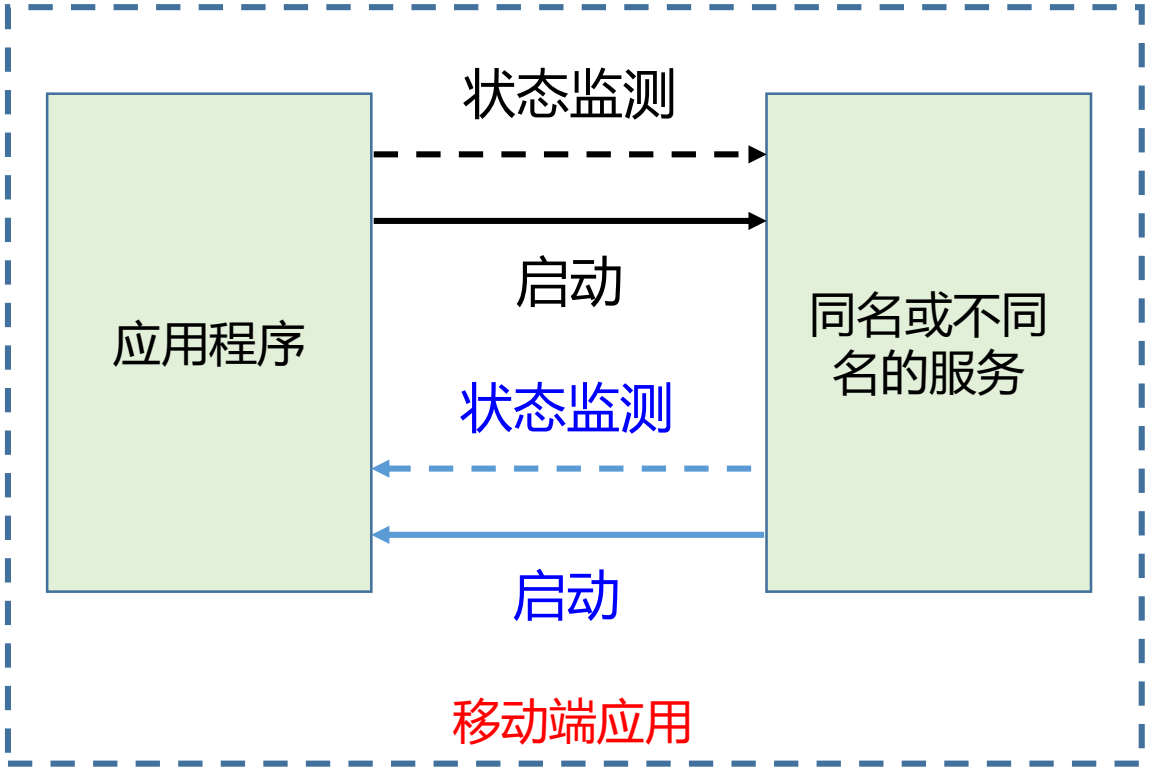
Q2：新数据来了能通知我吗?

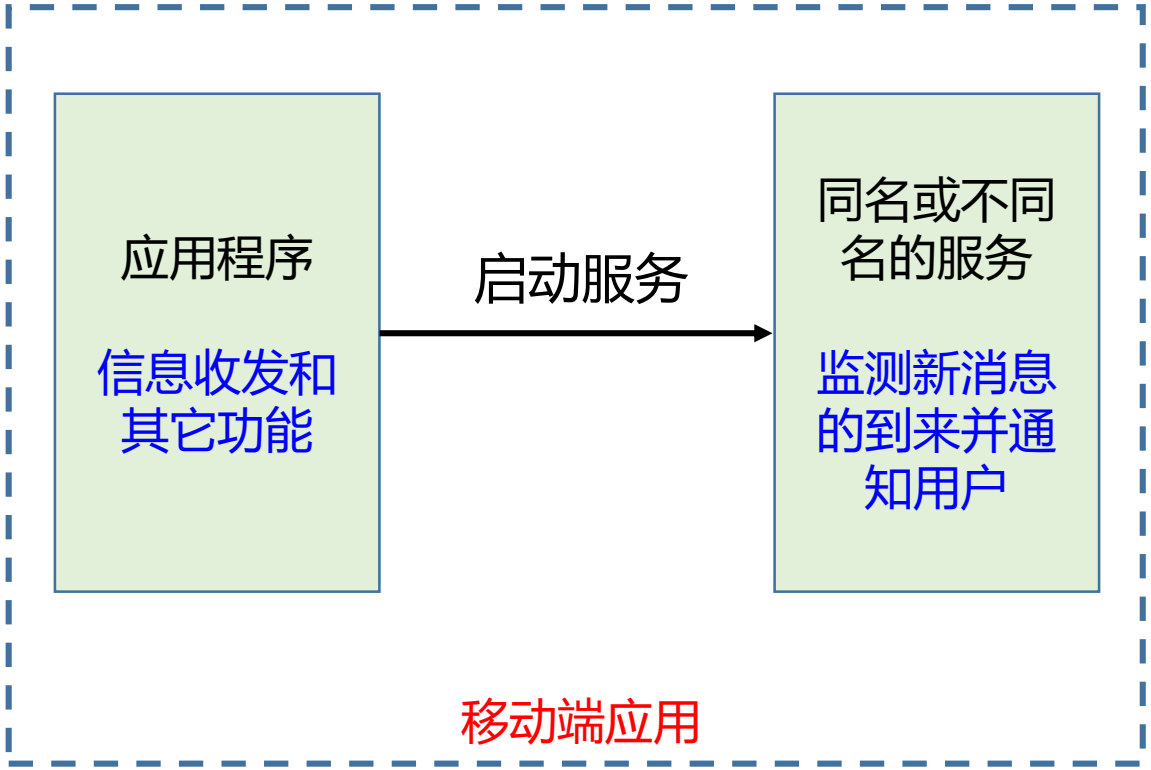
A2：能，但需要实时监测数据的变化，服务能实现。

Q3：能让哈工大APP显示数据吗?

A3：能，但需要跨进程服务。

服务的应用场景


状态监测
启动
应用程序
同名或不同名的服务
状态监测
启动
移动端应用


应用程序
信息收发和其它功能
启动服务
同名或不同名的服务
监测新消息的到来并通知用户
移动端应用

- ***Android Service***
- ***Harmony Service***

Service概念

- ***Service***适用于开发没有用户界面且长时间在后台运行的应用功能
- 因为手机硬件性能和屏幕尺寸的限制，系统仅允许一个应用程序处于激活状态并显示在手机屏幕上，而暂停其它处于未激活状态的程序
- 因此需要一种后台服务机制，允许在没有用户界面的情况下，使程序能够长时间在后台运行，实现应用程序的后台服务功能，并能够处理事件或数据更新， ***Service***组件就是能实现这一功能的应用程序组件

Service概念

- ***Service***组件通常不直接与用户进行交互，能够长期在后台运行
- 在实际的应用中，***QQ***、微信等很多应用都需要使用***Service***；还有***MP3***播放器，关闭播放器界面后仍能够保持音乐持续播放，需要在***Service***组件中实现音乐播放功能
- 随着应用类型和数量增多，***Service***的应用越来越广，大多数***Service***充当应用的消息拦截器（接收器）或守护者

Service的特点和功能

- ***Service***适用于无需用户干预，按照一定规则运行或长期运行的后台功能
- 因为***Service***没有用户界面，更加有利于降低系统资源的消耗，***Service***比***Activity***具有更高的系统优先级，因此在系统资源紧张时， ***Service***不会被系统优先终止
- 即使***Service***被系统终止，在系统资源恢复后***Service***也将自动恢复运行状态，因此可以认为***Service***是在系统中永久运行的组件
- ***Service***组件除了实现后台服务功能，还可用于进程间通信

启动/停止Service

- ***Service***不能自己运行，需要通过某个***Activity***或者其他***Context***对象来调用。 ***Service***在后台运行，不产生用户交互
- 启动运行***Service***的方式：

  1）***startService()*** //启动者与服务不相关或弱相关

  2）***bindService()*** //启动者与服务强相关

- 需要在*androidManifest.xml*中配置服务

```
<service android:enabled="true" android:name=".MyService" />
```

启动/停止Service

- 调用*Context.startService()*可启动*Service*

  调用*Context.stopService()*或*Service.stopSelf()*可停止*Service*

- *Service*一定是由其它组件启动的，但停止过程可以通过其它组件或自身完成

- 调用*Context.startService()*启动*Service*后，启动组件不能获取*Service*对象实例，因此无法调用*Service*的任何函数，也不能够获取到*Service*的任何状态和数据信息

- 以*Context.startService()*启动方式使用的*Service*，需要具备自管理的能力

Service的显式启动和隐式启动

- 显式启动（直接调用*Component*）

```
1.  final Intent serviceIntent = new Intent(this, RandomService.class);
2.  startService(serviceIntent);
```

- 隐式启动（设置和匹配*Action*）

```
1.  <service android:name=".RandomService">
2.          <intent-filter>
3.                  <action android:name="edu.cuc.RandomService" />
4.          </intent-filter>
5.  </service>
```

```
1.  final Intent serviceIntent = new Intent();
2.  serviceIntent.setAction("edu.cuc.RandomService");
```

绑定Service

- 绑定方式下，*Service*使用通过服务链接（*Connection*）实现，通过服务链接能获取*Service*的对象实例，因此绑定*Service*的组件可以调用*Service*的函数，或直接获取*Service*的状态和数据信息
- 通过*Context.bindService()*建立服务链接，自动启动服务

  通过*Context.unbindService()*停止服务链接
- 同一*Service*可以绑定多个服务链接，可以同时为多个不同的组件提供服务

Service启动方式和绑定方式结合

- 启动*Service*和绑定*Service*并不完全独立，支持混合使用
- 以*MP3*播放器为例：通过*Context.startService()*可启动播放，但在播放过程中如果用户需要暂停播放，则须通过*Context.bindService()*获取服务链接和服务对象实例，通过调用服务对象实例中的函数暂停播放过程，并保存相关信息
- 在这种情况下，调用*Context.stopService()*不能停止*Service*，所有服务链接关闭后，*Service*才能真正停止

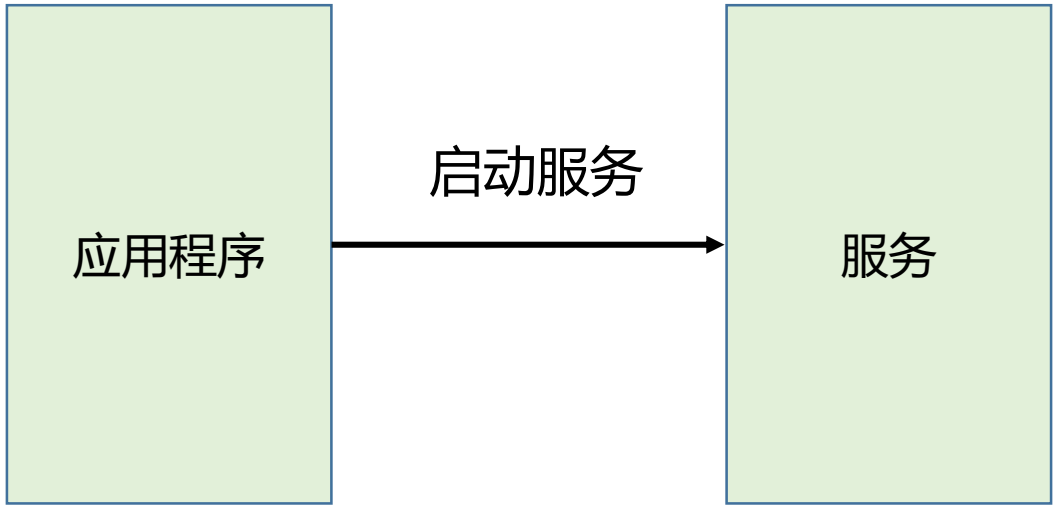
Android Service的生命周期方法

- *Service*包括若干生命周期方法
  - *onCreate()*用于创建*Service*实例
  - *onStartCommand()*用于启动*Service*
  - *onDestroy()*用于销毁*Service*

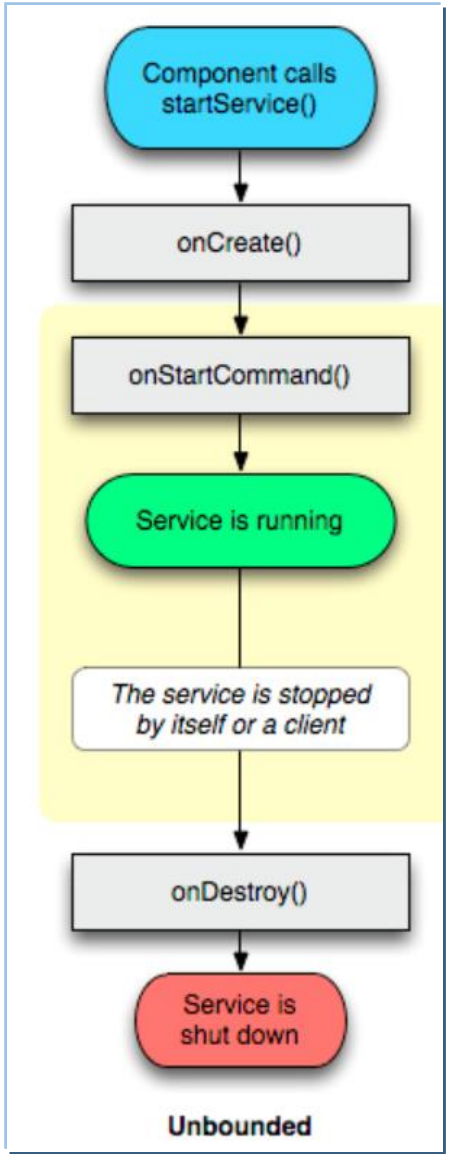
- 还有一些回调方法与*Service*生命周期相关
  - *onBind(Intent intent)*用于某进程和*Service*绑定
  - *onUnbind(Intent intent)*用于解除绑定
  - *onRebind(Intent intent)*用于*Service*重新绑定

Android Service的生命周期

- 若通过*Context.startService()*方式启动*Service*



应用和服务之间没有互相调用



Android Service的生命周期

Context.startService()—>onCreate()—>onStartCommand()—>Service running—>stopService() -->onDestroy()—>Service stop

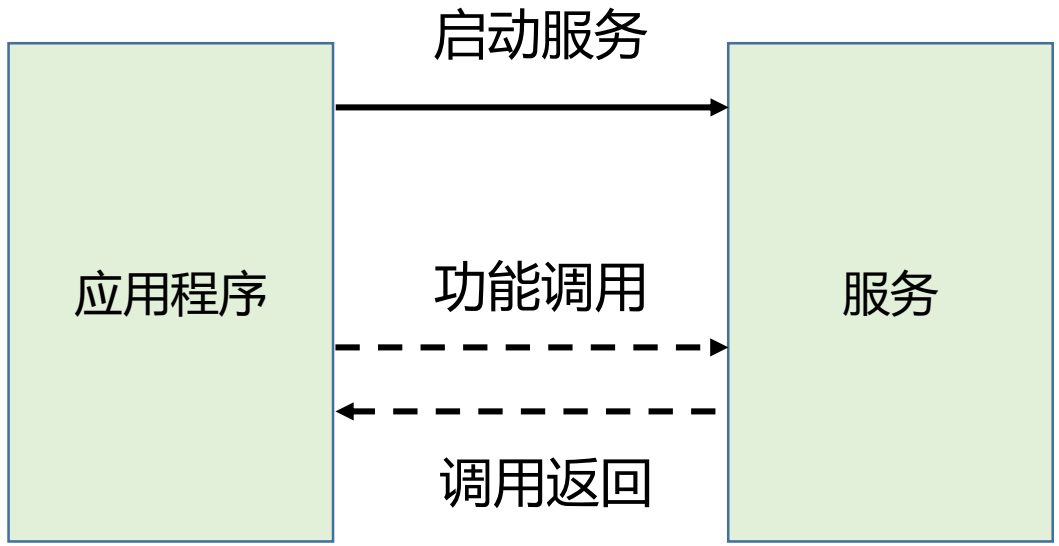
- **启动**：如果*Service*没有运行，先调用*onCreate()*，然后再调用*onStart()*启动*Service*；如果*Service*已经在运行了，只需调用*onStart()*启动*Service*，一个*Service*的*onStart()*方法可能会重复调用多次
- **关闭**：关闭*Service*需首先调用*stopService()*方法停止*Service*，然后再调用*onDestroy()*方法销毁*Service*

- *Service*的生命周期为

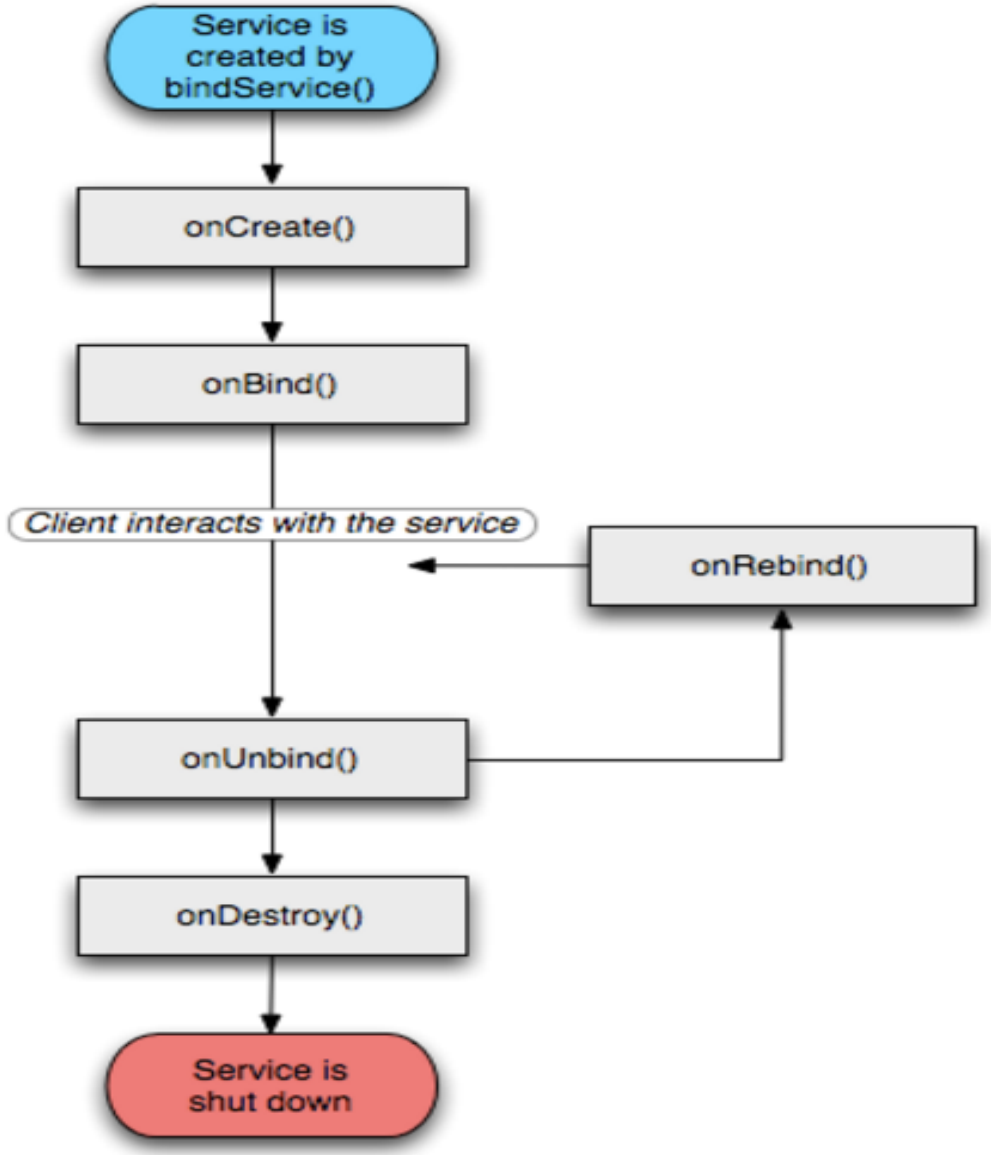
  onCreate()—>*onStart()*（可调用多次）—>*onDestroy()*

Android Service的生命周期

- 若通过*Context.bindService()*方式启动*Service*



应用和服务之间可以互相调用



Android Service的生命周期

Context.bindService()—>onCreate()—>onBind()—>Service running—> stopService()—>onUnbind()—>onDestroy()—> Service stop

- *onBind()*返回给客户端一个*IBind*实例，允许客户端回调服务方法，获取*Service*的运行状态或针对*Service*进行其他操作，可把调用者(*Context*或*Activity*)和*Service*绑定在一起，*Context*退出， *Service*也会调用*onUnbind()—> onDestroy()*退出

- *Service*生命周期为：*onCreate()—>onBind()*（只能绑定一次，不可多次绑定）—>*onUnbind()—>onDestroy()*

Service的运行机制

- ***Android***系统中，***Activity***、 ***Service***和***BroadcastReceiver***均工作在主线程，因此任何耗时的处理过程都会降低用户界面的响应速度，甚至导致用户界面失去响应
- 当用户界面长时间失去响应后， ***Android***系统会强行关闭应用程序
- 为了避免阻塞，或如果需要在***Service***中处理网络连接等耗时的操作，应该将这些任务放在单独的线程中进行处理，避免阻塞用户界面

使用线程运行Android Service

- 将耗时的处理过程转移到**子线程**，可缩短主线程的事件处理时间，从而避免用户界面长时间失去响应
- “**耗时处理过程**”一般是指复杂运算过程、大量的文件操作、存在延时的网络通信和数据库操作等
- 多线程（子线程）一般使用***Handler***更新用户界面， ***Handler***允许将***Runnable***对象发送到线程消息队列，每个***Handler***绑定到一个单独的线程和消息队列上，可通过***Post()***方法将***Runnable***对象从后台线程发送到***GUI***线程的消息队列

绑定Activity与Service

- 默认情况下，如果未调用*stopService()*方法停止服务，启动后的服务会随系统启动而启动，随系统关闭而关闭，服务一直在后台运行
- 如果希望启动服务的*Activity*关闭后服务自动关闭，需要将*Activity*与*Service*绑定：

  public boolean bindService(Intent service, ServiceConnection conn, int flags)
- 解除绑定关系可以调用如下方法进行

  unbindService(ServiceConnection conn);

Android开机启动Service

- 通过*Activity*启动*Service*有时并不符合实际应用需求，可能需要开机时启动*Service*
- *Broadcast Receiver* 可启动*Activity*，也可启动*Service*

```
public class StartupReceiver extends BroadcastReceiver{
    public void onReceive(Context context, Intent intent){   //接收系统广播
     //启动Service
     Intent serviceIntent = new Intent(context, MyService.class);
     context.startService(serviceIntent);
    }
}
```

Android开机启动Service

```
<application android:label="@string/app_name">

    <receiver android:name="StartupReceiver">
      <intent-filter>    //隐式调用
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>

    <service android:enabled="true" android:name=".MyService" />

</application>

<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

其它常用功能

- **判断服务是否已经注册**

  通过*PackageManager.queryIntentService*方法根据***IntentFilter***查询系统中某个或某组服务

- **判断服务是否已经启动**

  通过*ActivityManager.getRunningService*方法获得系统中正在运行的所有服务，然后查询指定的服务

Android进程间通信

- 在*Android*系统中，每个应用程序在各自的进程中运行，处于安全考虑，进程间彼此隔离，进程之间传递数据和对象，需要使用进程间通信机制（*IPC*）
- 在*Unix/Linux*系统中，传统的*IPC*机制包括共享内存、管道、消息队列和*Socket*等
- *Android*系统未使用传统*IPC*机制，采用*Intent*和远程服务方式实现*IPC*，应用程序具有更好的独立性

Android平台进程间通信

- *Android*允许应用程序使用*Intent*启动*Activity*和*Service*，还可传递数据，是一种简单、高效、易于使用的*IPC*机制
- *Android*另一种*IPC*机制是“**远程服务**”，服务和服务调用者在不同进程，调用过程需要跨越进程
- *Android*系统远程服务按照以下三步实现
  - 使用*AIDL*语言定义远程服务接口
  - 在*Service*类中实现*AIDL*中定义的方法
  - 需要调用远程服务的组件通过与*AIDL*相同的接口调用远程服务

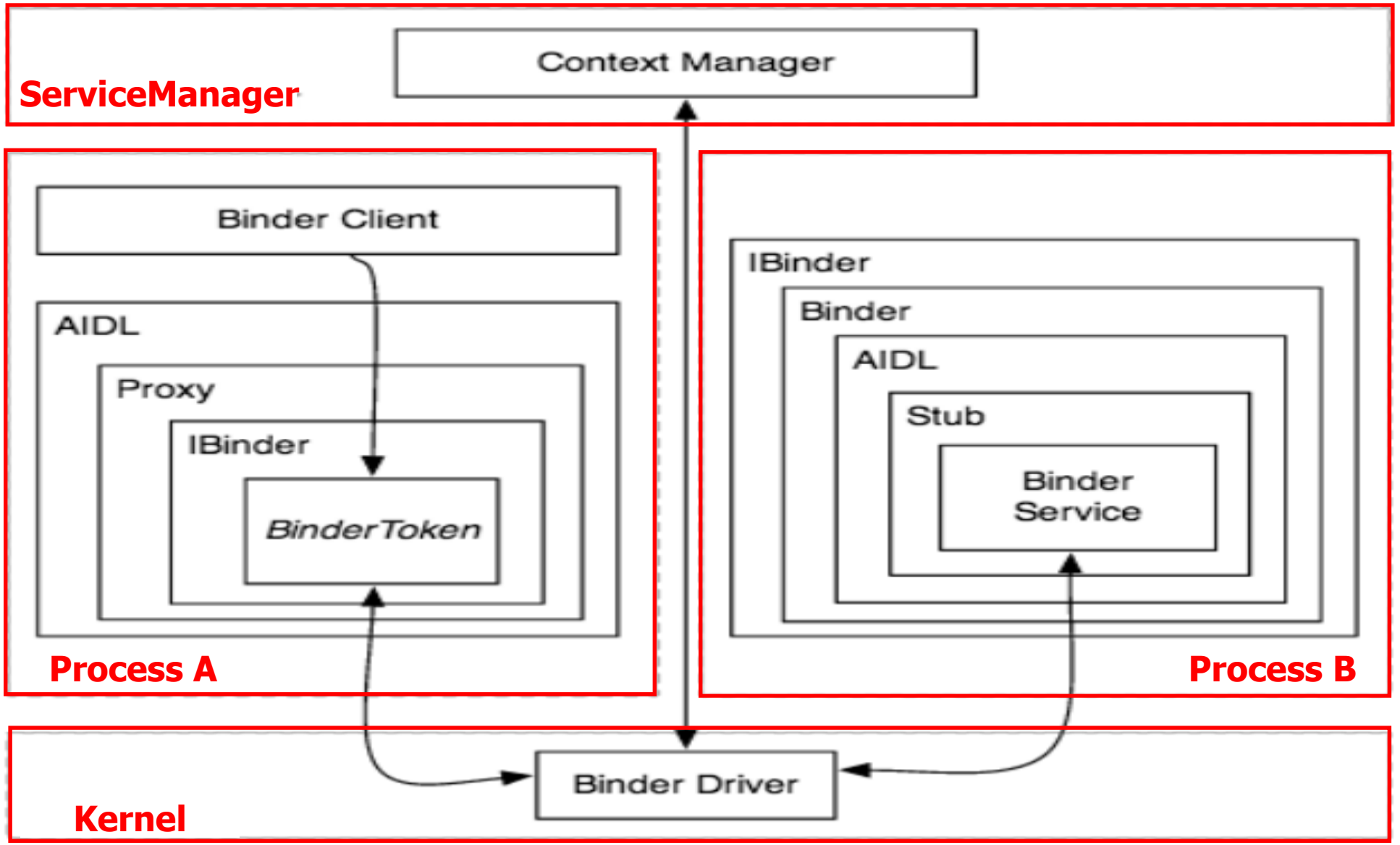
Android服务的创建与调用

- ***Android***系统进程之间不能直接访问相互的内存控件、为了使数据能在不同进程间传递，数据必须转换成能够穿越进程边界的系统级原语，完成边界穿越后，还需要转换回原有格式

- ***AIDL***（***Android Interface Definition Language***）是***Android***自定义接口描述语言，可以简化进程间数据格式转换和数据交换的代码，通过定义***Service***内部公共方法，允许处于不同进程间的调用者和***Service***之间传递数据

- *AIDL*的语法与*Java*语言的接口定义非常相似，唯一不同之处在于，*AIDL*允许定义函数参数的传递方向
- *AIDL*支持三种方向：*in*、*out*和*inout*
  - 标识为*in*的参数从调用者传递到远程服务中（默认）
  - 标识为*out*的参数从远程服务传递到调用者
  - 标识为*inout*的参数先从调用者传递到远程服务，再从远程服务返回到调用者

AIDL的工作流程


ServiceManager
Context Manager
Binder Client
AIDL
Proxy
IBinder
BinderToken
Process A
IBinder
Binder
AIDL
Stub
Binder Service
Process B
Binder Driver
Kernel

（1）建立*aidl*文件，定义接口

```
//IMyService.aidl
package mobile.android.ch12.aidl;
interface IMyService
{
    String getValue();                  //定义远程方法1：getValue()
    int add(in int op1,in int op2);     //定义远程方法2：add(in int op1,in int op2)
}
```

注意：*aidl*方法不能加修饰符（例如*public*、*private*等）

不能包含*AIDL*不支持的数据类型（如*InputStream*和*OutputStream*等）

建立AIDL服务

(2) 根据*aidl*文件自动生成*java*接口文件，注意：用户不需要维护该文件的具体内容

(3) 建立服务，实现*aidl*文件生成的*java*接口

```
public class MyService extends Service{
  public class MyServiceImpl extends IMyService.Stub{ //Stub是自动生成的
    @Override
    public String getValue() throws RemoteException{
       return "我喜欢开发";}
   }
  @Override
  public IBinder onBind(Intent intent){
   return new MyServiceImpl();
   //返回MyServiceImpl对象，否则客户端无法获得服务对象
  }
}
```

建立AIDL服务

（4）配置*AIDL*服务，需要注意的是，*<action>*标签*android:name*值代表客户端要引用的服务*ID*

```
<service android:name=".MyService" >
    <intent-filter>
        <action android:name="mobile.android.ch12.aidl.IMyService" />
    </intent-filter>
</service>
```

调用AIDL服务(客户端)

- 首先使用指定***AIDL***服务的ID绑定***AIDL***服务
- 创建***ServiceConnection***对象时如果绑定成功，系统会调用***onServiceConnected***方法，获得***AIDL***服务对象

```
private ServiceConnection serviceConnection = new ServiceConnection(){
  @Override
   public void onServiceConnected(ComponentName name, IBinder service){
      myService = IMyService.Stub.asInterface(service);
      btnInvokeAIDLService.setEnabled(true);
   }
  @Override
  public void onServiceDisconnected(ComponentName name){
    // TODO Auto-generated method stub
   }
};
```

- 除了自己创建服务类之外，开发者还可以使用系统提供的服务类，比如：
  - 来电服务类
  - 短信服务类
  - 网络服务类
  - 壁纸服务类
  - 电源服务类等
- 获得系统服务方法：

Context.getSystemService(Context.TELEPHONY_SERVICE);

- Android Service
- Harmony Service

- ***Harmony Service***是基于***Service***模板的***Ability***

- ***Android Service***是一种可在后台执行长时间运行操作而不提供界面的应用组件，可由其他组件启动，即使用户切换到其他应用，服务仍在后台继续运行。此外，组件可通过绑定服务与之进行交互，可执行进程间通信 (***IPC***)

- ***Harmony Service***主要用于后台运行任务（如执行音乐播放、文件下载等），不提供用户交互界面。***Service***由其他应用或***Ability***启动，即使用户切换到其他应用，***Service***仍在后台继续运行

- 两种系统的***Service***在定义、功能、使用方面基本相同。大部分用于后台运行任务，不提供界面，可以跨应用，也可以跨线程，都主要用于执行音乐播放、文件下载等任务，都可以分为本地服务和远程服务。***Service***是单例，可以被创建和销毁，共用***Service***必须开启多线程，否则会出现线程阻塞

创建Harmony Service

*Harmony Service*生命周期方法	*Android Service* 生命周期方法
onStart()	*onCreate()*
onCommand()	*onStartCommand()*
onConnect()	*onBind()*
onDisconnect()	*onUnbind()*
onStop()	*onDestroy()*

启动Harmony Service

- *Service*是*Ability*的一种，可通过*Intent*启动*Service*。支持启动本地*Service*和远程*Service*

```
Intent intent = new Intent();
Operation operation = new Intent.OperationBuilder()
        .withDeviceId("")
        .withBundleName("com.hit.music")
        .withAbilityName("com.hit.music.entry.ServiceAbility")
        .build();
intent.setOperation(operation);
startAbility(intent);
```

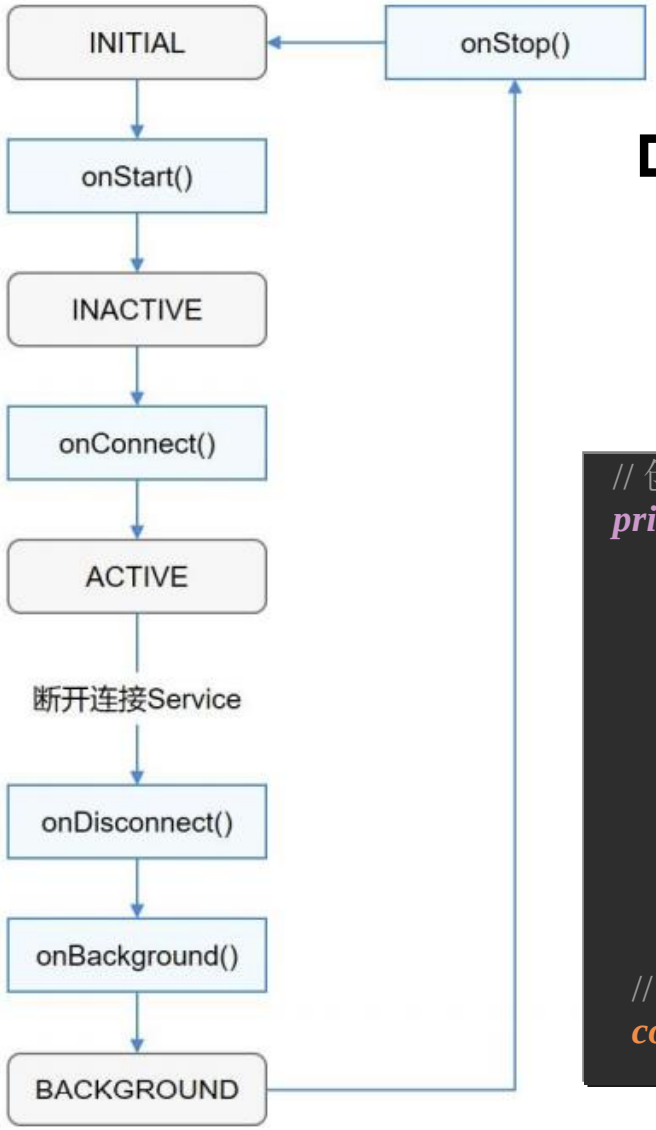
启动本地服务

```
Operation operation = new Intent.OperationBuilder()
        .withDeviceId("deviceId")
        .withBundleName("com.hit.music")
        .withAbilityName("com.hit.music.entry.ServiceAbility")
        // 设置支持分布式调度系统多设备启动的标识
        .withFlags(Intent.FLAG_ABILITYSLICE_MULTI_DEVICE)
        .build();
Intent intent = new Intent();
intent.setOperation(operation);
startAbility(intent);
```

启动远程服务

- *Service*会一直保持在后台运行，除非必须回收资源，否则不会停止或销毁。*Service*中可通过*terminateAbility()*停止本*Service*或在其他*Ability*调用*stopAbility()*停止 *Service*

连接Harmony Service



- 如果*Service*需要与*Page Ability*或其他应用的*Service*交互，应创建用于连接的*Connection*。*Service*支持通过*connectAbility()*方法与其他*Ability*连接

```
// 创建连接回调实例
private IAbilityConnection connection = new IAbilityConnection() {
    // 连接到Service的回调
    @Override
    public void onAbilityConnectDone(ElementName elementName, IRemoteObject iRemoteObject, int resultCode) {
        //拿到服务端传过来IRemoteObject对象，从中解析出服务端传过来的信息
    }

    // 断开与连接的回调
    @Override
    public void onAbilityDisconnectDone(ElementName elementName, int resultCode) { }
    };
// 连接Service
connectAbility(intent, connection);
```

前台 Service

- ***Service***通常运行在后台，后台***Service***优先级较低，当资源不足时，系统有可能回收正在运行的后台***Service***

- 在某些特殊场景下（如播放音乐），用户希望应用能够一直保持运行，可使用前台***Service***，前台***Service***始终保持运行

- 前台***Service***创建时须调用 ***keepBackgroundRunning()***，将***Service***与通知绑定。需声明***ohos.permission.KEEP_BACKGROUND_RUNNING***权限， 在***onStop()***方法中调用***cancelBackgroundRunning()***方法可停止前台***Service***

The End