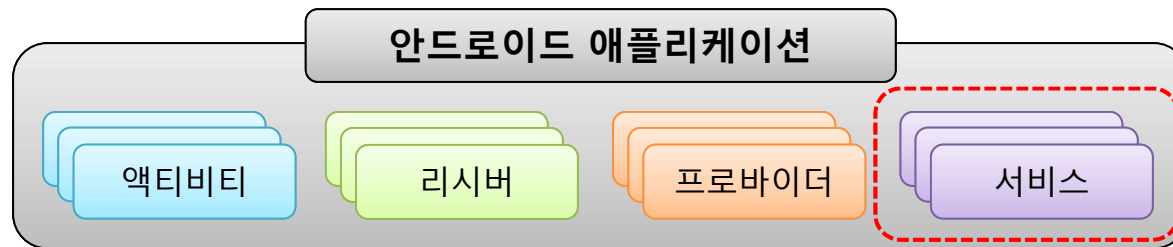


서비스

서비스

2



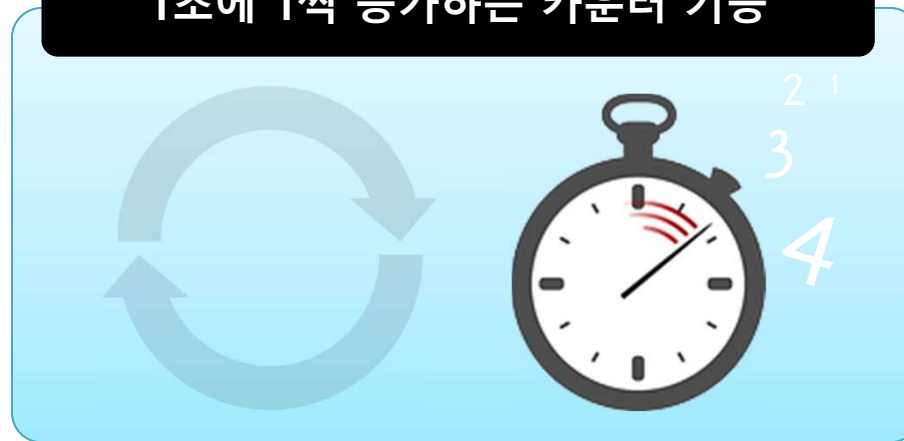
- 안드로이드 4대 컴포넌트 중 서비스Service를 배운다.
- 안드로이드를 개발하면서 아마도 가장 많이 사용되는 컴포넌트를 액티비티로 알고 있겠지만, 사실 시스템 측면에서는 서비스다.
안드로이드 시스템 자체가 대부분 서비스로 이뤄져 있고, 심지어 액티비티가 돌아가기 위해서도 서비스를 사용하기 때문이다
- 하지만 우리는 이러한 서비스가 존재하는 것을 전혀 인지하지 못한다. 왜일까?
- 바로 눈에 보이지 않고 백그라운드에서 동작하기 때문이다. 서비스는 액티비티와 달리 화면을 가지지 않고 오직 백그라운드에서만 동작한다.

서비스의 필요성

3

- 왜 컴포넌트 이름이 서비스일까? 이름 그대로 특정 기능을 누군가에게 제공하는 컴포넌트이기 때문이다. 이러한 특정 기능은 다양할 수 있고 서비스 컴포넌트는 그 기능을 담을 수 있도록 도와준다.
- 그렇다면 서비스의 필요성을 직접 느껴보자.
- 이를 위해 먼저 서비스 컴포넌트를 사용하지 초당 1씩 카운팅하는 기능을 구현해본다.

1초에 1씩 증가하는 카운터 기능



서비스의 필요성

4

- Application name: CountService
- Company Domain: company.co.kr
- Package name: kr.co.company.countservice

서비스의 필요성

5

□ activity_main.xml

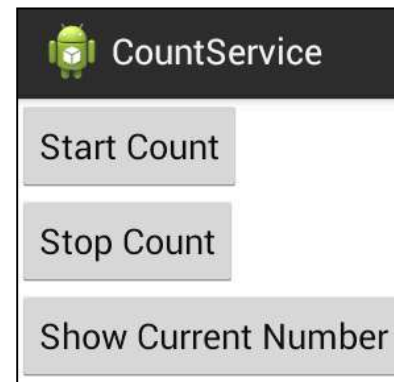
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
<Button android:id="@+id/start_count_btn"
    android:text="Start Count"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
<Button android:id="@+id/stop_count_btn"
    android:text="Stop Count"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
<Button android:id="@+id/show_cur_number_btn"
    android:text="Show Current Number"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
</LinearLayout>
```



● 레이아웃 구조

- LinearLayout
 - start_count_btn (Button) - "Start Count"
 - stop_count_btn (Button) - "Stop Count"
 - show_cur_number_btn (Button) - "Show Current Nu..."

서비스의 필요성

6

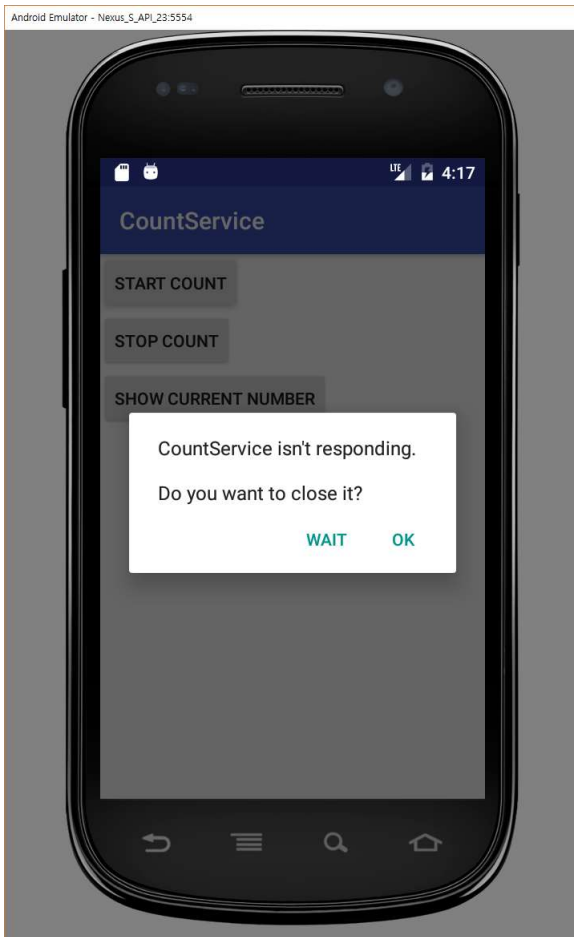
□ MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
  
    private int mCurNum = 0;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

서비스의 필요성

7

□ MainActivity.java



```
public void onClick(View v) {
    switch(v.getId()) {
        // 1. 카운트 시작
        // =====
        case R.id.start_count_btn:
            while(mCurNum <= 10000) {
                mCurNum ++;

                try { Thread.sleep(1000); }
                catch (InterruptedException e) {}
            }
            break;
        // =====
        // 2. 카운트 종료
        // =====
        case R.id.stop_count_btn:
            mCurNum = 0;
            break;
        // =====
        // 3. 현재까지 카운트 된 수치 보기
        // =====
        case R.id.show_cur_number_btn:
            Toast.makeText(this,
                "Current Number : " + mCurNum,
                Toast.LENGTH_LONG).show();
            break;
        // =====
    }
}
```

서비스의 필요성

8

■ 이 문제를 해결하려면 당연히 작업 스레드를 사용해야 한다.

□ MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
  
    private int mCurNum = 0;  
    private Thread mCountThread = null;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```


서비스의 필요성

9

□ MainActivity.java

```
public void onClick(View v) {  
    switch(v.getId()) {  
        // 1. 카운트 시작  
        // =====  
        case R.id.start_count_btn:  
            if (mCountThread == null) {  
                mCountThread = new Thread("Count Thread") {  
                    public void run() {  
                        while(mCurNum <= 10000) {  
                            mCurNum ++;  
  
                            try { Thread.sleep(1000); }  
                            catch (InterruptedException e) { break; }  
                        }  
                    }  
                };  
            }  
  
            mCountThread.start();  
        }  
  
        break;  
        // =====  
    }  
}
```

서비스의 필요성

10

- MainActivity.java
 - ▣ onClick 메소드 계속

// 2. 카운트 종료

// =====

```
case R.id.stop_count_btn:
    if(mCountThread != null) {
        mCountThread.interrupt();
        mCountThread = null;
        mCurNum = 0;
    }
```

break;

// =====

// 3. 현재까지 카운트 된 수치 보기

// =====

```
case R.id.show_cur_number_btn:
    Toast.makeText(this,
        "Current Number : " + mCurNum,
        Toast.LENGTH_LONG).show();
```

break;

// =====

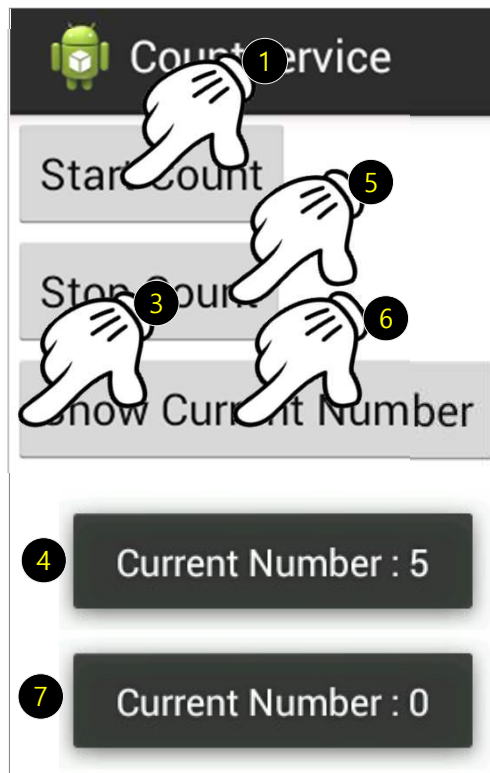
}

}

}

서비스의 필요성

11



● DDMS에서 스레드 확인

File Explorer Threads Heap Allocation Tracker					
ID	Tid	Status	u...	s...	Name
1	2340	Native	94	33	main
*2	2343	VmWait	0	0	GC
*3	2346	VmWait	0	0	Signal Catcher
*4	2347	Runnable	0	5	JDWP
*5	2348	VmWait	6	5	Compiler
*6	2349	Wait	0	0	ReferenceQueueDaemon
*7	2350	Wait	0	0	FinalizerDaemon
*8	2351	Wait	0	0	FinalizerWatchdogDaemon
9	2352	Native	1	0	Binder_1
10	2353	Native	1	0	Binder_2
11	3616	TimedWait	1	0	Count Thread

서비스의 필요성

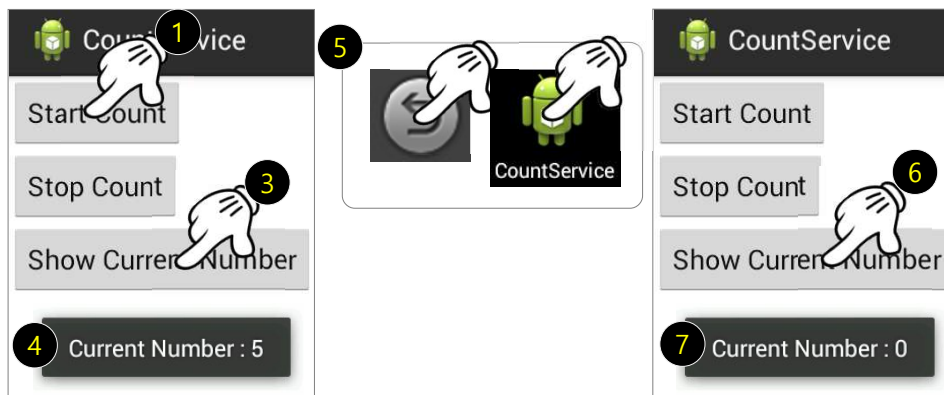
12

■ 하지만 액티비티가 종료되더라도 카운트 기능이 계속 동작해야 하는 경우 문제가 있다.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
    private int mCurNum = 0;  
    private Thread mCountThread = null;  
    ...  
}
```

모든 컴포넌트는 생명주기를 다하면
컴포넌트 객체 자체도 소멸된다.



● DDMS에서 스레드 확인

ID	Tid	Status	u...	s...	Name
1	8395	Native	41	27	main
8 11	9497	TimedWait	0	0	Count Thread 2

하지만 DDMS를 통해 스레드를 확인하면 여전히 Count Thread는 동작 중이다.

그 이유는 스레드를 중단한 적이 없기 때문이다.

심지어 멤버변수가 초기화 되었기 때문에 스레드를 제어할 방법조차 없게 되었다.

서비스의 필요성

13

□ 컴포넌트 생명주기 문제

■ 이 문제를 해결하기 위해서는 서비스가 필요하다.

서비스는 다른 컴포넌트에 의존하지 않고 백그라운드에서 독립된 동작을 수행한다.

또한 각종 컴포넌트에서 서비스를 제어하거나 원하는 정보를 참조할 수도 있다

서비스의 필요성

14

□ 외부의 다른 앱 프로세스가 사용할 수 없는 카운트 기능

■ 서비스의 필요성은 이뿐만이 아니다.

■ 카운트 기능을 내부에서만 사용하지 않고 다른 외부 앱에서도 사용하고 싶을 때가 있을 것이다.
예를 들어 현재까지 카운트된 값을 다른 앱에서 참조하는 경우다.

■ 하지만 안드로이드 시스템은 서로 다른 프로세스 간에 메모리의 접근을 허용하지 않는다.
그렇다면 결코 카운트 기능을 다른 앱에게 공유할 수 없단 말인가!

■ 지금까지 작업 스레드로 구성된 예제에서는 그렇지만 서비스 컴포넌트를 사용한다면 가능하다.

■ 서비스는 프로세스 간 메모리를 공유하거나 원하는 함수를 호출할 수 있도록 하는
IPCInterprocess Communication, **RPC**Remote Procedure Call를 지원하기 때문이다.

스타티드 서비스와 바운드 서비스

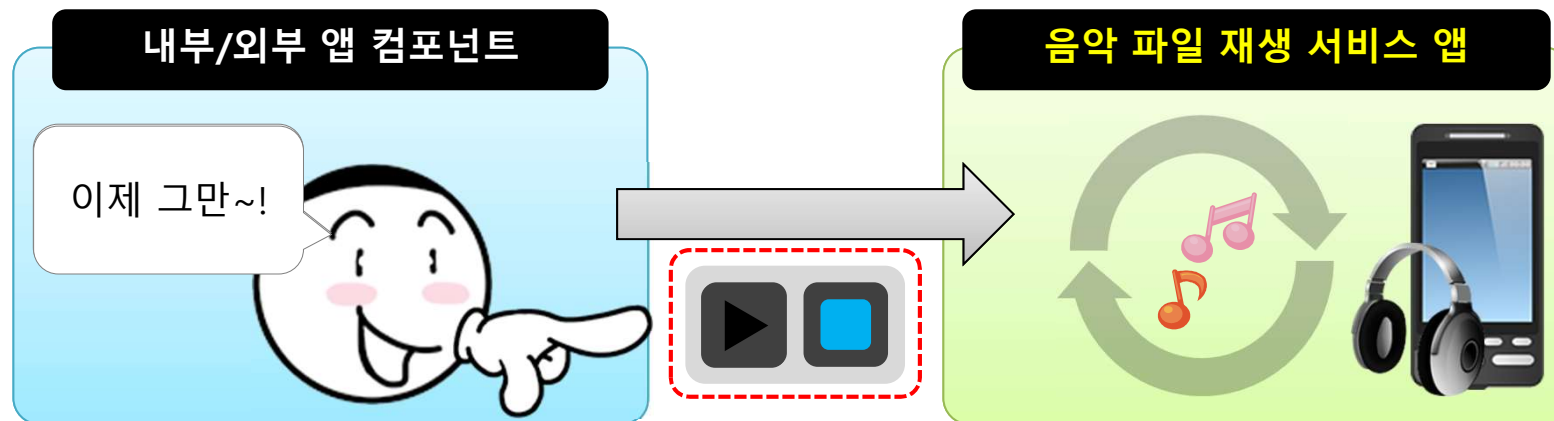
15

- 서비스는 기본적으로 스타티드 Started와 바운드 Bound 형태를 가진다.
이 두 가지 형태는 구현할 서비스의 목적에 따라 선택적으로 사용하거나
모두 활용하여 사용될 수도 있다.
- 하나씩 살펴보도록 하자.

스타티드 서비스와 바운드 서비스 – 스타티드 서비스

16

- 스타티드는 특정 서비스를 **동작시켜두는 데 목적을 둔 형태의 서비스**다.
- 따라서 서비스를 동작시킨 컴포넌트는 서비스를 중단하는 것 이외에 어떤 제어도 할 수 없다.
- 그리고 동작된 서비스는 알아서 백그라운드로 작업을 수행하고 종료된다.

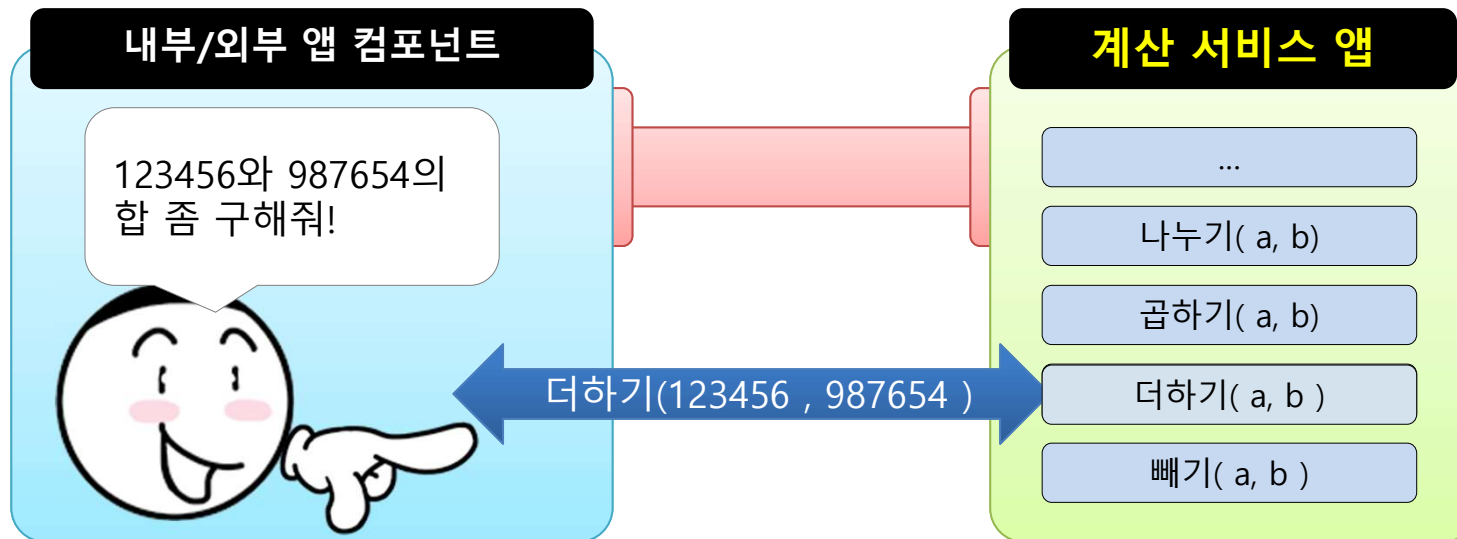


- 참고로 내/외부 앱과 음악 파일 재생 서비스의 관계는 서로 **독립적**이라 서비스를 동작시킨 컴포넌트가 종료되더라도 서비스는 작업이 끝날 때까지 동작을 유지한다.

스타티드 서비스와 바운드 서비스 – 바운드 서비스

17

- 바운드는 외부 라이브러리를 사용하는 것과 매우 유사하다.



- 서비스로 연결될 컴포넌트는 서비스에 존재하는 함수들을 마치 라이브러리를 가져다 사용하듯이 쓸 수 있다.
- 참고로 내/외부 앱과 계산 서비스의 관계는 서로 **의존적**이라 서비스를 요청한 컴포넌트가 종료되면 서비스 연결도 끊어진다.
반대로 서비스는 내/외부 컴포넌트가 연결을 끊기 전까지 요청한 작업이 완료되었더라도 연결은 유지한다.

스타티드 서비스와 바운드 서비스 - 스타티드와 바운드의 조합 형태 서비스

18

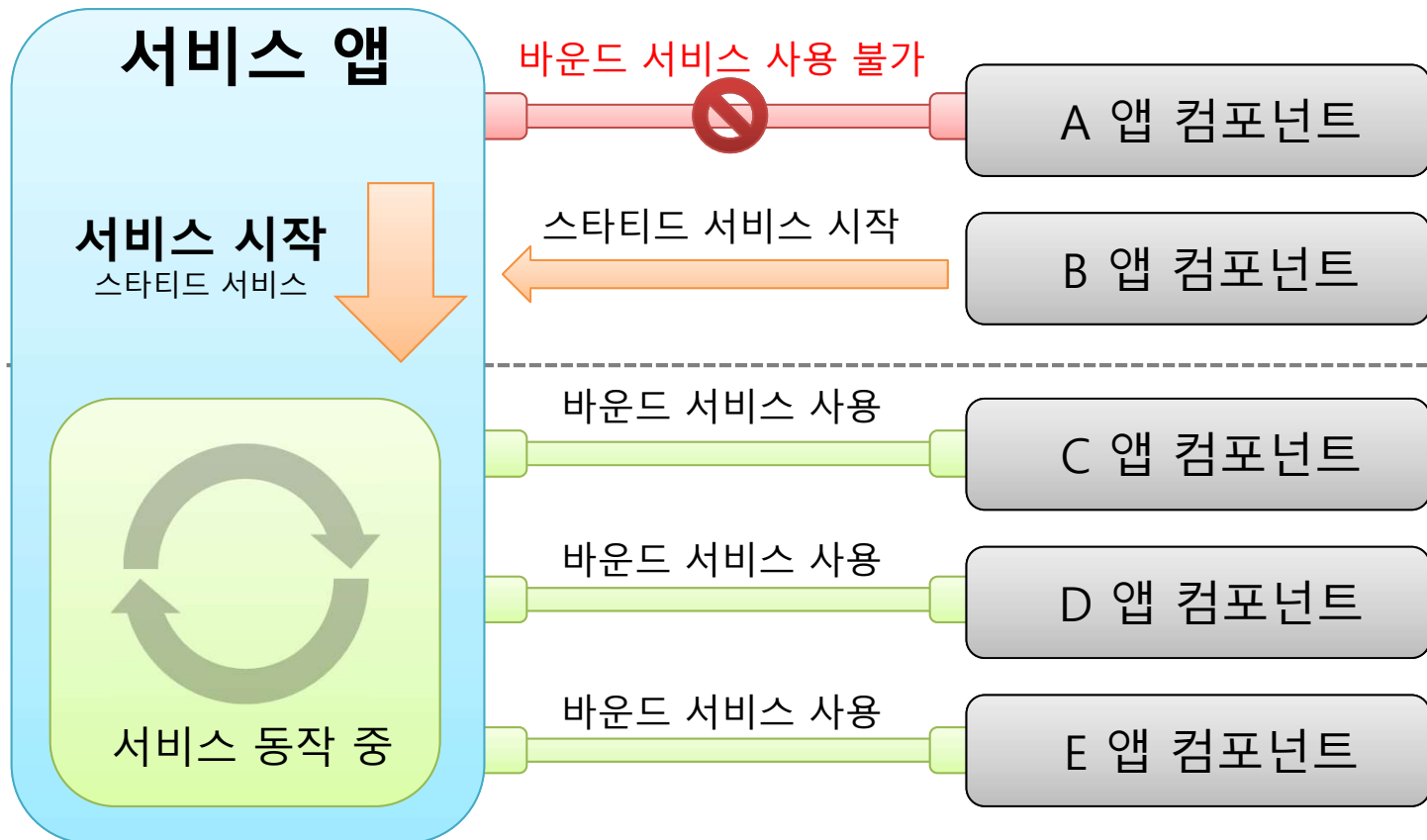
- 지금까지 두 가지 형태 서비스를 분리해서 설명했지만,
구현할 앱에 따라 각 형태의 특징을 모두 활용해야 하는 경우가 있다.



스타티드 서비스와 바운드 서비스 - 왜 스타티드와 바운드 형태로 서비스가 구분되었을까?

19

- 스타티드와 바운드 형태로 서비스가 구분된 이유를 살펴보기 전에
먼저 둘의 기본 관계를 이해할 필요가 있다.



스타티드 서비스와 바운드 서비스 – 왜 스타티드와 바운드 형태로 서비스가 구분되었을까?

20

- 바운드는 스타티드 형태의 서비스가 실행되어야만 사용할 수 있는데,
그 이유는 음악 재생기에 비유하면 이해하기 쉽다.

음악을 재생하기 전에 일시 정지나 다음 곡 재생 등의 기능이 의미가 없지 않은가!

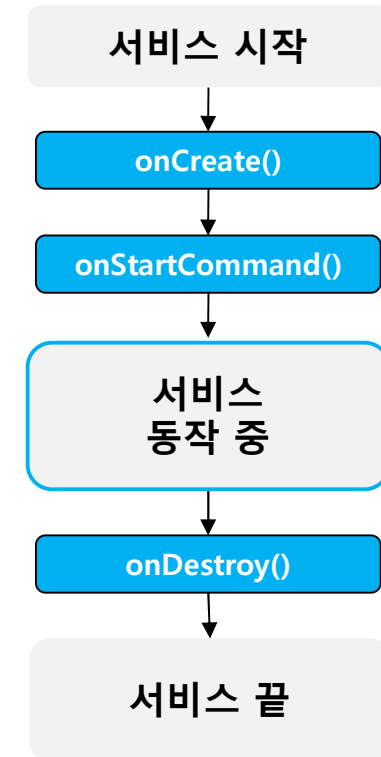
- 여기까지 생각해보면 바운드는 스타티드에 의존적이라 둘의 구별 없이 그냥 서비스라 보면 될 것 같다.
그리고 이 연관 관계가 안드로이드 서비스의 기본이다.
- 하지만 서비스를 사용하다 보면 서비스의 실행과는 상관없이 바운드 형태만 필요할 때가 많다.
예를 들면 앞서 설명된 계산 서비스 같은 경우다.
- 이를 위해 안드로이드는 스타티드와 바운드의 관계를 끊고 각각 독립적으로 사용할 수 있는
방법을 제공한다. 상세한 내용은 서비스를 알아가는 과정에서 차차 설명된다.

스타티드 서비스 - 서비스 클래스 만들기

21

□ CountService.java

```
public class CountService extends Service {  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Log.i("superdroid", "onCreate()");  
    }  
  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        super.onStartCommand(intent, flags, startId);  
        Log.i("superdroid", "onStartCommand()");  
        return START_STICKY;  
    }  
  
    @Override  
    public void onDestroy() {  
        Log.i("superdroid", "onDestroy()");  
        super.onDestroy();  
    }  
  
    @Nullable  
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
}
```



스타티드 서비스 – AndroidManifest.xml에 서비스 등록하기

22

□ AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.co.company.countservice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".CountService" />
    </application>

</manifest>
```

스타티드 서비스 – startService 함수로 스타티드 서비스 실행하기

23

□ MainActivity.java

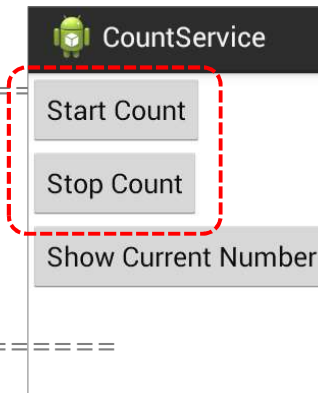
```
public class MainActivity extends AppCompatActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

스타티드 서비스 – startService 함수로 스타티드 서비스 실행하기

24

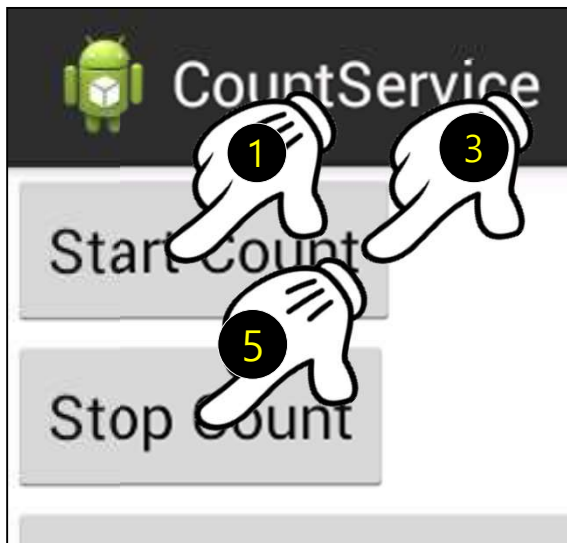
□ MainActivity.java

```
public void onClick(View v) {  
    switch(v.getId()) {  
        // 1. 카운트 시작  
        // =====  
        case R.id.start_count_btn: {  
            Intent serviceIntent = new Intent(this, CountService.class);  
  
            startService(serviceIntent);  
  
            break;  
        }  
        // =====  
  
        // 2. 카운트 종료  
        // =====  
        case R.id.stop_count_btn: {  
            Intent serviceIntent = new Intent(this, CountService.class);  
  
            stopService(serviceIntent);  
            break;  
        }  
        // =====  
  
        // 3. 현재까지 카운트 된 수치 보기  
        // =====  
        case R.id.show_cur_number_btn:  
            break;  
        // =====  
    }  
}
```



스타티드 서비스 – startService 함수로 스타티드 서비스 실행하기

25



● 로그 확인

2

```
onCreate()  
onStartCommand()
```

4

```
onStartCommand()
```

6

```
onDestroy()
```

서비스 시작

onCreate()

onStartCommand()

서비스
동작 중

onDestroy()

서비스 끝

스타티드 서비스 -

메인 스레드에서 동작하는 스타티드 서비스의 생명주기 함수

26

■ 이제 서비스에 초당 1씩 증가하는 서비스 기능을 추가해보자.

□ CountService.java

```
public class CountService extends Service {
```

```
    private int mCurNum = 0;
```

```
    ...
```

```
    @Override
```

```
    public int onStartCommand(Intent intent, int flags, int startId) {
```

```
        super.onStartCommand(intent, flags, startId);
```

```
        Log.i("superdroid", "onStartCommand()");
```

```
        while (true) {
```

```
            Log.i("superdroid", "Count: " + mCurNum);
```

```
            mCurNum++;
```

```
            try { Thread.sleep(1000); }
```

```
            catch (InterruptedException e) { break; }
```

```
        }
```

```
        return START_STICKY;
```

```
    }
```

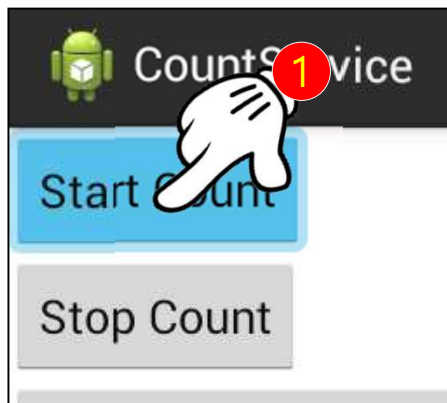
```
    ...
```

```
}
```

스타티드 서비스 -

메인 스레드에서 동작하는 스타티드 서비스의 생명주기 함수

27



PID	TID	Application	Tag	Text
7304	7304	kr.co.company.countservice	superdroid	Count: 17
7304	7304	kr.co.company.countservice	superdroid	Count: 18
7304	7304	kr.co.company.countservice	superdroid	Count: 19
1590	1604	system_process	ActivityMa...	Timeout executing service: ervice/.CountService}
1590	1604	system_process	Process	Sending signal. PID: 7304 S
7304	7309	kr.co.company.countservice	art	Thread[2,tid=7309,WaitingIn

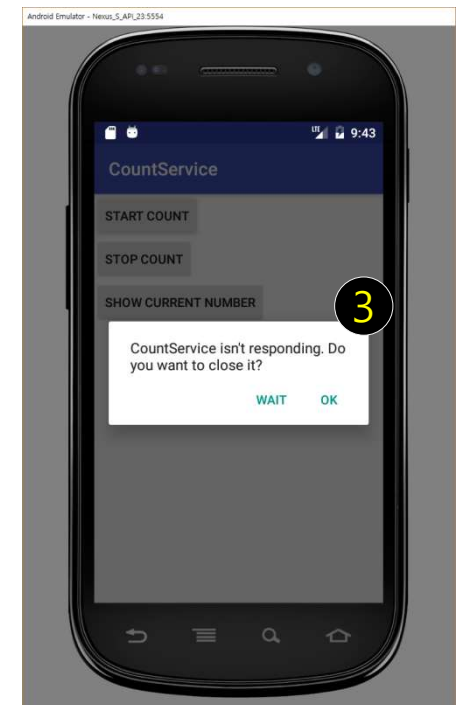
■ ANR이 발생되었다.

ANR은 메인 스레드에서 오랜 작업을 수행할 때 발생된다.

■ 정말 서비스가 메인 스레드에서 동작하는지 잠시 확인해보자.

PID	TID	Application	Tag	Text
7304	7304	kr.co.company.countservice	superdroid	onCreate()
7304	7304	kr.co.company.countservice	superdroid	onStartCommand()
7304	7304	kr.co.company.countservice	superdroid	Count: 0
7304	7304	kr.co.company.countservice	superdroid	Count: 1
7304	7304	kr.co.company.countservice	superdroid	Count: 2
7304	7304	kr.co.company.countservice	superdroid	Count: 3
7304	7304	kr.co.company.countservice	superdroid	Count: 4

■ 서비스도 메인 스레드에서 돌아간다. 잊지 말자!



스타티드 서비스 -

메인 스레드에서 동작하는 스타티드 서비스의 생명주기 함수

28

- 서비스 컴포넌트 역시 메인 스레드에서 동작하며, ANR에 걸리는 시간제한이 존재하며 서비스의 시간 제한은 20초다.

안드로이드 API 16까지의 프레임워크 소스 : `ActivityManagerService.java`

```
public final class ActivityManagerService extends ActivityManagerNative
    implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback
{
    ...
    // How long we wait for a service to finish executing.
    static final int SERVICE_TIMEOUT = 20*1000;
```

안드로이드 API 17부터의 프레임워크 소스 : `ActiveServices.java`

```
public class ActiveServices
{
    ...
    // How long we wait for a service to finish executing.
    static final int SERVICE_TIMEOUT = 20*1000;
```

- 액티비티, 리시버, 서비스 컴포넌트 모두 동일한 메인 스레드에서 동작하기 때문에 서비스가 오랜 작업으로 메인 스레드를 잡고 있으면 액티비티 대기할 수 밖에 없다. 참고로 액티비티는 5초, 리시버는 10초 혹은 1분, 그리고 서비스는 20초의 시간제한을 가진다.

스타티드 서비스 – 서비스에서 작업 스레드 사용하기

29

- 카운팅 동작을 작업 스레드에서 처리하도록 수정하자.

□ CountService.java

```
public class CountService extends Service {  
    ...  
    private Thread mCountThread = null;  
    ...  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        super.onStartCommand(intent, flags, startId);  
        Log.i("superdroid", "onStartCommand()");  
  
        if (mCountThread == null) {  
            mCountThread = new Thread("Count Thread") {  
                public void run() {  
                    while (true) {  
                        Log.i("superdroid", "Count: " + mCurNum);  
  
                        mCurNum++;  
  
                        try {  
                            Thread.sleep(1000);  
                        } catch (InterruptedException e) {  
                            break;  
                        }  
                    }  
                }  
            };  
            mCountThread.start();  
        }  
  
        return START_STICKY;  
    }  
}
```

스타티드 서비스 – 서비스에서 작업 스레드 사용하기

30

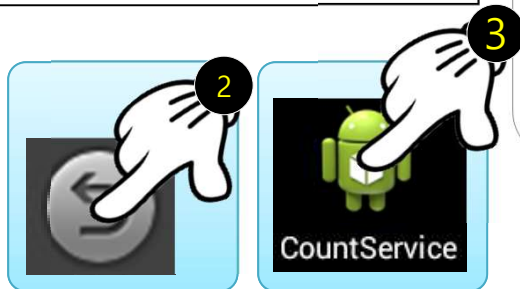
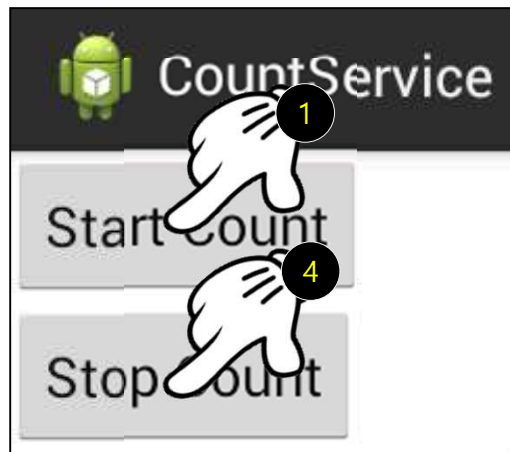
□ CountService.java

@Override

```
public void onDestroy() {  
    Log.i("superdroid", "onDestroy()");  
  
    if (mCountThread != null) {  
        mCountThread.interrupt();  
        mCountThread = null;  
        mCurNum = 0;  
    }  
  
    super.onDestroy();  
}
```

스타티드 서비스 – 서비스에서 작업 스레드 사용하기

31



● 로그 확인

L	Time	TID	Text
I	12-11 09:13:01.310	3498	onCreate()
I	12-11 09:13:01.310	3498	onStartCommand()
I	12-11 09:13:01.330	3576	Count : 0
I	12-11 09:13:02.376	3576	Count : 1
I	12-11 09:13:03.427	3576	Count : 2
I	12-11 09:13:04.464	3576	Count : 3
I	12-11 09:13:04.840	3498	onDestroy()

스타티드 서비스 –

스타티드 서비스의 생존 우선순위

32

- 백그라운드에서 동작 중인 서비스는 `stopService` 함수를 통해 의도적으로 종료한 경우를 제외하고 생존을 보장받을 수 있을까?
- 아쉽게도 그럴 수 없다.
- **안드로이드에서 가장 중요한 것은 시스템이다.** 만일 시스템이 사용해야 할 자원이 부족해지면 시스템은 멈춰 버리게 되고, 시스템 위에서 돌아가는 앱들도 사용할 수 없게 된다.
- 특히 자원 중에 메모리는 많은 앱이 실행되면 부족해질 수 밖에 없다.
따라서 메모리가 부족할 때 시스템은 생존을 위해 다른 앱의 프로세스들을 죽이고 메모리를 확보하게 된다.

스타티드 서비스 –

스타티드 서비스의 생존 우선순위

33

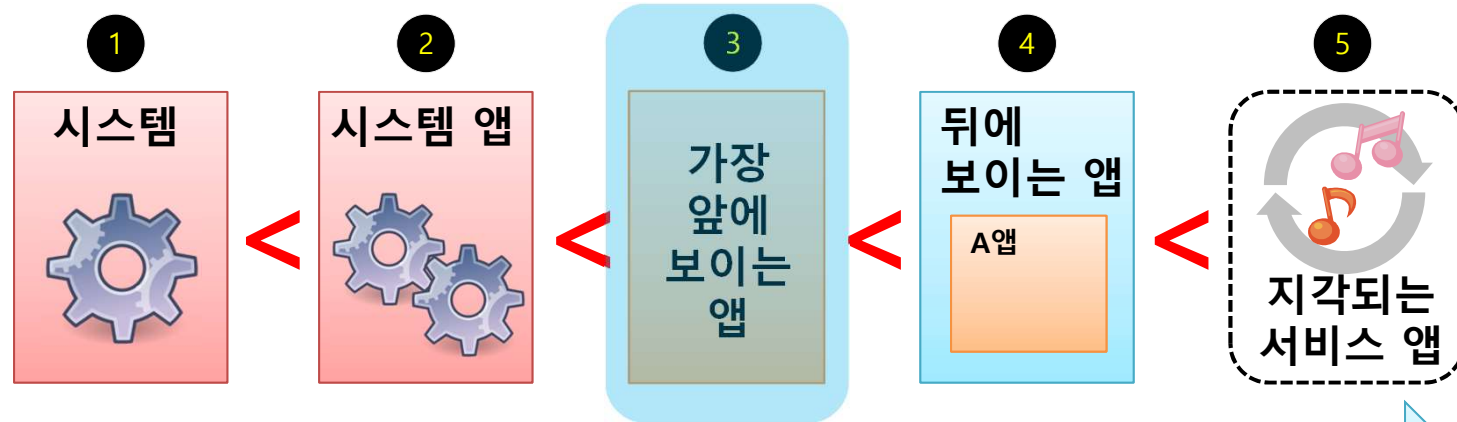
- 안드로이드에서 메모리가 부족할 때 다른 앱 프로세스를 죽이는 것을 LMK^{Low Memory Killer}라고 한다.
- LMK는 무작정 앱 프로세스를 강제로 종료하는 것은 아니다.
예를 들어 사용자가 열심히 게임을 즐기고 있는 상황에서 앱을 종료한다면
누가 불안하게 안드로이드 단말을 사용하겠는가!
- 따라서 LMK는 앱 상태에 따라 생존 우선순위를 매기고 가장 우선순위가 낮은 앱 프로세스부터
메모리가 확보될 때까지 죽여 나간다. 물론 우선순위가 낮은 앱은 당장 죽여도 큰 문제가 발생되지
않는 앱이다.

스타티드 서비스 -

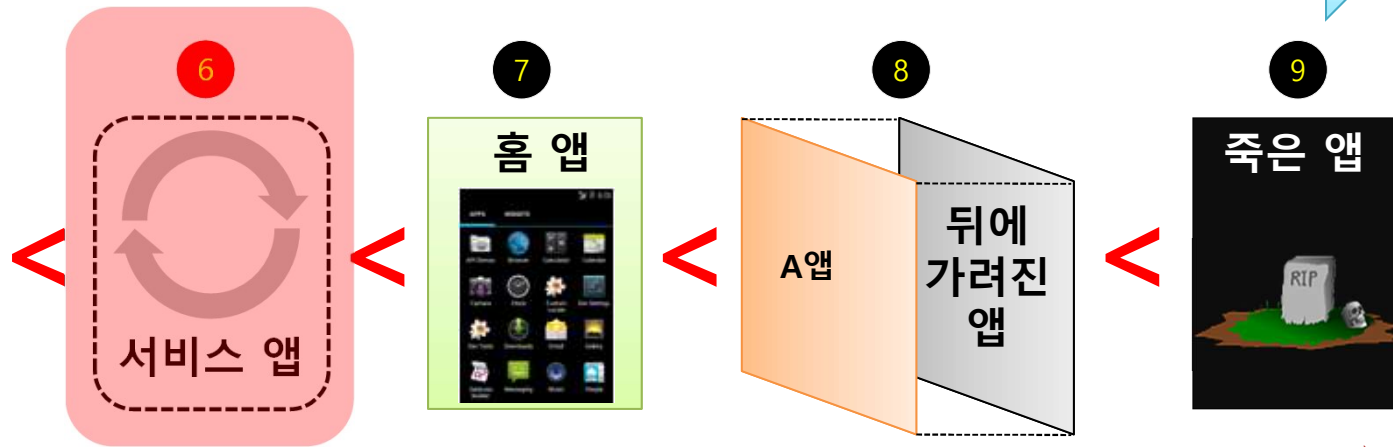
스타티드 서비스의 생존 우선순위

34

■ LMK 대상이 되는 앱의 우선순위를 살펴보자.



포그라운드 프로세스 상태 : 대부분 LMK의 정리 대상이 아니다.



백그라운드 프로세스 상태 : LMK에 의해 정리될 확률이 높다.

스타티드 서비스 –

스타티드 서비스의 생존 우선순위

35

■ 앱에 여러 상태가 존재하는 경우

예를 들어 나의 앱은 액티비티가 보이는 상태이면서,
동시에 서비스 컴포넌트도 동작 중이라면 어떤 우선순위일까?

이렇게 하나의 앱은 다양한 컴포넌트를 가질 수 있기 때문에
여러 우선순위 상태가 존재할 수 있다.

그러나 LMK가 앱을 바라보는 상태는 생존 우선순위가 가장 높은 것을 대상으로 한다.
그러므로 이 앱은 **액티비티가 보이는 상태**의 우선순위가 된다.

스타티드 서비스 –

스타티드 서비스의 생존 우선순위

36

그렇다고 백그라운드 상태인 일반 서비스가 무조건 위험한 것은 아니다.

안드로이드 API 16까지의 프레임워크 소스 : `ActivityManagerService.java`

```
public final class ActivityManagerService extends ActivityManagerNative
    implements Watchdog.Monitor, BatteryStatsImpl.BatteryCallback
{
    ...
    // Maximum amount of time for there to be no activity on a service before
    // we consider it non-essential and allow its process to go on the
    // LRU background list.
```

```
static final int MAX_SERVICE_INACTIVITY = 30*60*1000;
```

안드로이드 API 17부터의 프레임워크 소스 : `ActiveServices.java`

```
public class ActiveServices
{
    ...
    // Maximum amount of time for there to be no activity on a service before
    // we consider it non-essential and allow its process to go on the
    // LRU background list.
```

```
static final int MAX_SERVICE_INACTIVITY = 30*60*1000;
```

- 스타티드 서비스가 시작한지 30분 이내는 30분이 지난 다른 서비스 보다 안전하다.
- 어쨌든 서비스는 시작한지 30분을 초과하면 언제 죽을지 모른다.

스타티드 서비스 – 서비스의 생존 방법

37

- 백그라운드에서 잘 실행되는 서비스가 LMK에 의해 갑자기 종료해버리면 사용자에게 큰 불편을 줄 수 있다.
- 이를 위해 안드로이드는 서비스가 생존할 수 있는 두 가지 방법을 제공한다. 하나씩 살펴보자.

스타티드 서비스 – 서비스의 생존 방법

38

■ 첫 번째 서비스의 onStartCommand 함수 반환값을 살펴보자.

□ CountService.java

```
public class CountService extends Service {  
    ...  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        ...  
        return START_STICKY;  
    }  
}
```

스타티드 서비스 – 서비스의 생존 방법

39

상수 명	설명
START_STICKY	LMK가 서비스 앱의 프로세스를 강제 종료한 이후 가용 메모리가 확보되면 다시 서비스를 실행해준다. 그러므로 서비스의 생존 유지가 중요한 앱이라면 이 값을 반환하자. (default)
START_NOT_STICKY	LMK가 서비스 앱의 프로세스를 강제 종료한 이후 가용 메모리가 확보되어도 다시 서비스를 실행해주지 않는다. 그러므로 서비스의 생존이 중요하지 않은 앱이라면 이 값을 반환하자.
START_REDELIVER_INTENT	START_STICKY와 동일 하나 onStartCommand 매개변수의 인텐트 정보를 복원 해주는 점이 다르다. 특정 앱은 인텐트를 생성 및 설정하고 startService 함수의 인자로 전달한다. 서비스는 해당 인텐트를 onStartCommand의 매개변수로 전달받는다. 여기서 실행된 서비스가 LMK에 의해 강제 종료되고 다시 실행되면 onStartCommand 함수의 인텐트 매개변수를 이전에 받았던 인텐트로 다시 돌려준다. 그러므로 서비스의 생존 유지가 중요하고, 전달받은 인텐트 정보가 서비스 동작에 중요한 역할을 한다면 이 값을 반환하자.

스타티드 서비스 – 서비스의 생존 방법


40

■ START_REDELIVER_INTENT를 확인해보자.

□ CountService.java

```
public class CountService extends Service {  
    ...  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        super.onStartCommand(intent, flags, startId);  
        Log.i("superdroid", "onStartCommand(): " + intent);  
        ...  
        return START_REDELIVER_INTENT;  
    }  
}
```


스타티드 서비스 – 서비스의 생존 방법




CountService


Start Count

Stop Count


● 로그 확인

Time	TID	Text
12-12 10:49:06.059	701	onCreate()
12-12 10:49:06.069	701	onStartCommand() : Intent { ac
12-12 10:49:06.079	727	Count : 0
12-12 10:49:07.131	727	Count : 1
12-12 10:49:08.129	727	Count : 2
12-12 10:49:09.129	727	Count : 3
12-12 10:49:10.249	727	Count : 4
12-12 10:49:21.919	728	onCreate()
12-12 10:49:21.929	728	onStartCommand() : Intent { ac
12-12 10:49:21.949	741	Count : 0
12-12 10:49:22.988	741	Count : 1
12-12 10:49:24.041	741	Count : 2

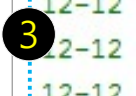





1




2



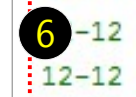
3



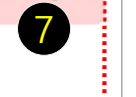
4




5



6



7



8

Devices

Name		
com.google.android.apps.maps:Locat	594	8618
com.android.defcontainer	627	8620
com.android.keychain	644	8621
com.svox.pico	658	8622
com.android.quicksearchbox	671	8623
kr.co.company.countservice	701	8619 / 87...

■ 그렇다면 LMK에 의해 죽었던 서비스가 다시 살아났다는 것을 알 방법은 없을까?

스타티드 서비스 – 서비스의 생존 방법

42

■ onStartCommand의 매개변수로 전달되는 플래그 값을 살펴보자.

□ CountService.java

```
public class CountService extends Service {  
    ...  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        super.onStartCommand(intent, flags, startId);  
        Log.i("superdroid", "onStartCommand(): " + intent);  
        ...  
        return START_REDELIVER_INTENT;  
    }  
}
```

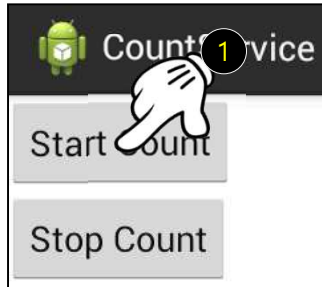
■ 이 플래그는 일반적인 환경에서는 0으로 넘어오지만,
서비스가 다시 살아나 이전에 인텐트를 재전달하는 경우
`Service.START_FLAG_REDELIVERY` 값이 넘어온다.

■ 따라서 해당 플래그 값을 비교함으로써 추가적인 예외처리를 하면 된다.

스타티드 서비스 – 서비스의 생존 방법

43

■ 두 번째 서비스의 startForeground 함수로 LMK 정리 대상에서 벗어나기



- 현재 서비스를 지각되는 서비스로 변경하기 전에 먼저 현재 서비스의 LMK 우선순위 상태를 확인해보자.

스타티드 서비스를 실행해둔다.

■ 다음의 ADB 명령어를 사용한다.

adb shell dumpsys activity

adb shell dumpsys activity 내용

...
ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)

Process LRU list (sorted by oom_adj):

```
PERS # 2: adj=sys /F trm= 0 285:system/1000 (fixed)
PERS #14: adj=pers /F trm= 0 434:com.android.phone/1001 (fixed)
PERS # 1: adj=pers /F trm= 0 391:com.android.systemui/u0a10023 (fixed)
Proc # 3: adj=fore /FA trm= 0 518:com.android.launcher/u0a10022 ...
Proc # 0: adj=prcp /F trm= 0 418:com.android.inputmethod.latin/...
Proc # 5: adj=bak /B trm= 0 608:com.android.quicksearchbox/u0a10030 ...
Proc # 4: adj=bak /B trm= 0 988: kr.co.company.countservice/...
```

스타티드 서비스 – 서비스의 생존 방법

44

■ 이제 서비스를 지각되는 서비스 상태로 변경한다.

□ AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.co.company.countservice">

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".CountService" />
    </application>

</manifest>
```

스타티드 서비스 – 서비스의 생존 방법

45

□ MainActivity.java

```
public void onClick(View v) {  
    switch(v.getId()) {  
        // 1. 카운트 시작  
        // =====  
        case R.id.start_count_btn: {  
            Intent serviceIntent = new Intent(this, CountService.class);  
  
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
                startForegroundService(serviceIntent);  
            } else {  
                startService(serviceIntent);  
            }  
  
            break;  
        }  
        // =====  
        ...  
    }  
}
```

스타티드 서비스 – 서비스의 생존 방법

```
@Override
public void onCreate() {
    super.onCreate();
    Log.i("superdroid", "onCreate()");

    // 1. noti/피케이션 객체 생성
    // =====
    Intent intent = new Intent(this, MainActivity.class);
    PendingIntent pIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_UPDATE_CURRENT);

    NotificationCompat.Builder builder;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        String channelId = "one-channel";
        String channelName = "My Channel One";
        NotificationChannel channel = new NotificationChannel(channelId, channelName, NotificationManager.IMPORTANCE_DEFAULT);

        NotificationManager manager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
        manager.createNotificationChannel(channel);

        builder = new NotificationCompat.Builder(this, channelId);
    } else {
        builder = new NotificationCompat.Builder(this);
    }

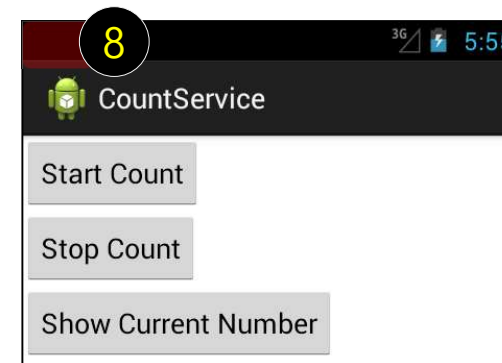
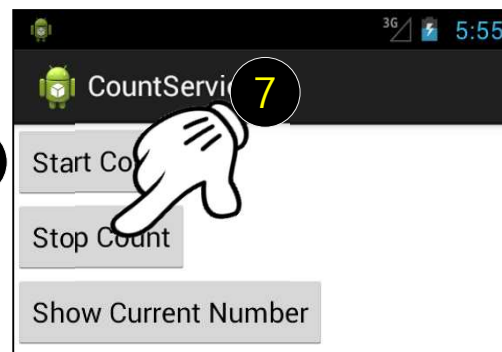
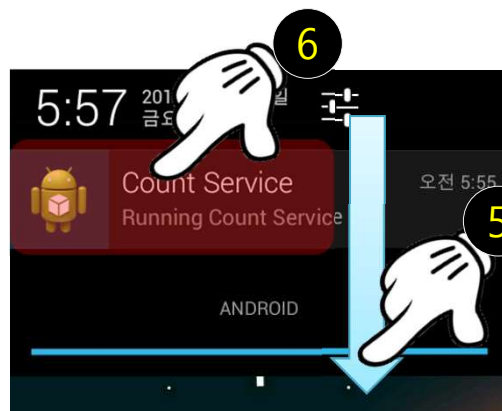
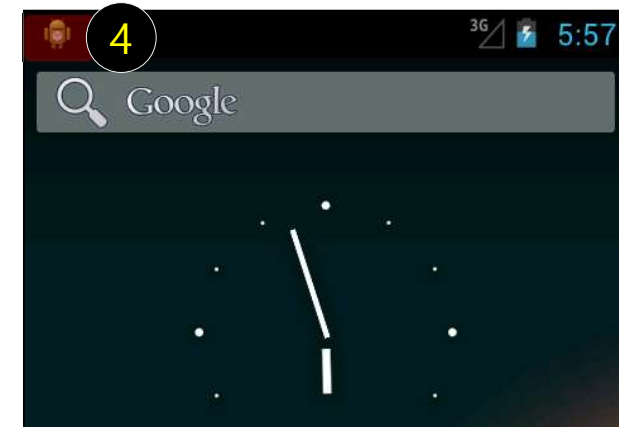
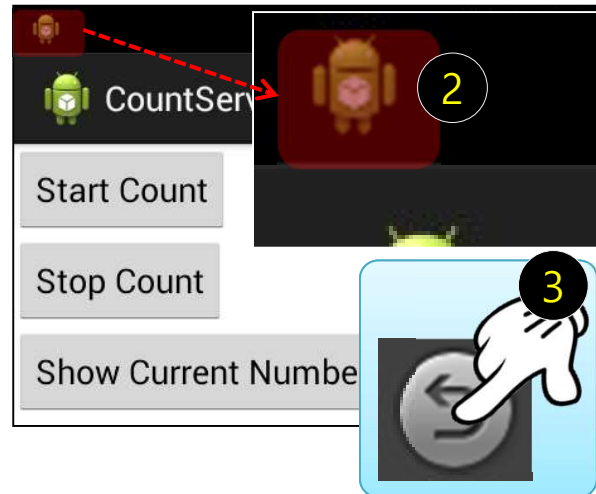
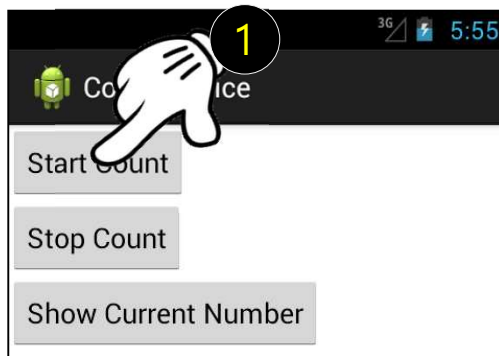
    builder.setContentTitle("Count Service")
        .setContentText("Running Count Service")
        .setSmallIcon(R.mipmap.ic_launcher)
        .setContentIntent(pIntent);

    Notification noti = builder.build();
    // =====

    // 2. 포그라운드 서비스 설정 (지각할 수 있는 서비스가 된다.)
    // =====
    startForeground(1234, noti);
    // =====
}
```

스타티드 서비스 – 서비스의 생존 방법

47



■ 왜 지각되는 서비스가 되었는지 이해했는가!

■ 사용자는 noti피케이션 영역에서 아이콘을 보고 서비스의 동작을 지각할 수 있기 때문이다.

스타티드 서비스 – 서비스의 생존 방법

48

adb dumpsys activity 내용

...

ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)

Process LRU list (sorted by oom_adj):

PERS # 2: adj=sys /F trm= 0 285:system/1000 (fixed)

PERS #13: adj=pers /F trm= 0 434:com.android.phone/1001 (fixed)

PERS # 1: adj=pers /F trm= 0 391:com.android.systemui/u0a10023 (fixed)

Proc # 3: adj=fore /FA trm= 0 518:com.android.launcher/u0a10022 (top-activity)

Proc # 4: **adj=prcp** **/FS** trm= 0 851: **kr.co.company.countservice**

...

■ prcp는 **Perceptible**의 약자로 지각할 수 있는 것을 의미한다.

바운드 서비스 – 서비스 연결

49

- 이제 바운드 서비스를 통해 외부 앱에서 서비스를 연결하고 서비스에 정의된 인터페이스를 직접 접근하여 사용해보자.
- 이를 위해 초당 1씩 증가하는 예제 앱에서 현재까지 증가된 수를 가지고 오는 기능을 바운드 서비스로 구현해본다.

바운드 서비스 – 서비스 연결

50

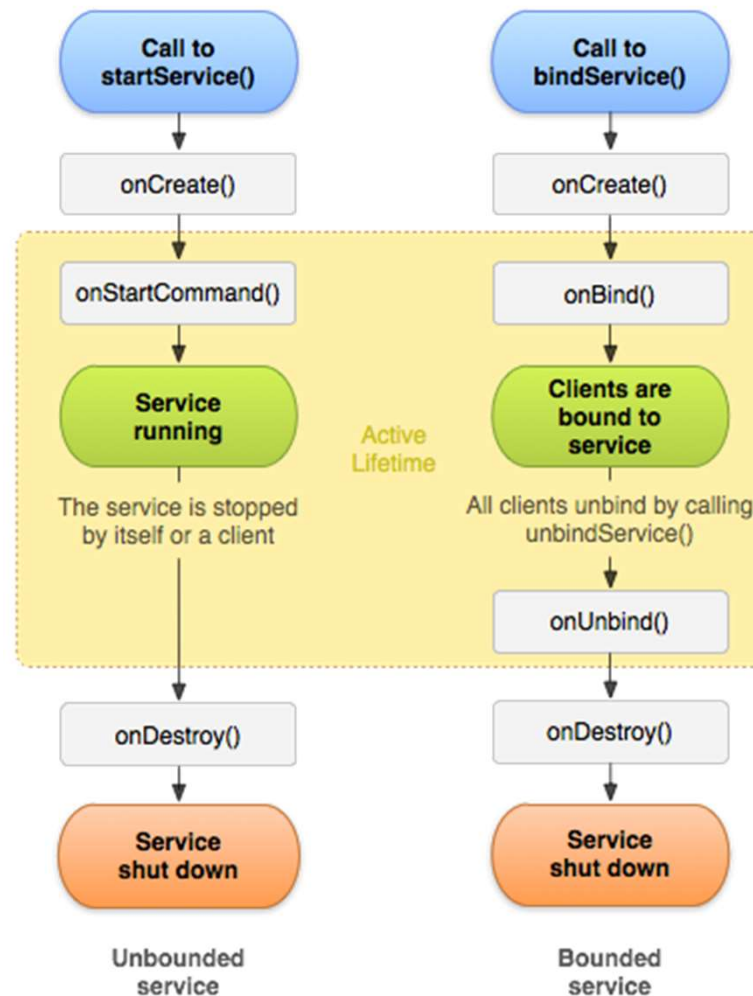
- 외부 앱 컴포넌트에서 서비스의 인터페이스를 사용하려면 서비스와 연결이 선행되어야 한다.

```
public class CountService extends Service {  
  
    ...  
  
    @Nullable  
    @Override  
    public IBinder onBind(Intent intent) {  
        Log.i("superdroid", "onBind()" + intent);  
        return null;  
    }  
  
    @Override  
    public boolean onUnbind(Intent intent) {  
        Log.i("superdroid", "onUnbind()");  
        return super.onUnbind(intent);  
    }  
}
```

바운드 서비스 - 서비스 연결

51

□ 스타티드와 바운드 서비스의 생명주기



바운드 서비스 – 서비스 연결

52

- 바운드 서비스를 사용할 클라이언트 앱을 구현해보자.
 - Application name: CountServiceClient
 - Company domain: company.co.kr

- AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.co.company.countserviceclient">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

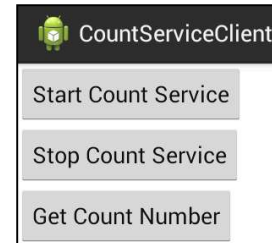
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

바운드 서비스 - 서비스 연결

53

□ activiy_main.xml



● 레이아웃 구조

LinearLayout
OK start_count_btn (Button) - "Start Col
OK stop_count_btn (Button) - "Stop Col
OK get_cur_count_number_btn (Button)

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">

<Button android:id="@+id/start_count_btn"
android:text="Start Count Service"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onClick"/>

<Button android:id="@+id/stop_count_btn"
android:text="Stop Count Service"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onClick"/>

<Button android:id="@+id/get_cur_count_number_btn"
android:text="Get Count Number"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onClick"/>

</LinearLayout>
```

바운드 서비스 – 서비스 연결

54

□ CountService의 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="kr.co.company.countservice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>

                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

        <service android:name=".CountService"
            android:exported="true"/>

    </application>

</manifest>
```

바운드 서비스 – 서비스 연결

55

이 객체를 통해 서비스의 연결과 해제를 탐지할 수 있다.

□ CountServiceClient의 MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
            Log.d("superdroid", "onServiceConnected()");
        }
        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            Log.d("superdroid", "onServiceDisconnected()");
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // 카운트 서비스 연결
        // =====
        Intent serviceIntent = new Intent();
        ComponentName componentName = new ComponentName("kr.co.company.countservice", "kr.co.company.countservice.CountService");
        serviceIntent.setComponent(componentName);
        bindService(serviceIntent, mConnection, BIND_AUTO_CREATE);
        // =====
    }

    @Override
    protected void onDestroy() {
        // 카운트 서비스 해제
        // =====
        unbindService(mConnection);
        // =====
        super.onDestroy();
    }
}
```

■ 왜 액티비티의 onCreate, onDestroy 생명주기 함수에 서비스 연결과 해제 처리를 할까?

바운드 서비스는 대부분 액티비티 생명주기 안에서 지속적으로 사용되는 경우가 대부분이다. 따라서 서비스 연결과 해제를 액티비티 생명주기에 맞추는 것이 편리하다.

바운드 서비스 – 서비스 연결

56

□ CountServiceClient의 MainActivity.java

```
public void onClick( View v ) {
    switch(v.getId()) {
        // 1. 카운트 서비스 시작
        // =====
        case R.id.start_count_btn:
        {
            Intent serviceIntent = new Intent();
            ComponentName componentName = new ComponentName("kr.co.company.countservice",
"kr.co.company.countservice.CountService");
            serviceIntent.setComponent(componentName);
            startService(serviceIntent);

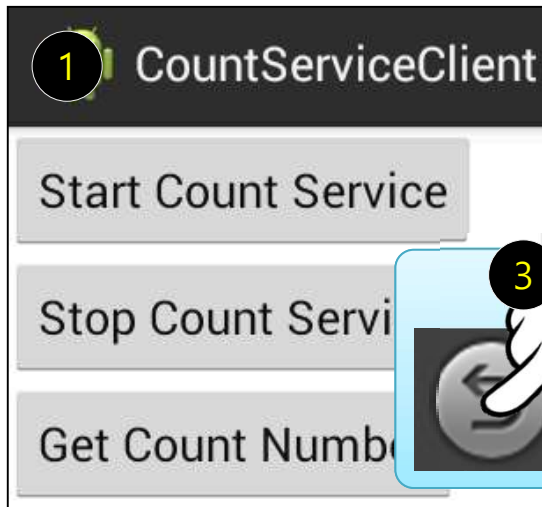
            break;
        }
        // =====

        // 2. 카운트 서비스 종료
        // =====
        case R.id.stop_count_btn:
        {
            Intent serviceIntent = new Intent();
            ComponentName componentName = new ComponentName("kr.co.company.countservice",
"kr.co.company.countservice.CountService");
            serviceIntent.setComponent(componentName);
            stopService(serviceIntent);
            break;
        }
        // =====

        // 3. 바인딩된 서비스에 현재까지 증가된 수 얻어오기
        // =====
        case R.id.get_cur_count_number_btn:
            break;
        // =====
    }
}
}
```


바운드 서비스 - 서비스 연결

57



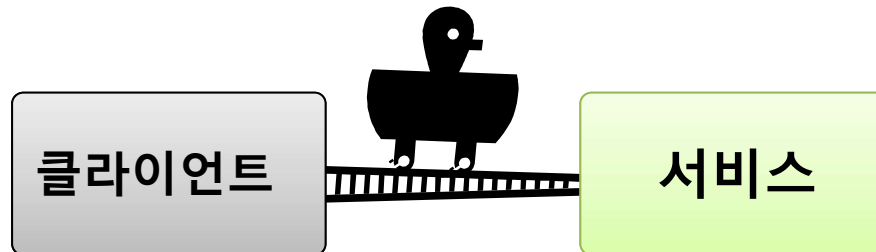
● 로그 확인

2 onCreate()
onBind()
4 onUnbind()
onDestroy()

onServiceConnected
onServiceDisconnected



실질적으로 통신하게 될 매개체가 없다.



■ 안드로이드에서는 이렇게 통신을 가능케 하는 매개체를 **바인더**라고 부른다.

바운드 서비스 – 바인더와 AIDL

58

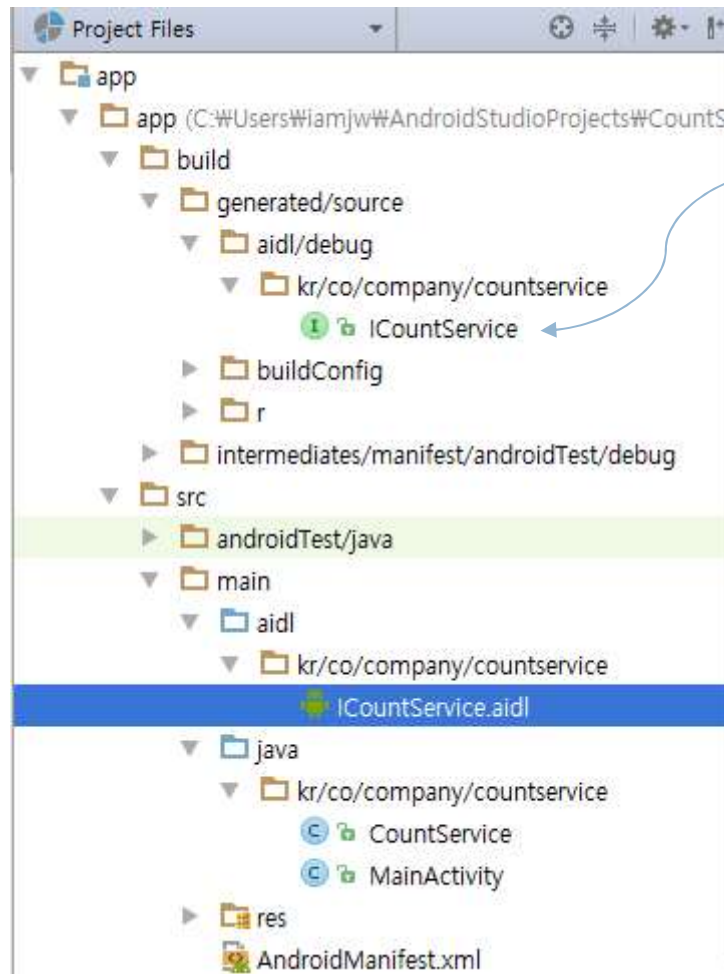
- 바인더는 서비스에 존재하는 인터페이스를 포함하며, 클라이언트는 바인더를 이용해서 서비스의 인터페이스를 호출할 수 있다.
- 안드로이드는 인터페이스 정의 언어인 AIDL (Android Interface Definition Language)을 제공하고, 해당 언어로 인터페이스를 작성하면 자동으로 바인더를 생성해 주고 있다.



- AIDL은 자바에서 인터페이스 클래스를 정의하는 방법과 유사하여 별도로 공부할 필요가 없다.

바운드 서비스 – 바인더와 AIDL

59



```
../../../../
package kr.co.company.countservice;
// Declare any non-default types here with import statements

public interface ICountService extends android.os.IInterface
{
    /** Local-side IPC implementation stub class. */
    public static abstract class Stub extends android.os.Binder implements kr.co.company.countservice.ICountService
    {
        private static final java.lang.String DESCRIPTOR = "kr.co.company.countservice.ICountService";
        /** Construct the stub at attach it to the interface. */
        public Stub() { this.attachInterface(this, DESCRIPTOR); }
        /**
         * Cast an IBinder object into an kr.co.company.countservice.ICountService interface,
         * generating a proxy if needed.
         */
        public static kr.co.companv.countservice.ICountService asInterface(android.os.IBinder obj)
    }
}
```

```
// ICountService.aidl
package kr.co.company.countservice;

// Declare any non-default types here with import statements

interface ICountService {

    int getCurCountNumber();

}
```

바운드 서비스 – 바인더와 AIDL

60

```
public class CountService extends Service {
```

```
...
```

```
    ICountService.Stub mBinder = new ICountService.Stub() {  
        @Override 클라이언트는 전달받은 바인더를 통해 서비스에 정의된 인터페이스를 사용할 수 있다.  
        public int getCurCountNumber() throws RemoteException {  
            return mCurNum;  
        }  
    };
```

```
    @Nullable
```

```
    @Override
```

```
    public IBinder onBind(Intent intent) {  
        Log.i("superdroid", "onBind()" + intent);  
        return mBinder;  
    }
```

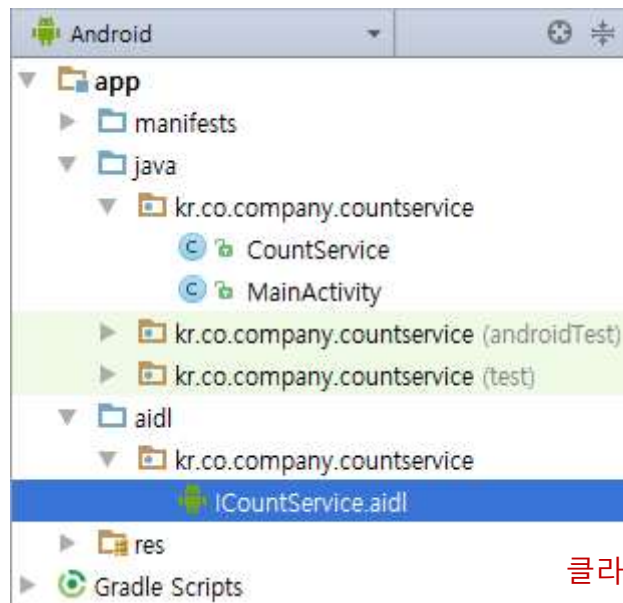
```
...
```

```
}
```

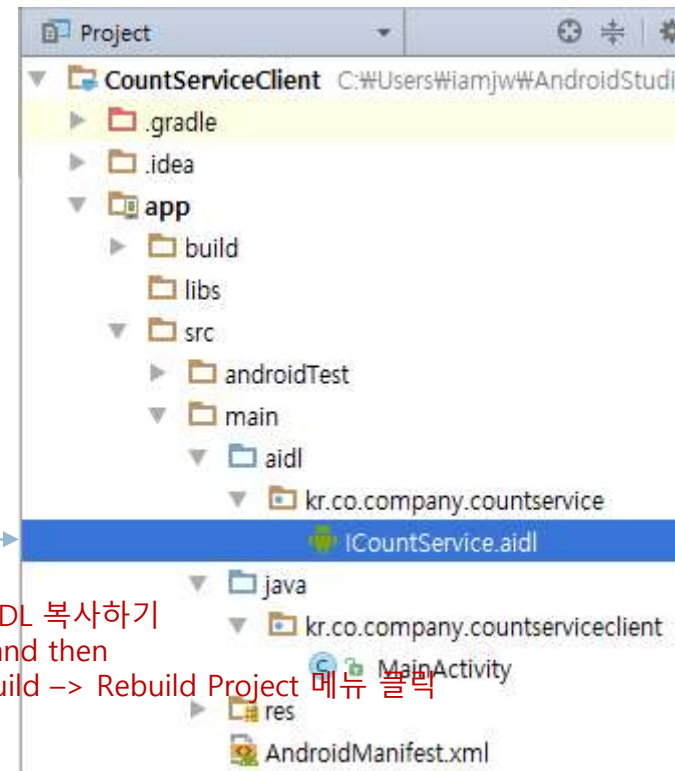
바운드 서비스 – 바인더와 AIDL

61

● 카운트 서비스 프로젝트



● 카운트 서비스 클라이언트 프로젝트



클라이언트 프로젝트로 AIDL 복사하기
and then
안드로이드 스튜디오의 Build -> Rebuild Project 메뉴 클릭

바운드 서비스 – 바인더와 AIDL

```
public class MainActivity extends AppCompatActivity {
```

```
    private ICountService mBinder = null;
```

```
    private ServiceConnection mConnection = new ServiceConnection() {
```

```
        @Override
```

```
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
```

```
            Log.d("superdroid", "onServiceConnected()");
```

```
            mBinder = ICountService.Stub.asInterface(iBinder);
```

```
        }
```

```
    ...
```

```
};
```

```
...
```

```
public void onClick(View v) {
```

```
    switch(v.getId()) {
```

```
        ...
```

```
        // 3. 바인딩된 서비스에 현재까지 증가된 수 얻어오기
```

```
        // =====
```

```
        case R.id.get_cur_count_number_btn:
```

```
        {
```

```
            int curCountNumber;
```

```
            try {
```

```
                curCountNumber = mBinder.getCurCountNumber();
```

```
                Toast.makeText( this, "Cur Count : " + curCountNumber, Toast.LENGTH_LONG ).show();
```

```
            } catch (RemoteException e) {
```

```
                e.printStackTrace();
```

```
            }
```

```
            break;
```

```
        }
```

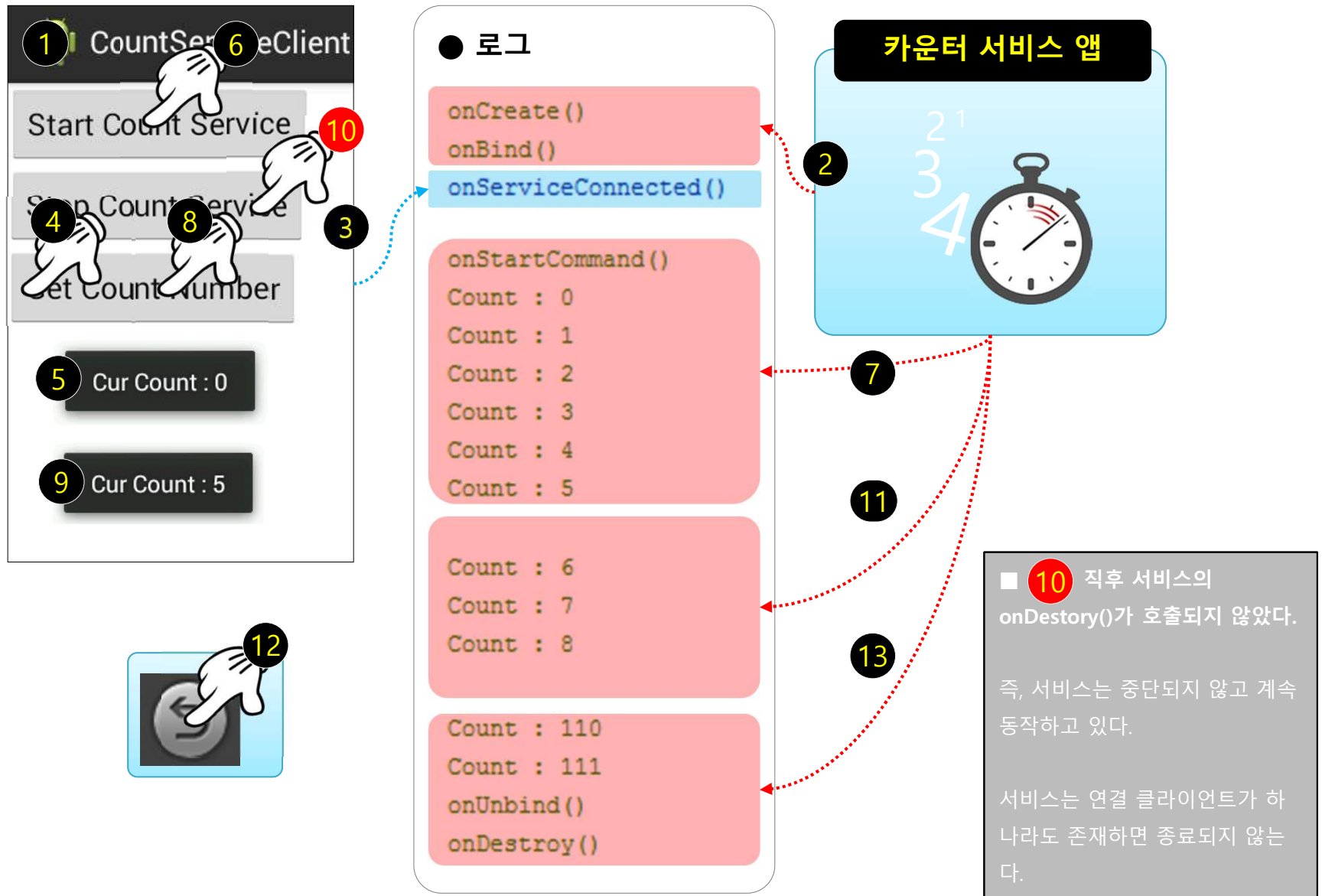
```
        // =====
```

```
    }
```

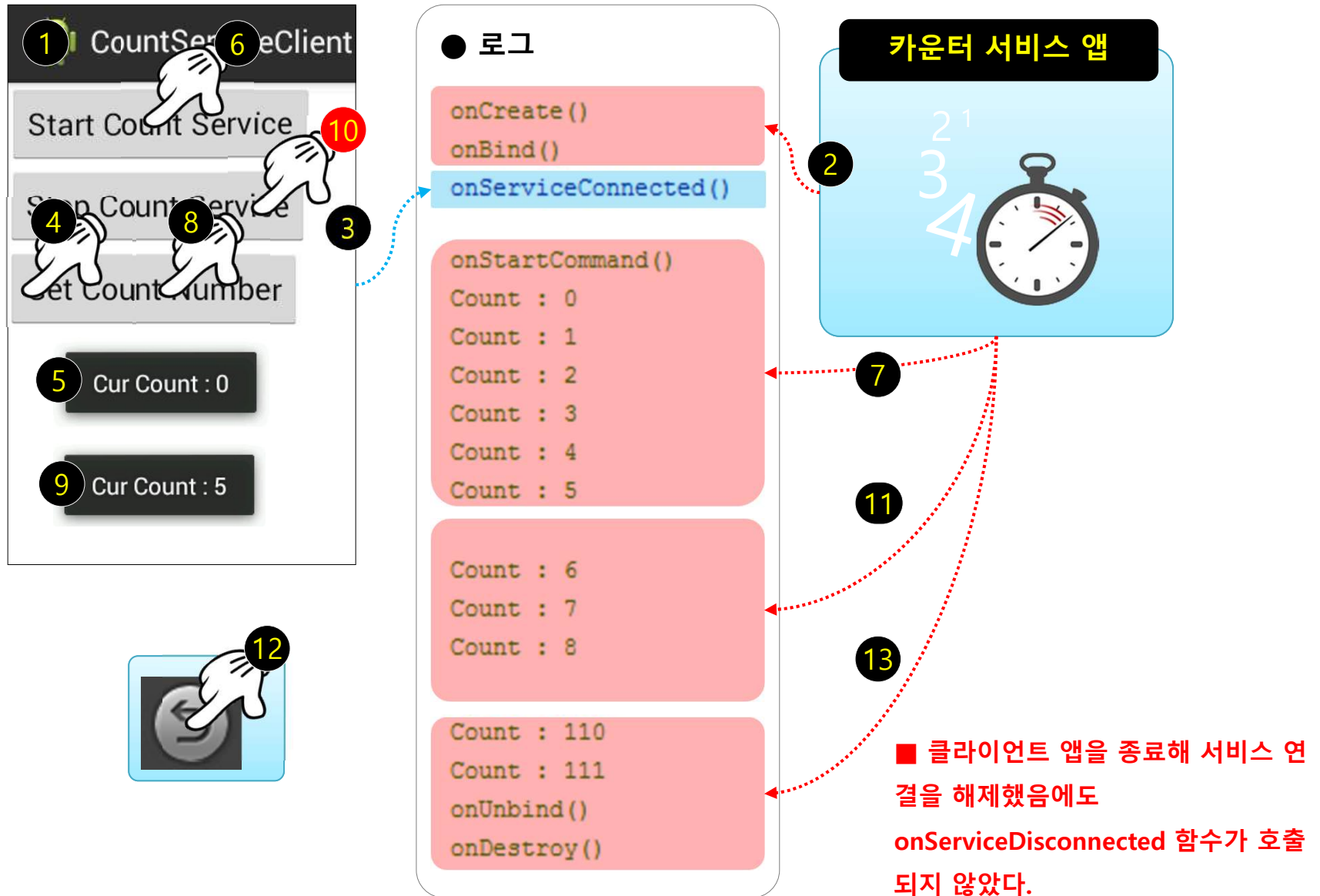
```
}
```

```
}
```

바운드 서비스 – 바인더와 AIDL



바운드 서비스 – 바인더와 AIDL



바운드 서비스 – BIND_AUTO_CREATE 플래그

65

- 그 이유는 클라이언트에서 서비스 연결을 위해 사용했던 bindService() 메소드의 세 번째 인자 BIND_AUTO_CREATE 플래그 때문이다.

```
public class MainActivity extends AppCompatActivity {  
    ...  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        // 카운트 서비스 연결  
        // =====  
        Intent serviceIntent = new Intent();  
        ComponentName componentName = new ComponentName("kr.co.company.countservice", "kr.co.company.countservice.CountService");  
        serviceIntent.setComponent(componentName);  
        bindService(serviceIntent, mConnection, BIND_AUTO_CREATE);  
        // =====  
    }  
    ...  
}
```

- 안드로이드 서비스는 기본적으로 클라이언트의 startService() 호출을 통해 서비스가 시작되지 않은 상태에서는 bindService() 호출을 통해 서비스를 연결할 수 없다.
 - 즉, 스타티드 서비스가 실행되지 않은 상태에서는 바운드 서비스를 사용할 수 없다는 말이다.

바운드 서비스 – BIND_AUTO_CREATE 플래그

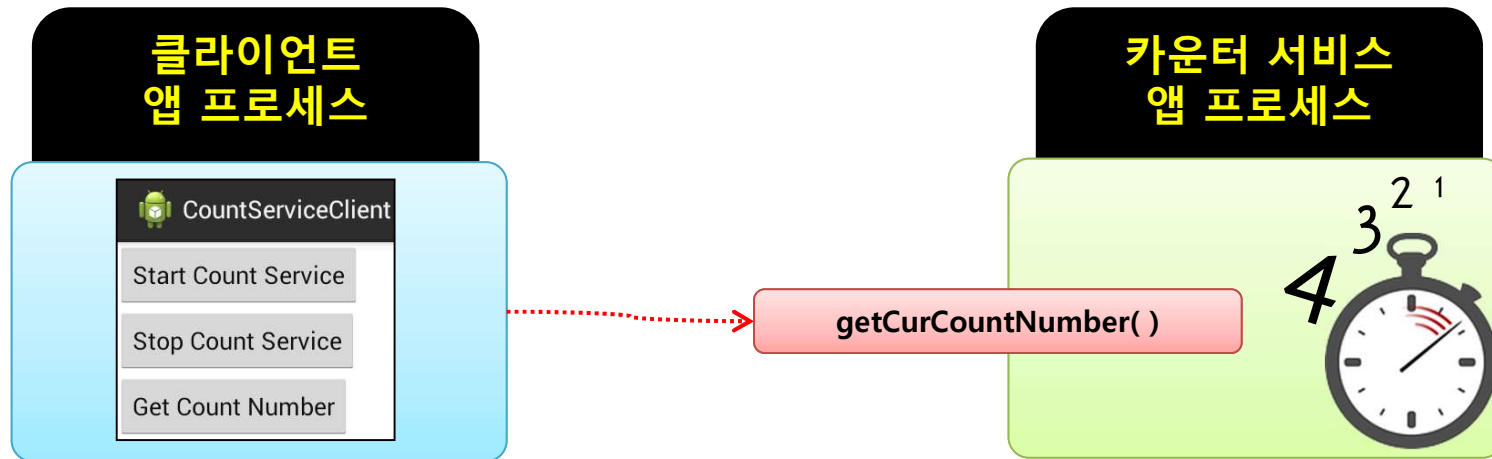
66

- 하지만 bindService() 메소드의 인자로 **BIND_AUTO_CREATE** 플래그를 전달하면 스타티드 서비스가 실행되지 않은 상태에서도 바운드 서비스의 사용이 가능해진다.
- **따라서 스타티드 & 바운드 서비스가 아니라면 BIND_AUTO_CREATE 옵션은 필수적이다.**
 - ▣ 이렇게 분리되어 독립적으로 동작하는 바운드 서비스는 스타티드 서비스와 상관없이 언제든지 사용할 수 있다.
- 또한 클라이언트에서 한번 연결된 바운드 서비스는 특정한 이유로 서비스가 죽더라도, 자동으로 다시 연결된다.
 - ▣ 즉, 바운드 서비스의 안정성을 시스템이 보장하는 것이다.
- BIND_AUTO_CREATE 플래그가 설정된 상태에서 Android 시스템이 onServiceDisconnected()를 호출하는 경우는 서비스로의 연결이 예기치 못하게 끊어졌을 때, 즉 서비스가 충돌했거나 중단되었을 때 등이다.
 - ▣ 클라이언트가 바인딩을 해제한다고 이것이 호출되지는 않는다.

바운드 서비스 – 바인더 객체

67

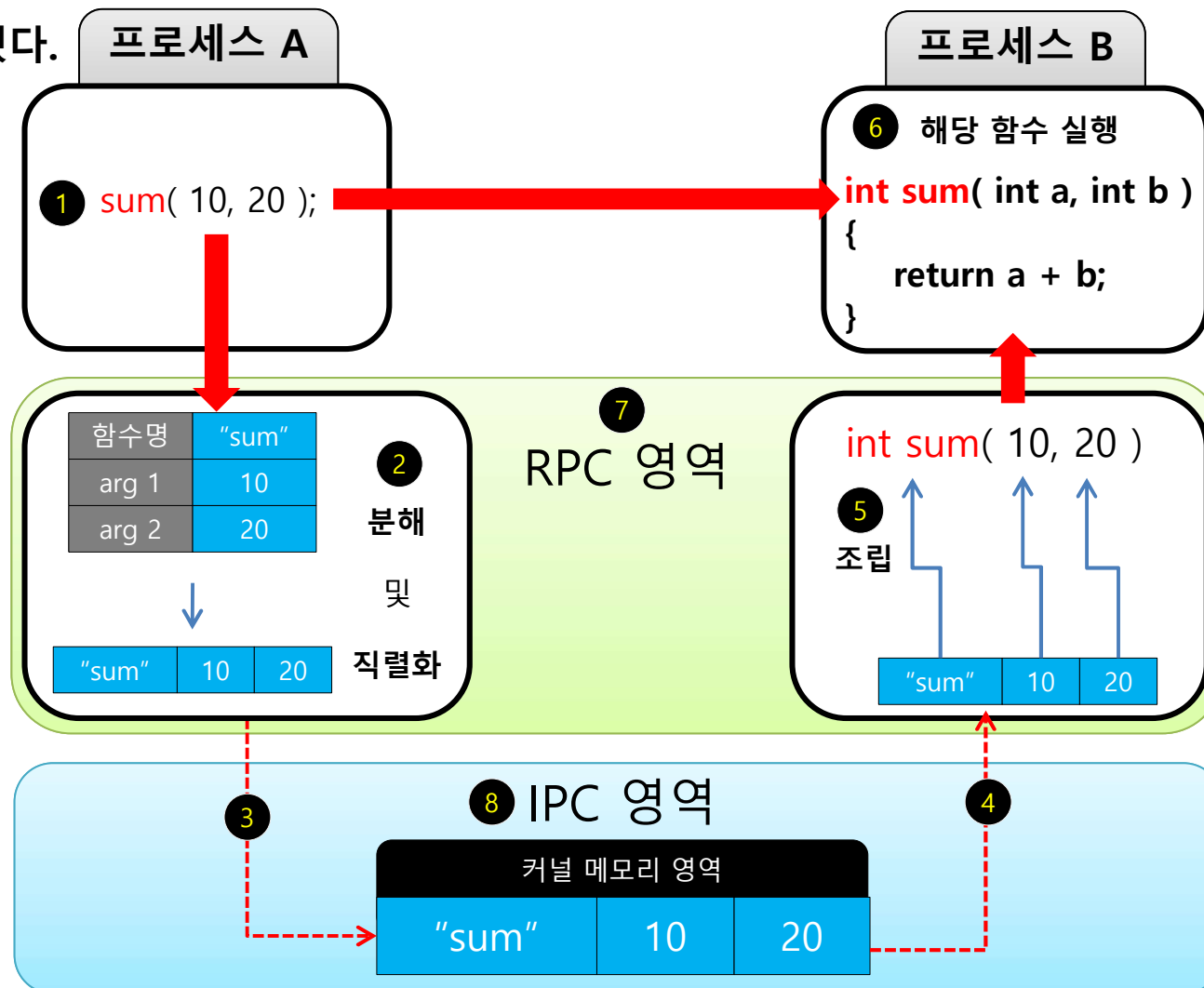
- AIDL을 통해 자동으로 생성된 바인더 객체에 대해서 좀더 살펴보자.



- 클라이언트 앱에서 카운트 서비스 앱의 함수를 **직접** 호출할 수 있었다.
 - ▣ 서로 다른 프로세스 간에는 메모리를 공유할 수 없기 때문에 직접적으로 함수를 호출하는 것은 불가능하다.
 - 따라서 커널의 공유 메모리를 사용하여 프로세스 간 통신을 해야 한다.

바운드 서비스 – 바인더 객체

- 안드로이드는 이를 위해 프로세스 간 통신을 가능케 하는 **IPC** Inter Process Communication 기술과 프로세스 간 함수를 호출할 수 있는 **RPC** Remote Procedure Call 기술을 바인더에 적용했다.



바운드 서비스 – 바인더 객체

69

- RPC와 IPC가 어떻게 안드로이드에 적용되었고 바인더란 녀석은 어떤 동작을 하는지 알아보자.
- 먼저 AIDL 파일을 통해 자동으로 생성된 자바 클래스 파일을 살펴본다.



```
interface ICountService {
    Stub();
    asInterface(IBinder): ICountService
    asBinder(): IBinder
    onTransact(int, Parcel, Parcel, int): boolean
    DESCRIPTOR: String = "kr.co.company.countservice.ICountService"
    TRANSACTION_getCurCountNumber: int = (android.os.IBinder.FIRST_CALL_TRANSACTION + 0)
    getCurCountNumber(): int
}
```

● 함수를 호출하는 측

Proxy : 함수를 분해한 뒤 전송한다.

● 함수가 호출 되는 측 (외부 프로세스의 함수)

Stub : 함수를 조립한 뒤 호출한다.

바운드 서비스 – 바인더 객체

70

ICountService.java

```
package kr.co.company.countservice;
public interface ICountService extends android.os.IInterface
{
    /** Local-side IPC implementation stub class. */
    public static abstract class Stub extends android.os.Binder ...
    {
        ...
        @Override
        public boolean
        onTransact( int code, android.os.Parcel data, android.os.Parcel reply, int flags )
                                                    throws android.os.RemoteException
        {
            switch ( code )
            {
                ...
                case TRANSACTION_getCurCountNumber:
                {
                    data.enforceInterface( DESCRIPTOR );
                    int _result = this.getCurCountNumber();
                    reply.writeNoException();
                    reply.writeInt( _result );
                    return true;
                }
            }
            return super.onTransact( code, data, reply, flags );
        }
    }

    public int getCurCountNumber() throws android.os.RemoteException;
}
```

바운드 서비스 – 바인더 객체

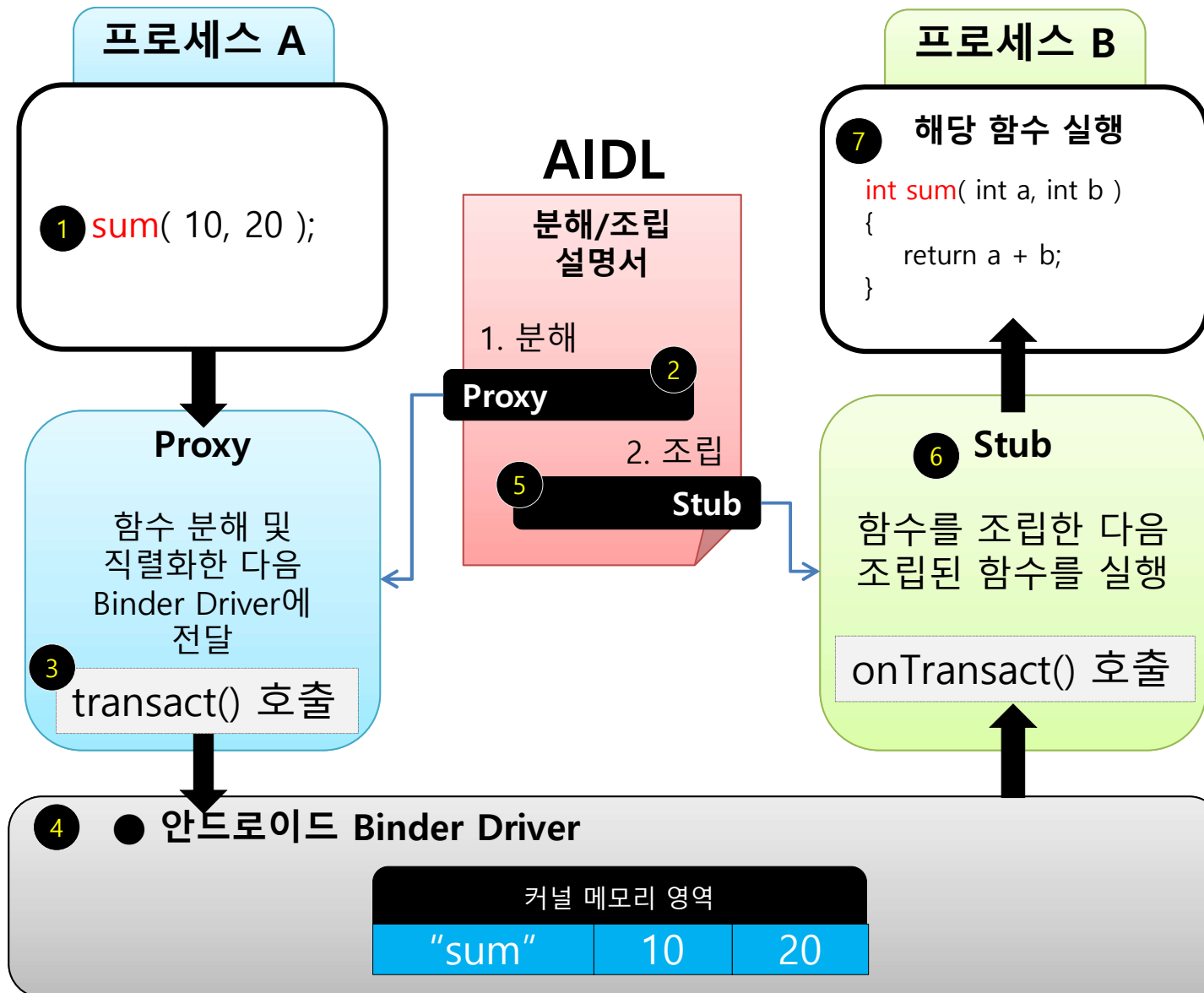
71

```
public int getCurCountNumber() throws android.os.RemoteException;

private static class Proxy implements ...
{
    ...
    @Override
    public int getCurCountNumber() throws android.os.RemoteException
    {
        android.os.Parcel _data = android.os.Parcel.obtain();
        android.os.Parcel _reply = android.os.Parcel.obtain();
        int _result;
        try
        {
            data.writeInterfaceToken( DESCRIPTOR );
            mRemote.transact( Stub.TRANSACTION_getCurCountNumber,
                             _data, _reply, 0 );
            _reply.readException();
            _result = _reply.readInt();
        }
        finally
        {
            _reply.recycle();
            _data.recycle();
        }
        return _result;
    }
}

static final int TRANSACTION_getCurCountNumber = ( android.os.IBinder.FIRST_CALL_TRANSACTION + 0 );
}
```

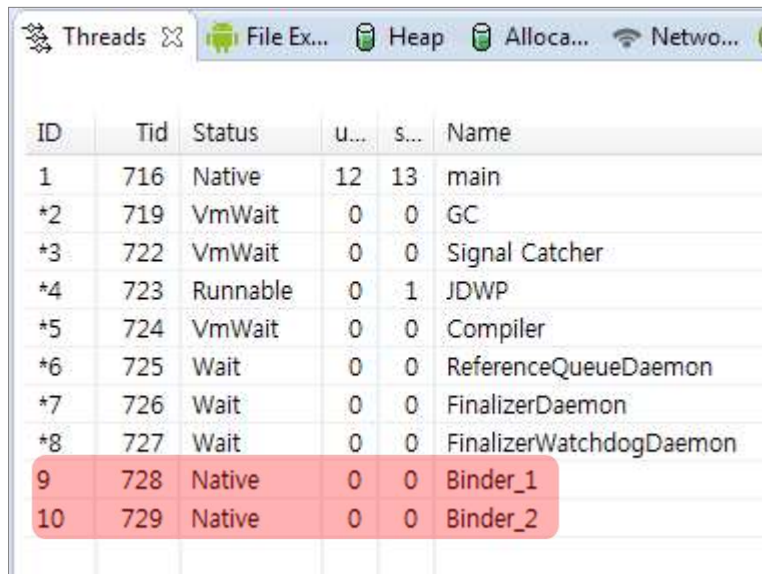
바운드 서비스 – 바인더 객체



바운드 서비스 – 바인더 객체

73

- 바운드 서비스의 메소드는 어느 스레드에서 동작할까?
- 리모트 클라이언트에서 호출하는 경우에는 별도의 작업 스레드에서 동작한다.



ID	Tid	Status	u...	s...	Name
1	716	Native	12	13	main
*2	719	VmWait	0	0	GC
*3	722	VmWait	0	0	Signal Catcher
*4	723	Runnable	0	1	JDWP
*5	724	VmWait	0	0	Compiler
*6	725	Wait	0	0	ReferenceQueueDaemon
*7	726	Wait	0	0	FinalizerDaemon
*8	727	Wait	0	0	FinalizerWatchdogDaemon
9	728	Native	0	0	Binder_1
10	729	Native	0	0	Binder_2

- 로컬 클라이언트(서비스 앱 내)에서 호출하는 경우에는 호출하는 스레드에서 서비스의 메소드가 동작한다.
 - 이 경우에는 메인 스레드에서 동작할 수도 있으므로 주의해서 사용해야 한다.

바운드 서비스 – 바운드 서비스의 콜백 인터페이스

74

- 서비스의 공개된 인터페이스의 메소드가 오래 걸리는 작업을 수행한다면 그 만큼 해당 인터페이스를 사용한 클라이언트는 대기해야 한다.

```
public class CountService extends Service {  
  
    ...  
  
    ICountService.Stub mBinder = new ICountService.Stub() {  
        @Override  
        public int getCurCountNumber() throws RemoteException {  
            try { Thread.sleep(10000); }  
            catch (InterruptedException e) { e.printStackTrace(); }  
  
            return mCurNum;  
        }  
    };  
  
    ...  
}
```

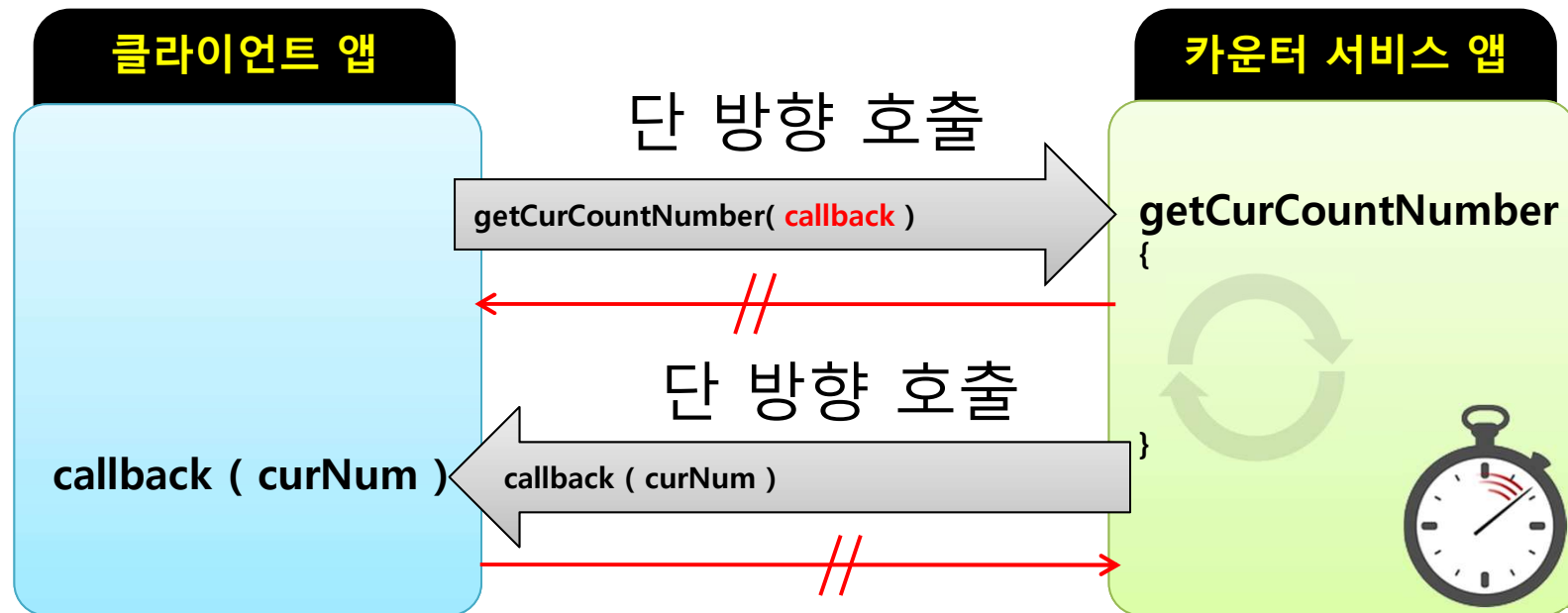


액티비티 ANR 발생

바운드 서비스 – 바운드 서비스의 콜백 인터페이스

75

- 이렇게 즉각적인 응답을 줄 수 없는 경우를 위해 서비스는 콜백 인터페이스를 지원한다.
 - ▣ 콜백 인터페이스도 AIDL로 정의한다.



바운드 서비스 – 바운드 서비스의 콜백 인터페이스

76

- ICountService.aidl과 동일한 폴더에 ICountServiceCallback.aidl을 생성한다.

```
// ICountServiceCallback.aidl
package kr.co.company.countservice;

interface ICountServiceCallback {
    oneway void getCurCountNumberCallback(int curNum);
}
```

해당 함수를 호출하는 클라이언트는 반환을 기다리지 않게 된다.

- 기존의 ICountService.aidl을 수정한다.

```
// ICountService.aidl
package kr.co.company.countservice;

import kr.co.company.countservice.ICountServiceCallback;

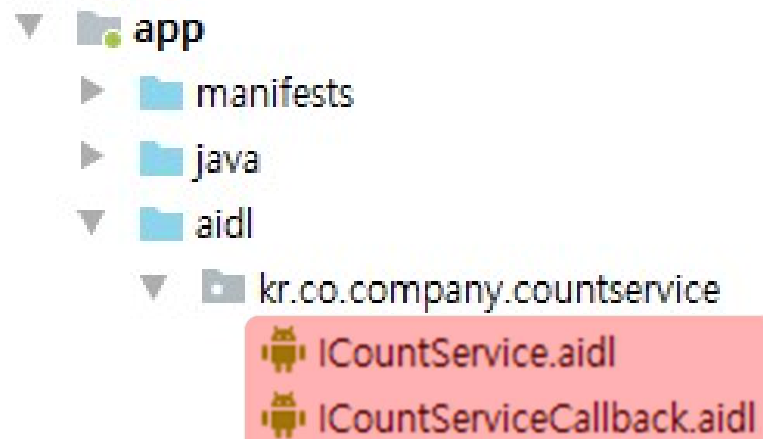
interface ICountService {
    oneway void getCurCountNumber(ICountServiceCallback callback);
}
```

동일한 패키지에 있는 클래스도 import해야 한다.

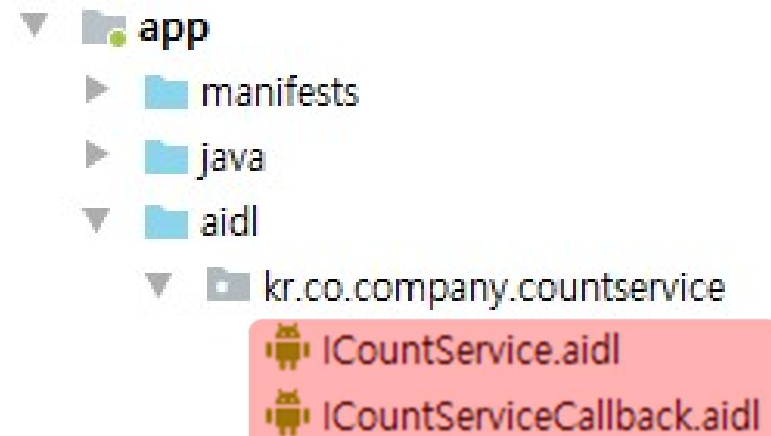
바운드 서비스 – 바운드 서비스의 콜백 인터페이스

77

● 카운트 서비스 패키지



● 카운트 서비스 클라이언트 패키지



바운드 서비스 – 바운드 서비스의 콜백 인터페이스

78

□ 카운트 서비스를 수정한다.

```
public class CountService extends Service {  
  
    ...  
  
    ICountService.Stub mBinder = new ICountService.Stub() {  
        @Override  
        public void getCurCountNumber(ICountServiceCallback callback) throws RemoteException {  
            try { Thread.sleep(10000); }  
            catch (InterruptedException e) { e.printStackTrace(); }  
            callback.getCurCountNumberCallback(mCurNum);  
        }  
    };  
  
    ...  
}
```

바운드 서비스 – 바운드 서비스의 콜백 인터페이스

79

□ 클라이언트 액티비티를 수정한다.

```
public class MainActivity extends AppCompatActivity {

    private ICountService mBinder = null;

    ICountServiceCallback mCurCountCallback = new ICountServiceCallback.Stub() {
        @Override
        public void getCurCountNumberCallback(final int curNum) throws RemoteException {
            Log.i("superdroid", "Cur Count Number: " + curNum);

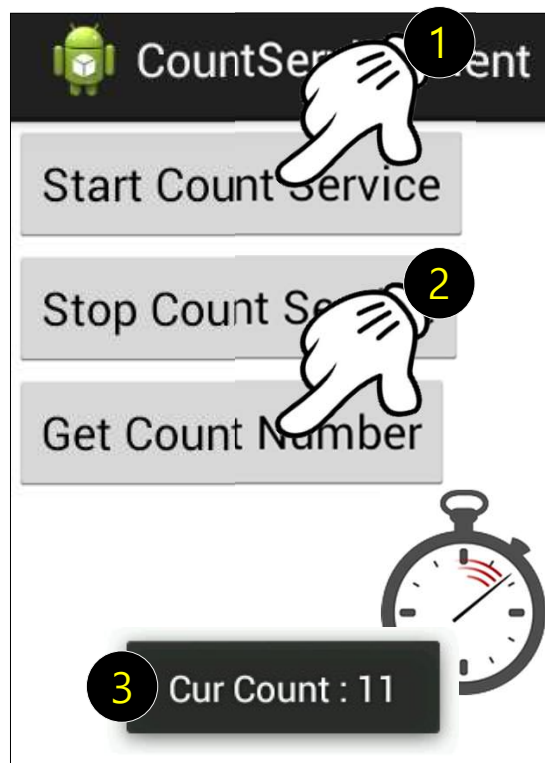
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    Toast.makeText(MainActivity.this, "Cur Count: " + curNum, Toast.LENGTH_LONG).show();
                }
            });
        }
    };

    ...
    public void onClick(View v) {
        switch(v.getId()) {
            ...
            // 3. 바인딩된 서비스에 현재까지 증가된 수 얻어오기
            // =====
            case R.id.get_cur_count_number_btn:
            {
                try { mBinder.getCurCountNumber(mCurCountCallback); }
                catch (RemoteException e) { e.printStackTrace(); }

                break;
            }
            // =====
        }
    }
}
```

바운드 서비스 – 바운드 서비스의 콜백 인터페이스

80



● 클라이언트 앱의 로그

PID	TID	Text
1124	1136	Cur Count Number : 11

● 클라이언트 앱 프로세스의 스레드 정보

Threads					
File Explorer Heap Allocation Tracker					
ID	Tid	Status	u...	s...	Name
1	1124	Native	44	35	main
9	1136	Native	0	1	Binder_1
10	1137	Native	0	0	Binder_2

바운드 서비스 – 바운드 서비스의 생존

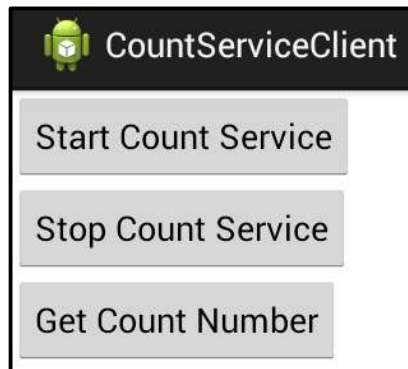
81

- 스타티드 서비스의 생존에 대한 우선순위는 클라이언트에 독립적이다.
- 그러나 바운드 서비스의 생존에 대한 우선순위는 클라이언트에 종속적이다.
 - ▣ 클라이언트 프로세스 상태를 그대로 따라간다.
- 우선 다음과 같이 카운트 서비스 소스에 포그라운드 서비스가 설정되어 있다면 제거한다.

```
public class CountService extends Service {  
    ...  
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Log.i("superdroid", "onCreate()");  
        ...  
        // 2. 포그라운드 서비스 설정 (지각할 수 있는 서비스가 된다.)  
        // =====  
        startForeground(1234, noti);  
        // =====  
    }  
    ...  
}
```

바운드 서비스 – 바운드 서비스의 생존

82



카운터 서비스 앱

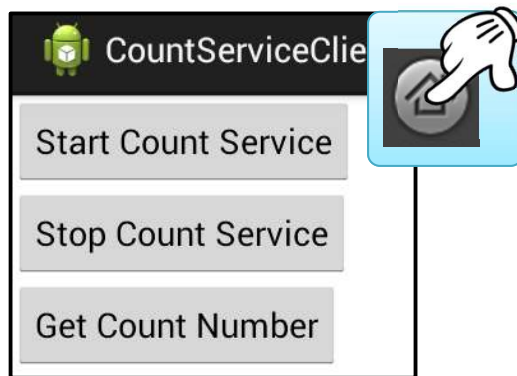


adb dumpsys activity 내용

```
...
ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)
Process LRU list (sorted by oom_adj):
...
Proc # 1: fore  T/A/T  trm: 0 12887:kr.co.company.countserviceclient/u0a113 (top-activity)
Proc # 0: vis   F/  /T  trm: 0 12742:kr.co.company.countservice/u0a114 (service)
kr.co.company.countservice/.CountService<=Proc{12887:kr.co.company.countserviceclient/u0a113}
```

바운드 서비스 – 바운드 서비스의 생존

83



카운터 서비스 앱



adb dumpsys activity 내용

```
...
ACTIVITY MANAGER RUNNING PROCESSES (dumpsys activity processes)
Process LRU list (sorted by oom_adj):
...
Proc # 2: prev B/ /LA trm: 0 12887:kr.co.company.countserviceclient/u0a113 (previous)
Proc # 1: cch B/ /LA trm: 0 12742:kr.co.company.countservice/u0a114 (cch-bound-ui-services)
kr.co.company.countservice/.CountService<=Proc{12887:kr.co.company.countserviceclient/u0a113}
```

내부 바운드 서비스

84

- 지금까지는 바운드 서비스를 사용하기 위해서는 AIDL 파일을 작성해야만 했다.
- 하지만 서비스의 인터페이스를 외부 앱에 공개하지 않고 내부 앱에서만 사용하는 경우에는 복잡한 IPC, RPC 등을 사용할 필요가 없다.
 - ▣ 즉, 내부 프로세스는 메모리가 공유되고, 서비스의 인터페이스를 직접 호출할 수 있기 때문이다.
- 이를 위해 안드로이드에서는 내부 바운드 서비스 사용을 위한 방법을 제공한다.

내부 바운드 서비스

- 먼저 AIDL은 필요 없기 때문에 카운트 패키지 내부에서 삭제하고 다음과 같이 서비스 소스를 수정한다.

```
public class CountService extends Service {  
  
    ...  
  
    // ICountService.Stub mBinder = new ICountService.Stub() {  
    //     @Override  
    //     public void getCurCountNumber(ICountServiceCallback callback) throws RemoteException {  
    //  
    //         try { Thread.sleep(10000); }  
    //         catch (InterruptedException e) { e.printStackTrace(); }  
    //  
    //         callback.getCurCountNumberCallback(mCurNum);  
    //     }  
    // };  
  
    public int getCurCountNumber() { return mCurNum; }  
  
    public class LocalBinder extends Binder {  
        CountService getCountService() { return CountService.this; }  
    }  
  
    private final Binder mBinder = new LocalBinder();  
  
    @Nullable  
    @Override  
    public IBinder onBind(Intent intent) {  
        Log.i("superdroid", "onBind()" + intent);  
        return mBinder;  
    }  
  
    @Override  
    public boolean onUnbind(Intent intent) {  
        Log.i("superdroid", "onUnbind()");  
        return super.onUnbind(intent);  
    }  
}
```

내부 바운드 서비스

86

- 카운트 서비스 앱의 액티비티 소스를 수정한다.

```
public class MainActivity extends AppCompatActivity {

    CountService mCountService = null;

    private ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.d("superdroid", "onServiceConnected()");

            mCountService = ((CountService.LocalBinder) service).getCountService();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            Log.d("superdroid", "onServiceDisconnected()");
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // 카운트 서비스 연결
        // =====
        Intent serviceIntent = new Intent();
        ComponentName componentName = new ComponentName("kr.co.company.countservice", "kr.co.company.countservice.CountService");
        serviceIntent.setComponent(componentName);
        bindService(serviceIntent, mConnection, BIND_AUTO_CREATE);
        // =====
    }
}
```

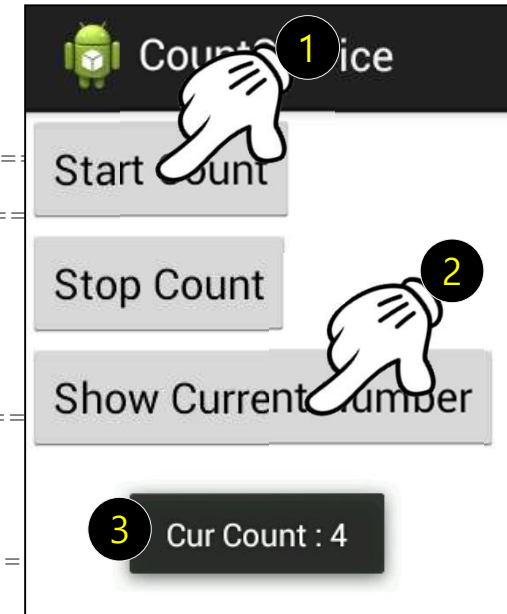
내부 바운드 서비스

```
@Override
protected void onDestroy() {
    // 카운트 서비스 해제
    // =====
    unbindService(mConnection);
    // =====
    super.onDestroy();
}

public void onClick(View v) {
    switch(v.getId()) {
        // 1. 카운트 서비스 시작
        // =====
        case R.id.start_count_btn:
        {
            // CountServiceClient와 동일
        }
        // =====

        // 2. 카운트 서비스 종료
        // =====
        case R.id.stop_count_btn:
        {
            // CountServiceClient와 동일
        }
        // =====

        // 3. 현재까지 카운트된 수치 보기
        // =====
        case R.id.show_cur_number_btn:
        {
            Toast.makeText(this, "Cur Count: " + mCountService.getCurCountNumber(), Toast.LENGTH_LONG).show();
            break;
        }
        // =====
    }
}
}
```



인텐트 서비스와 메신저를 이용한 서비스

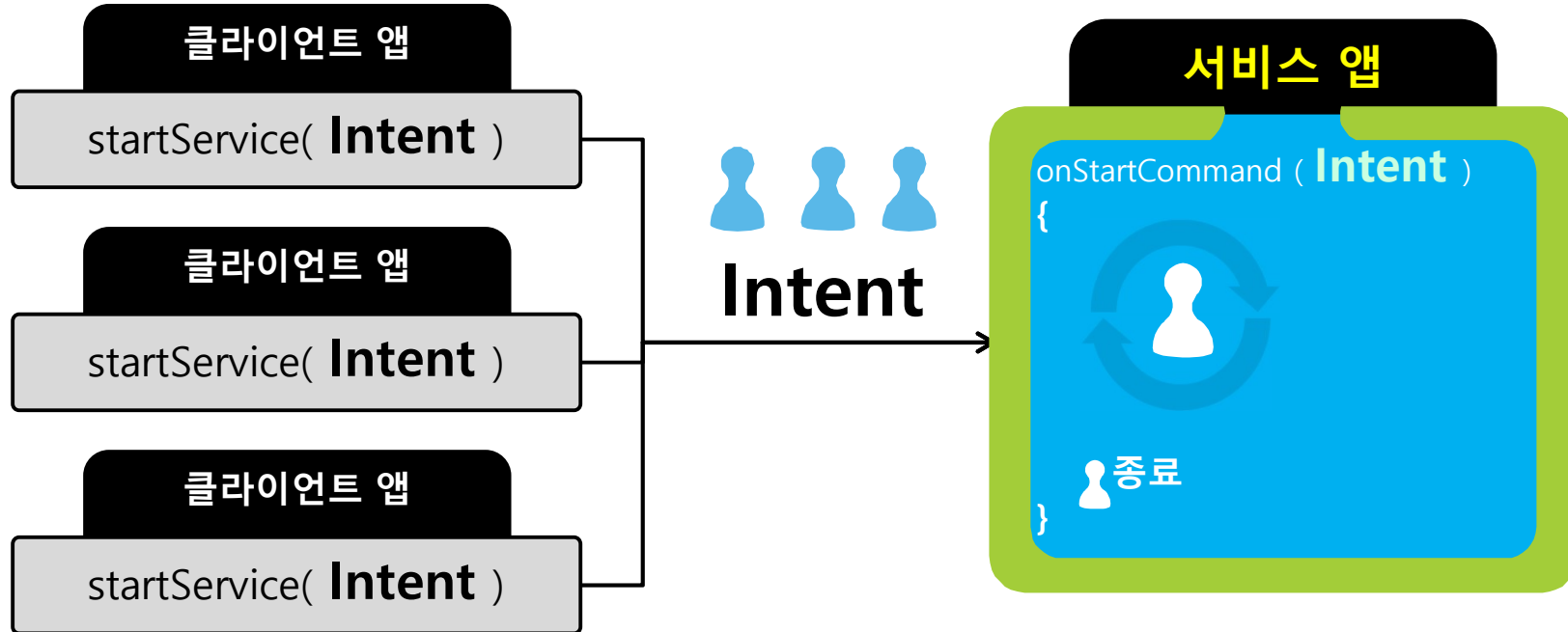
88

- 여러분이 구현하려는 앱의 특성에 따라 안드로이드에서 제공하는 더 편리한 서비스 관련 클래스들을 사용할 수 있다.
- 물론 서비스의 종류는 스타티드와 바운드 형태의 서비스가 전부다.
 - ▣ 다만 설명하려는 클래스들은 이 두 가지 형태의 서비스를 더 편하게 구현하도록 도와준다.

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

89

- 인텐트 서비스 `IntentService`는 스타티드 서비스의 파생 클래스다.
 - ▣ 즉, 바운드 서비스와 같이 인터페이스를 제공하는 것이 아니라 특정 동작을 수행하는 데 목적이 있다.



- 동작을 인텐트 단위별로 순서대로 동작시켜야 하는 환경에서 매우 유용하다.

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

90

- 인텐트 서비스를 이용하지 않는 카운트다운 서비스 구현

- 먼저 새로운 패키지를 생성하자.

- Application name: IntentService

- activity_main.xml

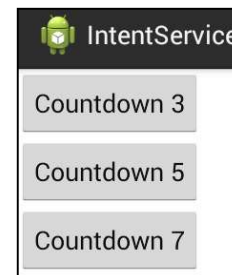
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
<Button android:id="@+id/countdown_3_btn"
    android:text="Countdown 3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
<Button android:id="@+id/countdown_5_btn"
    android:text="Countdown 5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
<Button android:id="@+id/countdown_7_btn"
    android:text="Countdown 7"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
</LinearLayout>
```



● 레이아웃 구조

- └─ LinearLayout
 - └─ countdown_3_btn (Button)
 - └─ countdown_5_btn (Button)
 - └─ countdown_7_btn (Button)

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

91

□ CountdownService.java

```
public class CountdownService extends Service {  
    int mCountdownNum = 0;
```

```
    @Override  
    public void onCreate() {  
        super.onCreate();  
        Log.d("superdroid", "onCreate()");  
    }
```

```
    @Override  
    public int onStartCommand(Intent intent, int flags, final int startId) {  
        super.onStartCommand(intent, flags, startId);  
  
        mCountdownNum = intent.getExtras().getInt("COUNTDOWN_NUM");  
  
        Log.d("superdroid", "onStartCommand(startID " + startId + ") Num: " + mCountdownNum);  
  
        new Thread("Countdown Thread") {  
            @Override  
            public void run() {  
                while (mCountdownNum > 0) {  
                    Log.i("superdroid", "start Id " + startId + " " + "Countdown: " + mCountdownNum);  
  
                    mCountdownNum--;  
  
                    try { Thread.sleep(1000); }  
                    catch (InterruptedException e) { break; }  
                }  
  
                Log.d("superdroid", "stopSelf: " + startId);  
                stopSelf(startId);  
            }  
        }.start();  
  
        return START_STICKY;  
    }
```

```
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        Log.d("superdroid", "onDestroy()");  
    }
```

```
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

92

□ MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.countdown_3_btn: {
                Intent serviceIntent = new Intent(this, CountdownService.class);
                serviceIntent.putExtra("COUNTDOWN_NUM", 3);
                startService(serviceIntent);
                break;
            }

            case R.id.countdown_5_btn: {
                Intent serviceIntent = new Intent(this, CountdownService.class);
                serviceIntent.putExtra("COUNTDOWN_NUM", 5);
                startService(serviceIntent);
                break;
            }

            case R.id.countdown_7_btn: {
                Intent serviceIntent = new Intent(this, CountdownService.class);
                serviceIntent.putExtra("COUNTDOWN_NUM", 7);
                startService(serviceIntent);
                break;
            }
        }
    }
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

93



● 로그

`onCreate()`

`onStartCommand(startID 1) Num : 3`

`start Id 1 Countdown : 3`

`start Id 1 Countdown : 2`

`start Id 1 Countdown : 1`

`stopSelf : 1`

`onDestroy()`

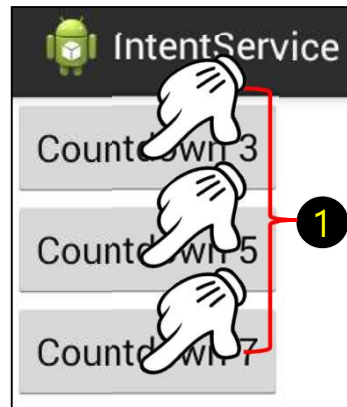
3

4

2

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

94



● 로그

```
onCreate()
onStartCommand(startID 1) Num : 7
start Id 1 Countdown : 7
onStartCommand(startID 2) Num : 5
start Id 2 Countdown : 5
onStartCommand(startID 3) Num : 3
start Id 3 Countdown : 3
start Id 1 Countdown : 2
start Id 2 Countdown : 1
stopSelf : 3
onDestroy()
stopSelf : 1
stopSelf : 2
```

- 이렇게 동시에 처리되는 서비스 동기화는 여간 귀찮은 문제가 아니다. 하지만 인텐트 서비스를 이용하면 매우 간단히 동시에 요청되는 서비스를 순차적으로 처리할 수 있다.
- **stopSelf(int startId) 메소드를 통한 서비스 종료**
 - ▣ 서비스가 발급한 최종 서비스 ID를 통해 서비스를 종료

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

95

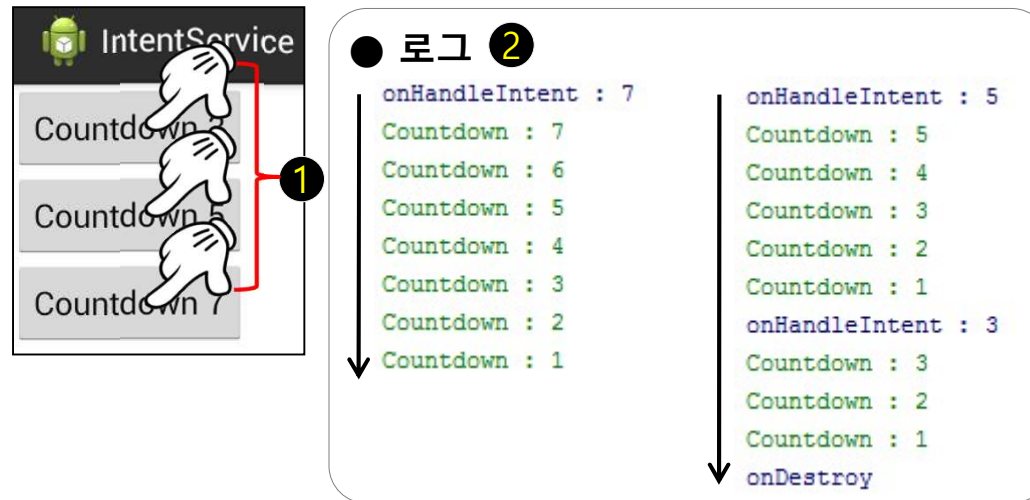
□ 인텐트 서비스를 이용한 카운트다운 서비스 구현

```
public class CountdownService extends IntentService {  
    int mCountdownNum = 0;  
  
    public CountdownService() {  
        super("Countdown Service");  
    }  
  
    @Override  
    protected void onHandleIntent(@Nullable Intent intent) {  
        mCountdownNum = intent.getExtras().getInt("COUNTDOWN_NUM");  
  
        Log.d("superdroid", "onHandleIntent: " + mCountdownNum);  
  
        while (mCountdownNum > 0) {  
            Log.i("superdroid", "Countdown: " + mCountdownNum);  
  
            mCountdownNum--;  
  
            try { Thread.sleep(1000); }  
            catch (InterruptedException e) { break; }  
        }  
    }  
  
    @Override  
    public void onDestroy() {  
        super.onDestroy();  
        Log.d("superdroid", "onDestroy()");  
    }  
}
```

생성자도 추가해야 한다. 그렇지 않으면 'no empty constructor exception'이 발생한다. super의 생성자에 들어가는 파라미터는 백그라운드 스레드의 이름으로 사용된다.

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

96



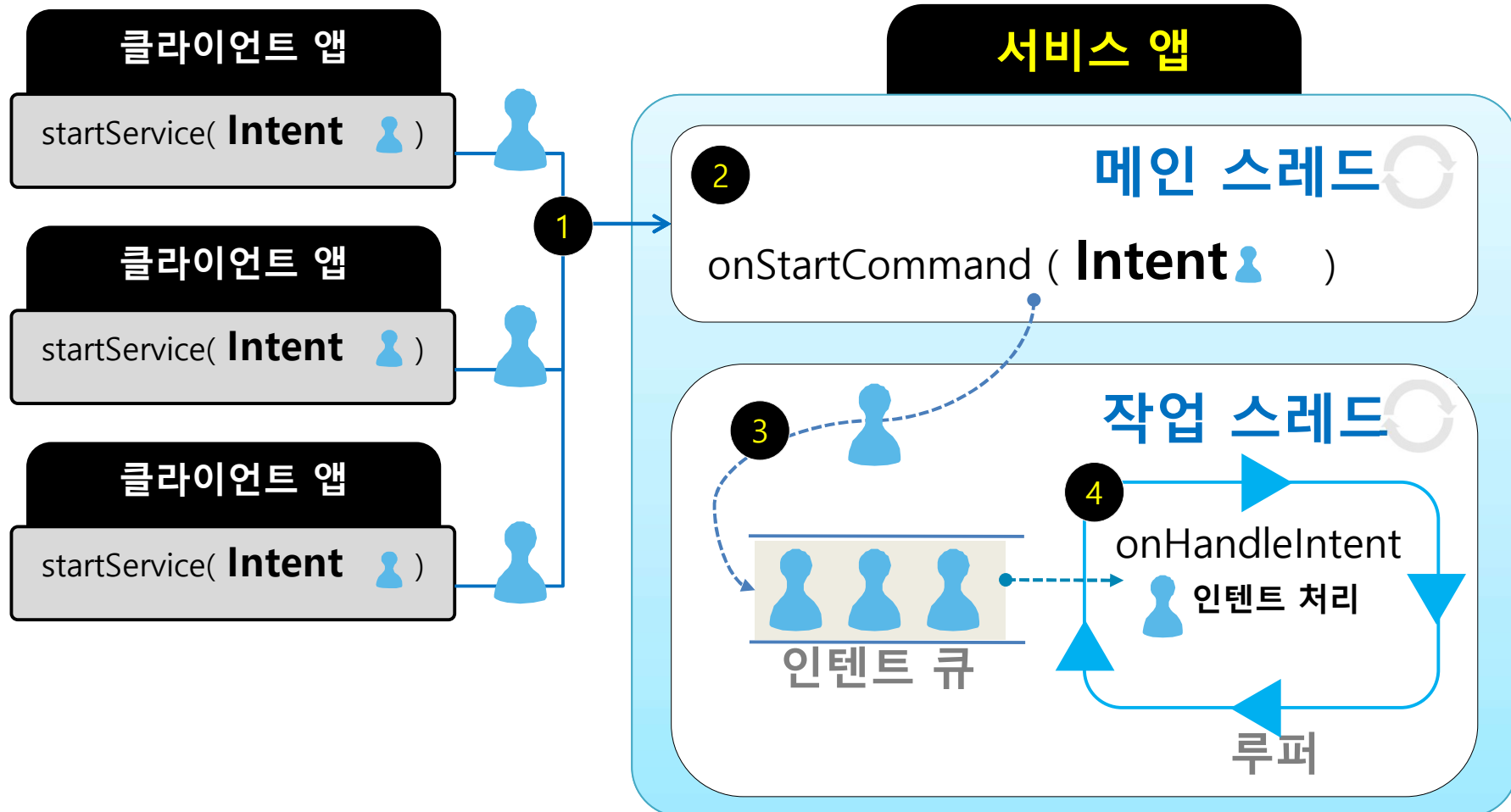
■ 너무 간단하지 않은가!

onHandleIntent 메소드는 별도의 스레드에서 동작하기 때문에 따로 스레드를 생성할 필요도 없다.

인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 **인텐트 서비스**

97

- 인텐트 서비스는 어떤 구조이기에 이러한 처리가 가능한지 내부 구조를 잠시 살펴보자.



- 인텐트 서비스에서도 내부적으로 HandlerThread를 사용한다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

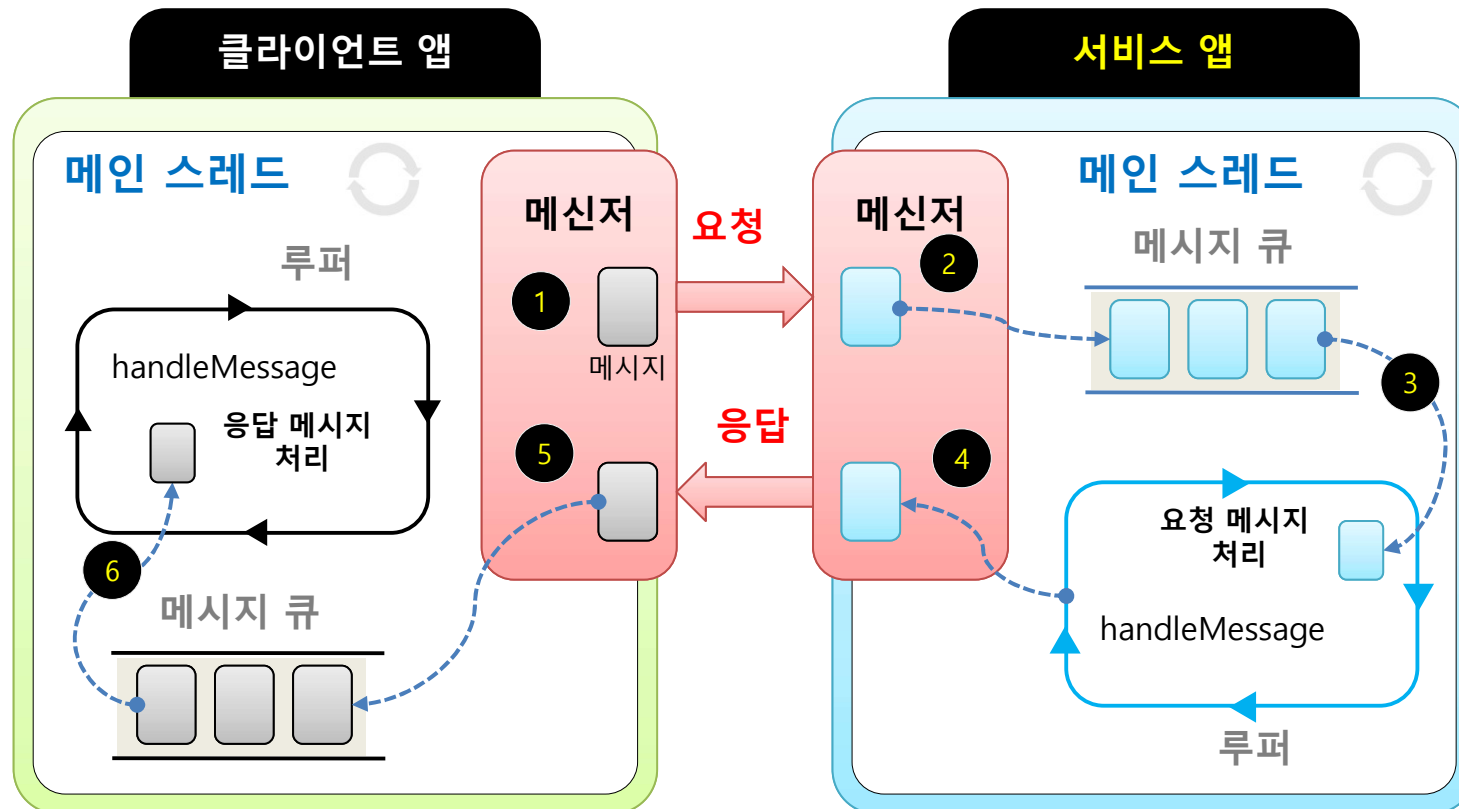
98

- 메신저 서비스를 이용하면 바운드 형태의 서비스를 좀 더 쉽게 사용할 수 있다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

99

- 메신저(Messenger)의 가장 큰 장점은 AIDL 파일 없이 다른 프로세스 간 통신이 가능하다는 것과 인텐트 서비스와 같이 여러 클라이언트 요청에 대해 순서대로 처리하기 때문에 동기화에 대한 신경을 쓸 필요가 없는 것이다.



각 앱 메인 스레드의 큐에 메시지를 전달하는 메신저

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

100

- 두 수의 합과 차를 계산하는 메신저 서비스 구현
- 먼저 새로운 패키지를 생성하자.
 - Application name: Messenger
- AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.iamjw.messenger">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".MessengerService"
            android:enabled="true"
            android:exported="true"></service>
    </application>

</manifest>
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

101

```
public class MessengerService extends Service {  
    // 요청 메시지 코드 정의  
    // =====  
    static final int REQUEST_MSG_PLUS_VALUE = 1001;  
    static final int REQUEST_MSG_MINUS_VALUE = 1002;  
    // =====  
  
    // 응답 메시지 코드 정의  
    // =====  
    static final int RESPONSE_MSG_PLUS_VALUE = 2001;  
    static final int RESPONSE_MSG_MINUS_VALUE = 2002;  
    // =====  
  
    private MsgRequestHandler mMsgRequestHandler = null;  
    private Messenger mMessengerService = null;
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

```
private class MsgRequestHandler extends Handler {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            // 두 수의 합을 계산하고 결과를 클라이언트에게 전달한다.
            // =====
            case REQUEST_MSG_PLUS_VALUE:
            {
                int responsePlusValue = msg.arg1 + msg.arg2;

                Log.i("superdroid", "plus result: " + responsePlusValue);

                Message replyMsg = Message.obtain(null,
                    RESPONSE_MSG_PLUS_VALUE,
                    responsePlusValue, 0);

                try { msg.replyTo.send(replyMsg); }
                catch (RemoteException e) { e.printStackTrace(); }

                break;
            }
            // =====
            // 두 수의 차를 계산하고 결과를 클라이언트에게 전달한다.
            // =====
            case REQUEST_MSG_MINUS_VALUE:
            {
                int responseMinusValue = msg.arg1 - msg.arg2;

                Log.i("superdroid", "Minus result : " + responseMinusValue);

                Message replyMsg = Message.obtain(null,
                    RESPONSE_MSG_MINUS_VALUE,
                    responseMinusValue, 0);

                try { msg.replyTo.send(replyMsg); }
                catch (RemoteException e) { e.printStackTrace(); }

                break;
            }
            // =====
        }
    }
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

103

@Override

```
public void onCreate() {  
    super.onCreate();
```

```
    // 요청 메시지를 처리할 핸들러 객체를 생성한다.
```

```
    mMsgRequestHandler = new MsgRequestHandler();
```

```
    // 메신저 객체를 생성하고, 메신저가 메시지를 수신받았을 때 메시지 큐에 추가하여  
    // 요청을 처리하도록 핸들러를 객체를 생성자로 전달한다.
```

```
    mMessengerService = new Messenger(mMsgRequestHandler);
```

```
}
```

@Override

```
public IBinder onBind(Intent intent) {
```

```
    // 클라이언트와 연결되면 통신을 위한 바인더 객체를 클라이언트에게 전달한다.
```

```
    return mMessengerService.getBinder();
```

```
}
```

```
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

104

□ 정말 메신저는 AIDL 파일 없이 프로세스 통신을 하는 것일까?

- 사실 안드로이드 프레임워크 내부에는 AIDL 파일이 존재한다.

```
package android.os;
import android.os.Message;

/** @hide */
oneway interface IMessenger
{
    void send(in Message msg);
}
```

- 메신저의 AIDL은 send라는 단 하나의 메소드를 정의한다.
- 메신저는 이 인터페이스를 통해 메시지를 서비스와 클라이언트 간에 주고받는다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

105

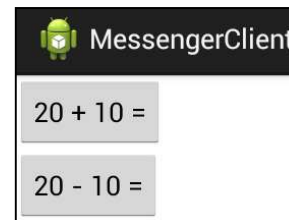
- 메신저 서비스를 사용할 클라이언트 구현
- 먼저 새로운 패키지를 생성하자.
 - Application name: MessengerClient
 - activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
<Button android:id="@+id/plus_btn"
    android:text="20 + 10 ="
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
<Button android:id="@+id/minus_btn"
    android:text="20 - 10 ="
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"/>
```

```
</LinearLayout>
```



● 레이아웃 구조

```
LinearLayout
├── plus_btn (Button) - "20 + 10 ="
└── minus_btn (Button) - "20 - 10 ="
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

106

```
public class MainActivity extends AppCompatActivity {
```

```
    // 본 코드 값은 서비스 측과 동일하게 설정되어야 한다.
```

```
    // =====
```

```
    // 요청 메시지 코드 정의
```

```
        static final int REQUEST_MSG_PLUS_VALUE = 1001;
```

```
        static final int REQUEST_MSG_MINUS_VALUE = 1002;
```

```
    // 응답 메시지 코드 정의
```

```
        static final int RESPONSE_MSG_PLUS_VALUE = 2001;
```

```
        static final int RESPONSE_MSG_MINUS_VALUE = 2002;
```

```
    // =====
```

```
    private Messenger mMessengerService;
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

107

```
private ServiceConnection mConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        mMessengerService = new Messenger(service);
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        mMessengerService = null;
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Intent serviceIntent = new Intent();
    serviceIntent.setClassName("com.example.iamjw.messenger",
        "com.example.iamjw.messenger.MessengerService");
    bindService(serviceIntent, mConnection, BIND_AUTO_CREATE);
}

@Override
protected void onDestroy() {
    unbindService(mConnection);

    super.onDestroy();
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

108

// 클라이언트의 메신저 객체, 서비스에게 전달하여 응답 결과를 받기 위한 메신저이다.

```
private final Messenger mMessengerResponse = new Messenger(mMessengerResponseHandler);
```

```
public void onClick(View v) {  
    switch (v.getId()) {  
        case R.id.plus_btn:  
        {  
            try {  
                Message msg = Message.obtain(null, REQUEST_MSG_PLUS_VALUE, 20, 10);  
                msg.replyTo = mMessengerResponse;  
                mMessengerService.send(msg);  
            } catch (RemoteException e) { e.printStackTrace(); }  
  
            break;  
        }  
  
        case R.id.minus_btn:  
        {  
            try {  
                Message msg = Message.obtain(null, REQUEST_MSG_MINUS_VALUE, 20, 10);  
                msg.replyTo = mMessengerResponse;  
                mMessengerService.send(msg);  
            } catch (RemoteException e) { e.printStackTrace(); }  
  
            break;  
        }  
    }  
}
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

109

```
private Handler mMessengerResponseHandler = new Handler() {
```

```
    @Override
```

```
    public void handleMessage(Message msg) {
```

```
        switch (msg.what) {
```

```
            // 두 수의 합의 결과 처리
```

```
            // =====
```

```
            case RESPONSE_MSG_PLUS_VALUE:
```

```
            {
```

```
                Toast.makeText(MainActivity.this,
```

```
                    "20 + 10 = " + msg.arg1,
```

```
                    Toast.LENGTH_LONG).show();
```

```
                break;
```

```
            }
```

```
            // =====
```

```
            // 두 수의 차의 결과 처리
```

```
            // =====
```

```
            case RESPONSE_MSG_MINUS_VALUE:
```

```
            {
```

```
                Toast.makeText(MainActivity.this,
```

```
                    "20 - 10 = " + msg.arg1,
```

```
                    Toast.LENGTH_LONG).show();
```

```
                break;
```

```
            }
```

```
            // =====
```

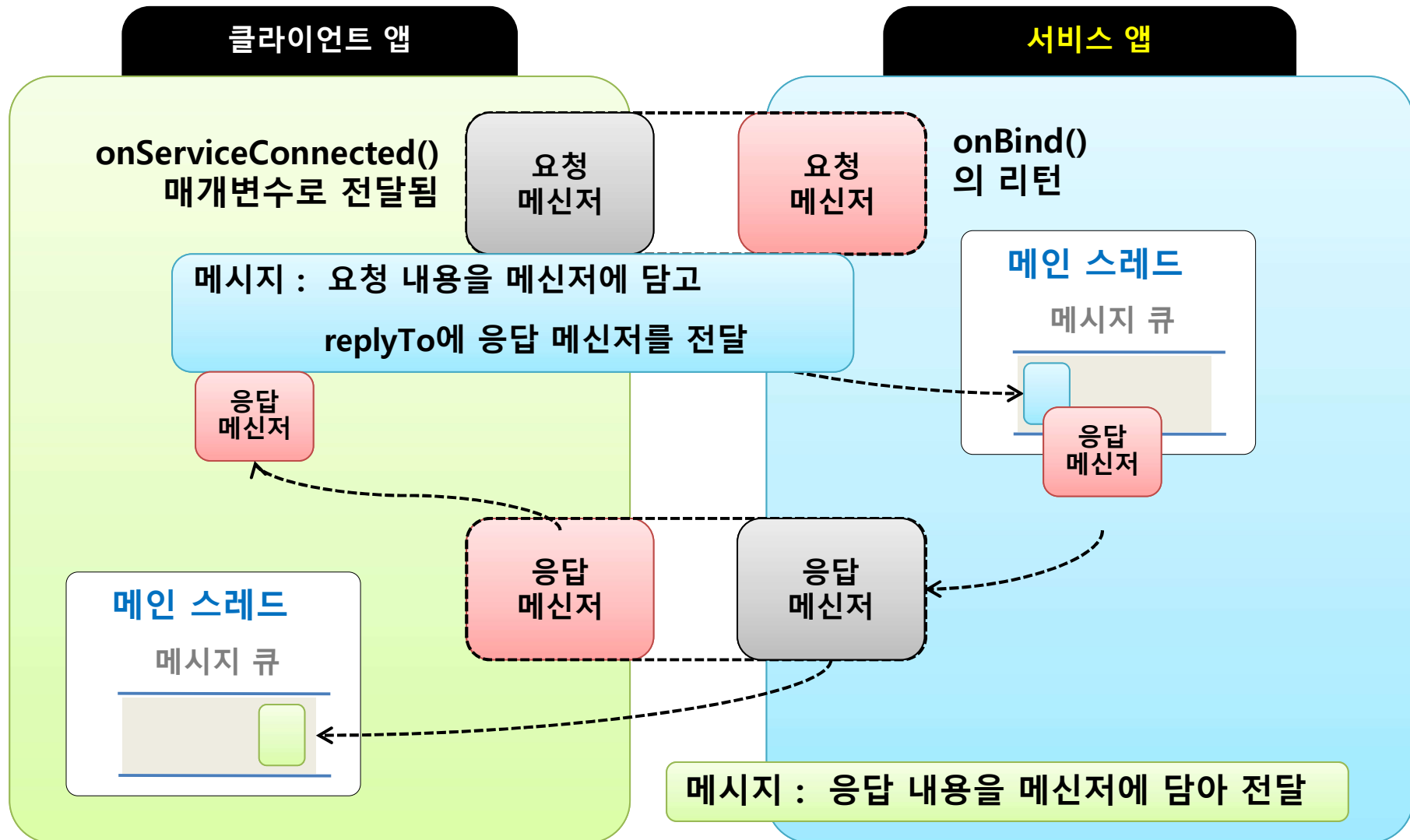
```
        }
```

```
    }
```

```
};
```

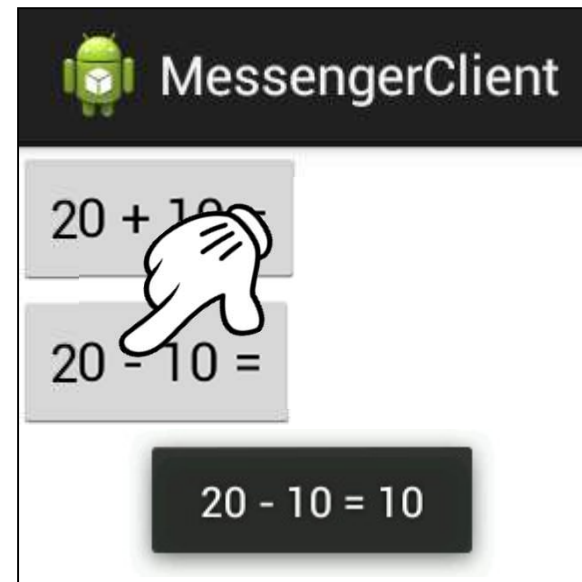
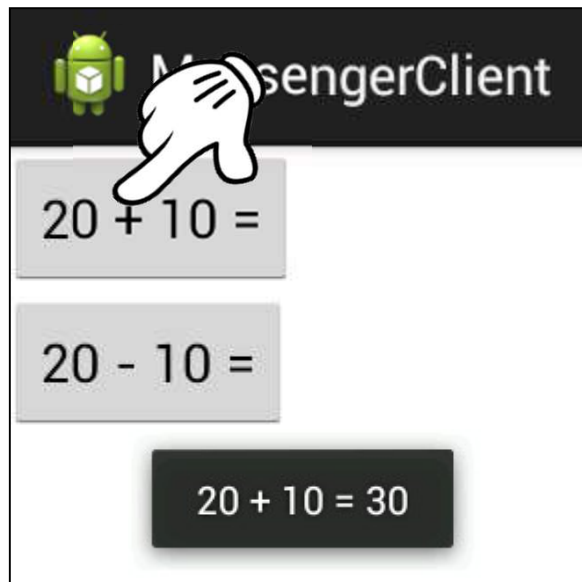
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

110



인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

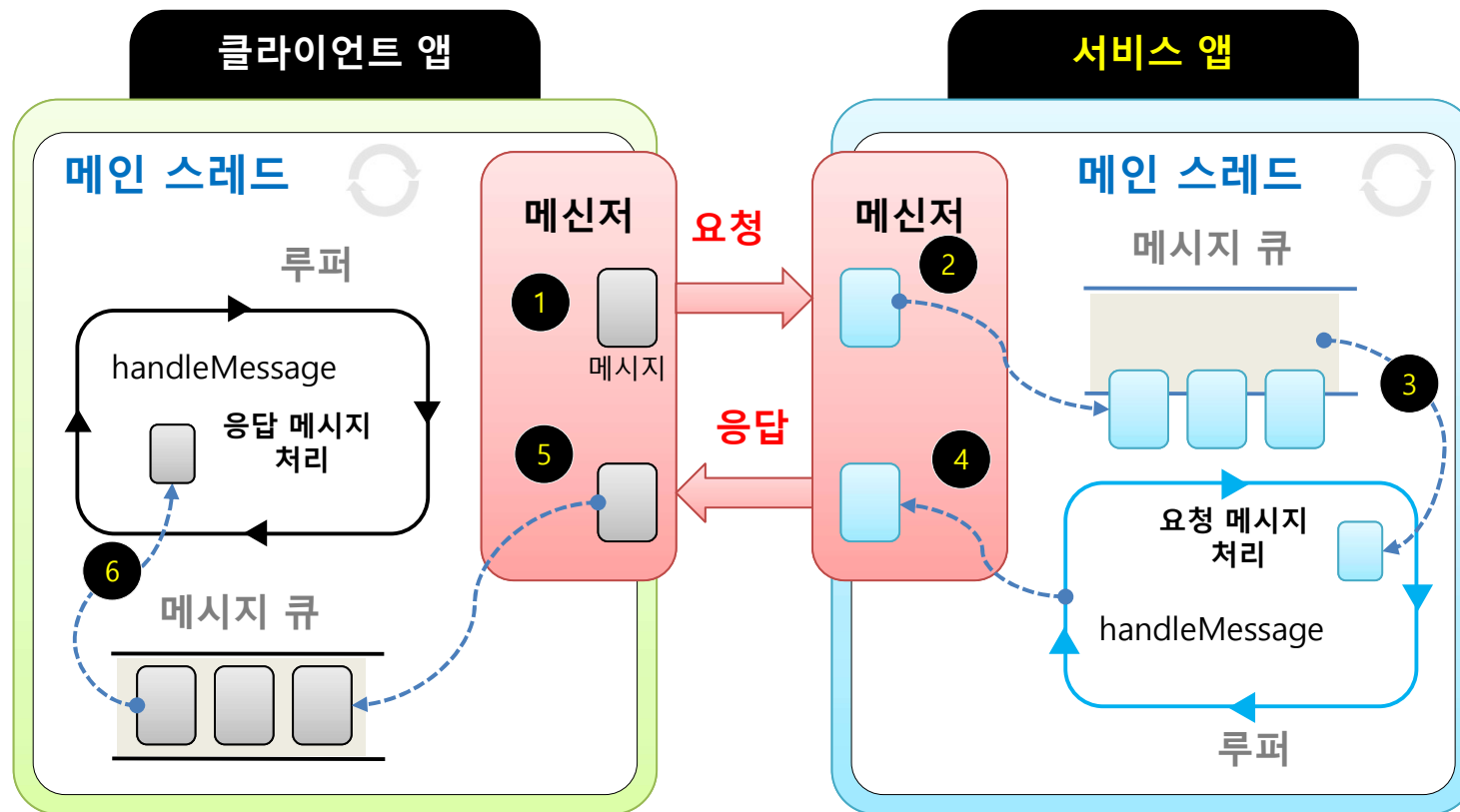
111



인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

112

- HandlerThread를 이용한 안전한 메신저 서비스 구현



서비스는 메인 스레드에서 동작한다.

따라서 서비스 제한 시간인 20초 이상의 작업을 수행한다면 ANR이 발생된다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

113

- 서비스는 최대한 메인 스레드가 아닌 작업 스레드를 이용하는 것이 안전하다.
- 하지만 메신저는 메시지 큐와 루퍼 기반으로 동작하며, 이러한 환경의 작업 스레드를 만드는 것은 쉽지 않다.
- 이를 위해 안드로이드에서는 스레드를 상속받은 HandlerThread라는 클래스를 제공하는데, 이 클래스는 메시지 큐와 루퍼를 기반으로 동작한다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

114

- 서비스는 최대한 메인 스레드가 아닌 작업 스레드를 이용하는 것이 안전하다.
- 하지만 메신저는 메시지 큐와 루퍼 기반으로 동작하며, 이러한 환경의 작업 스레드를 만드는 것은 쉽지 않다.
- 이를 위해 안드로이드에서는 스레드를 상속받은 HandlerThread라는 클래스를 제공하는데, 이 클래스는 메시지 큐와 루퍼를 기반으로 동작한다.

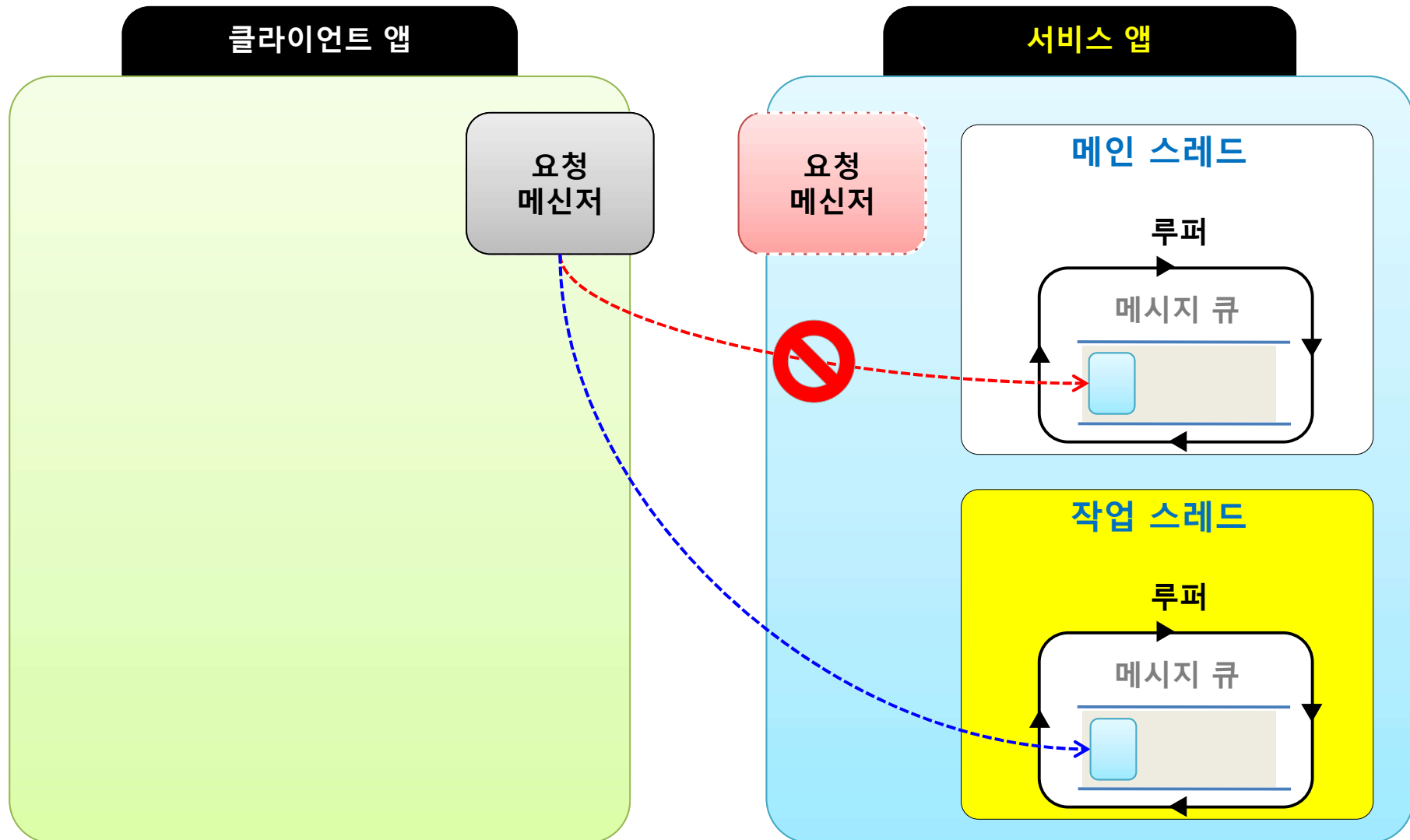
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

115

```
public class MessengerService extends Service {  
    ...  
  
    private class MsgRequestHandler extends Handler {  
        public MsgRequestHandler(Looper looper) {  
            super(looper);  
        }  
    }  
    ...  
  
    @Override  
    public void onCreate() {  
        super.onCreate();  
  
        HandlerThread thread = new HandlerThread("Messenger Service");  
        thread.start();  
  
        mMsgRequestHandler = new MsgRequestHandler(thread.getLooper());  
  
        mMessengerService = new Messenger(mMsgRequestHandler);  
    }  
  
    @Override  
    public void onDestroy() {  
        mMsgRequestHandler.getLooper().quit();  
  
        super.onDestroy();  
    }  
    ...  
}
```

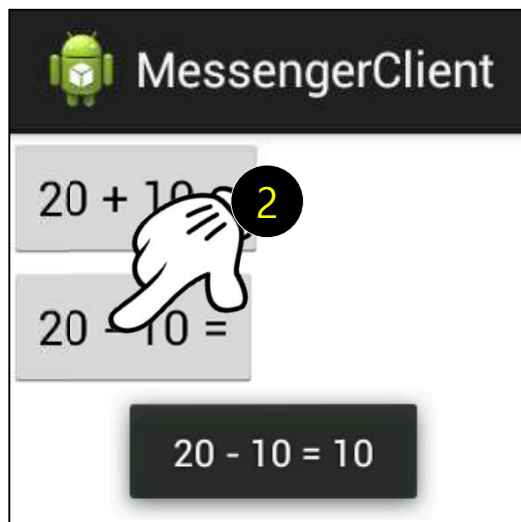
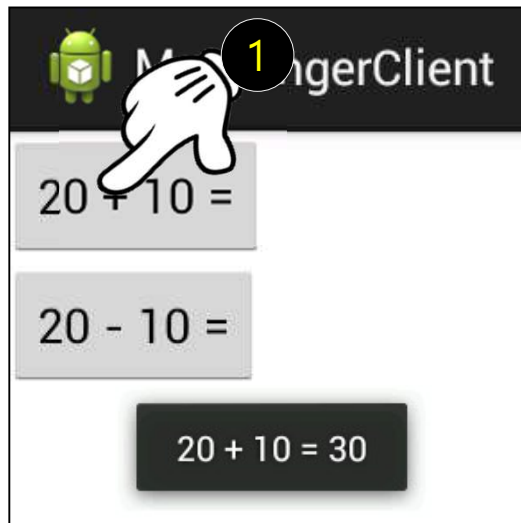
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

116



인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

117



● DDMS에서 스레드 확인

Threads					
File Explorer					
Heap					
Allocation Tracker					
ID	Tid	Status			
1	1840	Native			
*2	1843	VmWait			
*3	1845	VmWait			
*4	1849	Runnable			
*5	1853	Wait			
*6	1854	Wait	0	0	ReferenceQueueDaemon
*7	1855	Wait	0	0	FinalizerDaemon
*8	1856	Wait	0	0	FinalizerWatchdogDaemon
9	1857	Native	1	0	Binder_1
10	1858	Native	0	0	Binder_2
11	2337	Native	0	0	Messenger Service

● 로그

TID	Text
2337	plus result : 30
2337	Minus result : 10

단말기에서 실행 중인 서비스 정보 보기

118

❑ adb shell dumpsys activity services

ACTIVITY MANAGER SERVICES (dumpsys activity services)

Active services:

...

* **ServiceRecord**{49dfc60 u0 com.example.iamjw.messenger/.MessengerService}

intent={cmp=com.example.iamjw.messenger/.MessengerService}

packageName=com.example.iamjw.messenger

processName=com.example.iamjw.messenger

baseDir=/data/app/com.example.iamjw.messenger-2/base.apk

dataDir=/data/user/0/com.example.iamjw.messenger

app=ProcessRecord{59cf08f 8188:com.example.iamjw.messenger/u0a80}

createTime=-30s562ms startingBgTimeout=--

lastActivity=-30s562ms restartTime=-30s562ms createdFromFg=true

Bindings:

* **IntentBindRecord**{2b82f1c CREATE}:

intent={cmp=com.example.iamjw.messenger/.MessengerService}

binder=android.os.BinderProxy@26ec725

requested=true received=true hasBound=true doRebind=false

* **Client AppBindRecord**{57640fa ProcessRecord{2ca241d 8272:com.example.iamjw.messengerclient/u0a81}}

Per-process Connections:

ConnectionRecord{6d85863 u0 CR com.example.iamjw.messenger/.MessengerService:@2806692}

All Connections:

ConnectionRecord{6d85863 u0 CR com.example.iamjw.messenger/.MessengerService:@2806692}

* ServiceRecord{...