바운드 서비스 – 서비스 연결

- 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.

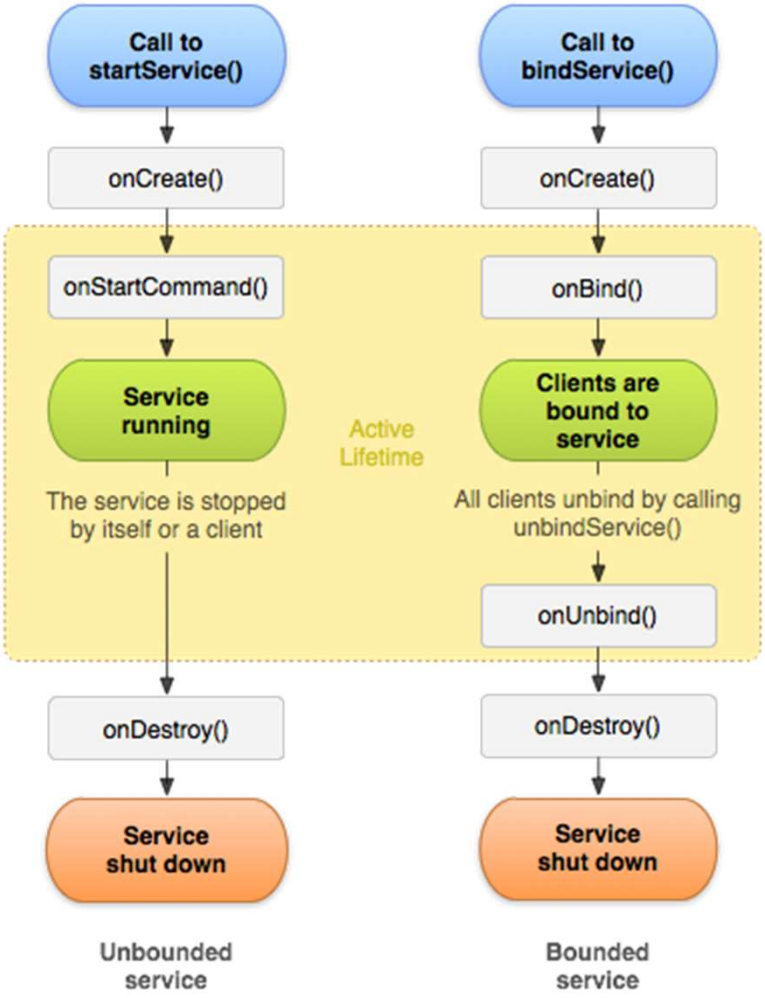
바운드 서비스 – 서비스 연결

- 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

- 스타티드와 바운드 서비스의 생명주기



바운드 서비스 – 서비스 연결

- 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: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>
   </application>

</manifest>
```

바운드 서비스 – 서비스 연결

- activiy_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 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

- 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

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, onDestory 생명주기 함수에 서비스 연결과 해제 처리를 할까?

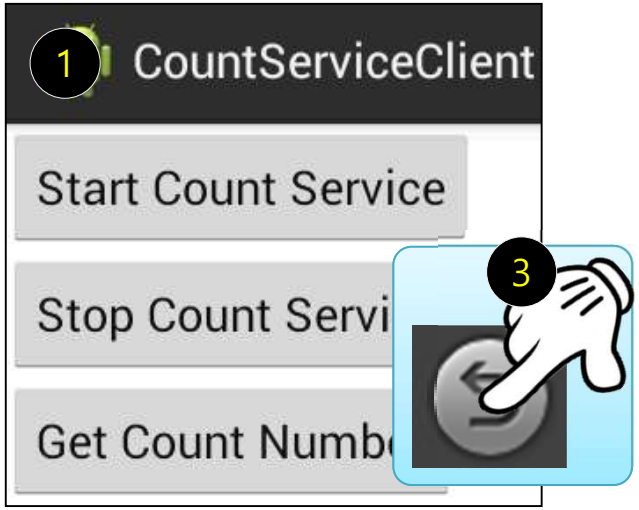
바운드 서비스는 대부분 액티비티 생명주기 안에서 지속적으로 사용되는 경우가 대부분이다. 따라서 서비스 연결과 해제를 액티비티 생명주기에 맞추는 것이 편리하다.

바운드 서비스 – 서비스 연결

- 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



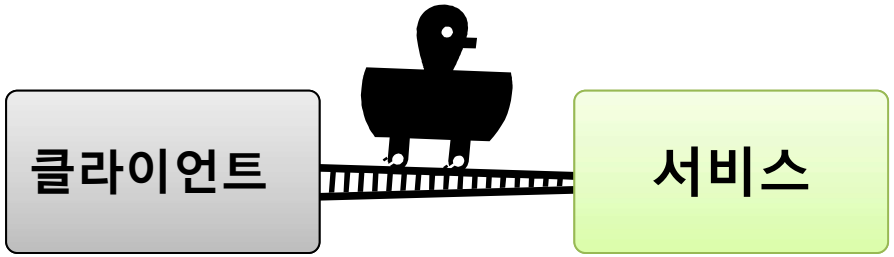
● 로그 확인

2 onCreate()
onBind()

4 onUnbind()
onDestroy()

onServiceConnected
onServiceDisconnected

실질적으로 통신하게 될 매개체가 없다.

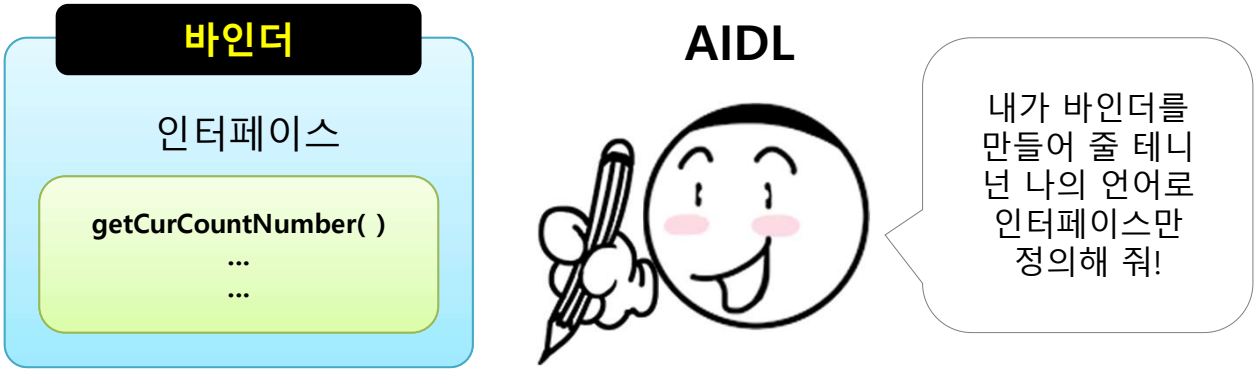


■ 안드로이드에서는 이렇게 통신을 가능케 하는 매개체를 바인더라고 부른다.

바운드 서비스 – 바인더와 AIDL

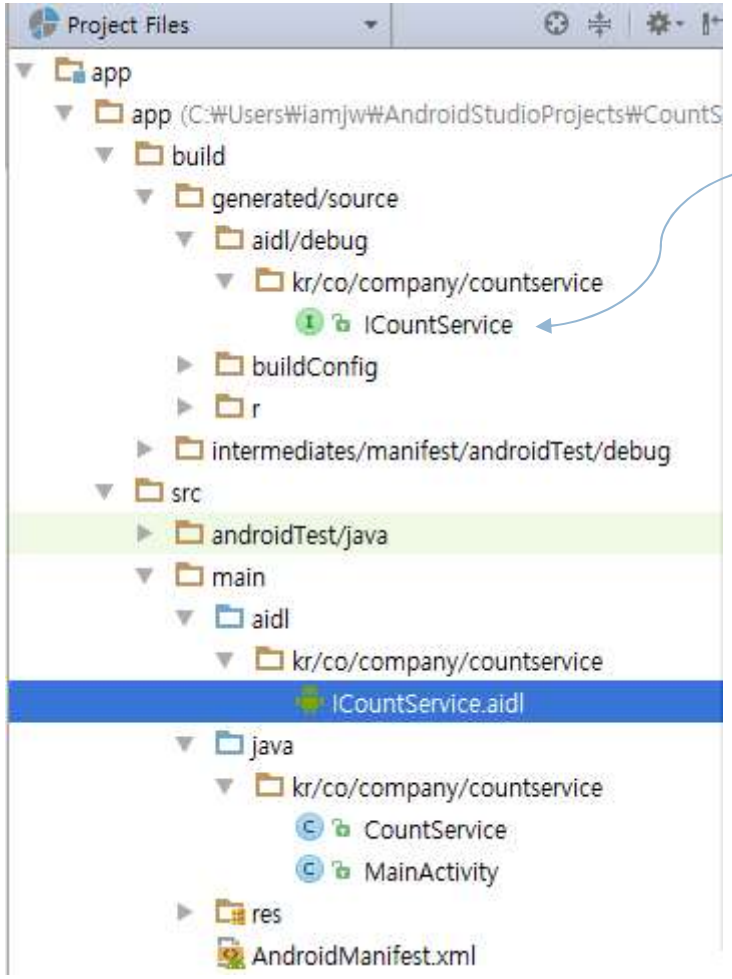
- 바인더는 서비스에 존재하는 인터페이스를 포함하며,
  클라이언트는 바인더를 이용해서 서비스의 인터페이스를 호출할 수 있다.

- 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




```
/.../
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.company.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();
}
```

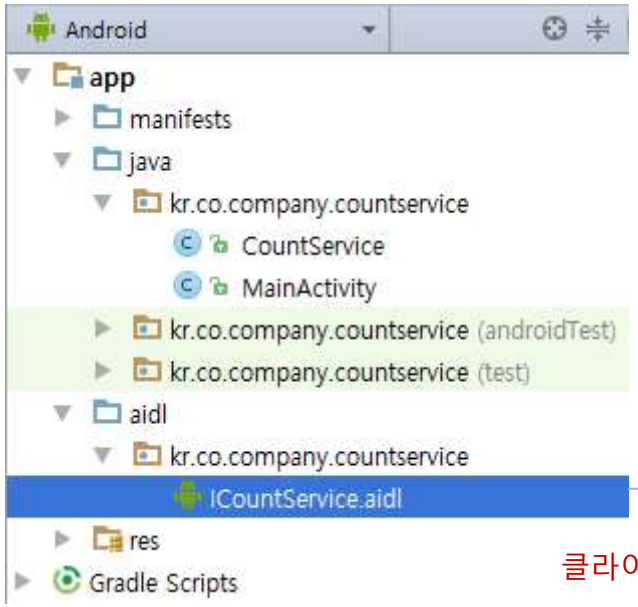
바운드 서비스 – 바인더와 AIDL


```
public class CountService extends Service {

    ...

    ICountService.Stub mBinder = new ICountService.Stub() {
        @Override
        public int getCurCountNumber() throws RemoteException {
            return mCurNum;
        }
    };

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

    ...
}
```

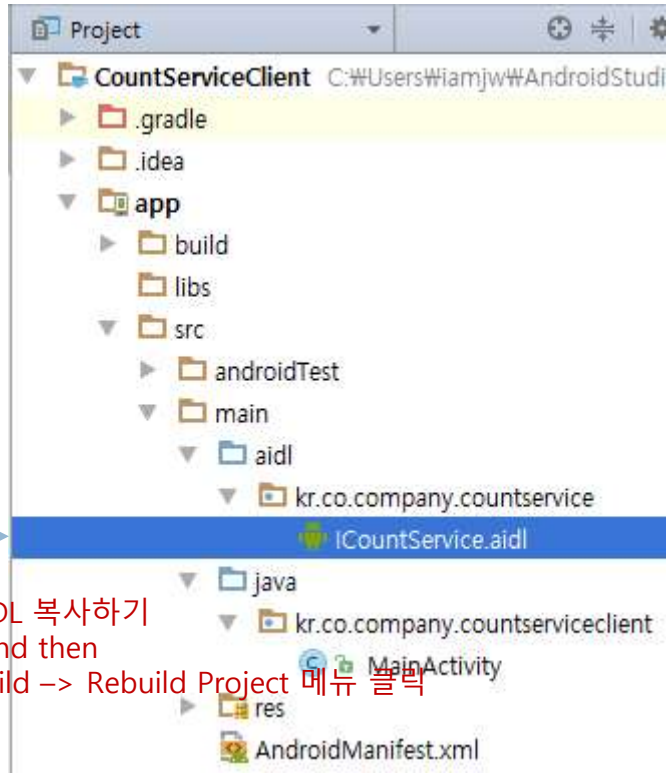
클라이언트는 전달받은 바인터를 통해 서비스에 정의된 인터페이스를 사용할 수 있다.

바운드 서비스 – 바인더와 AIDL

● 카운트 서비스 프로젝트



● 카운트 서비스 클라이언트 프로젝트

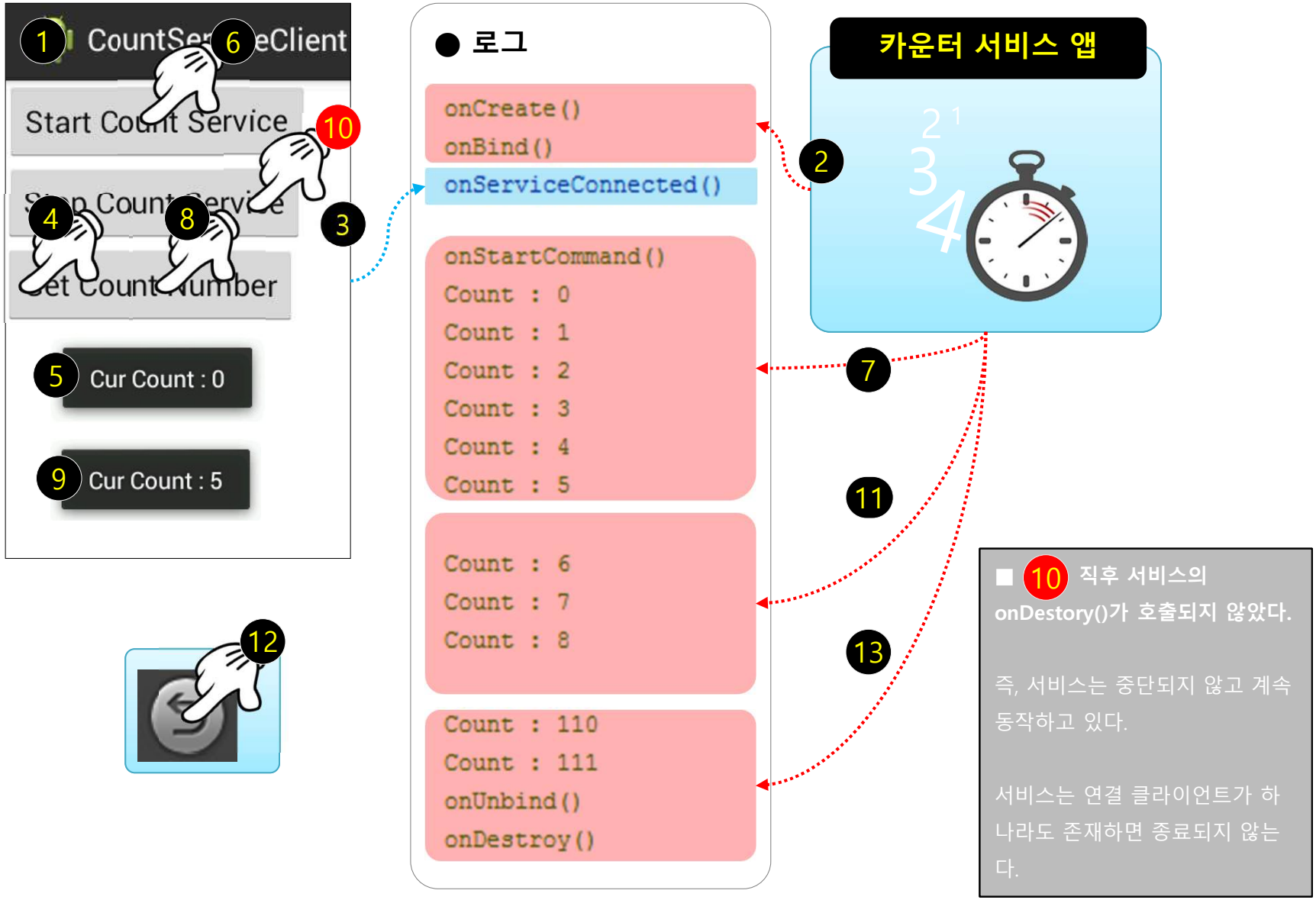


클라이언트 프로젝트로 AIDL 복사하기
and then
안드로이드 스튜디오의 Build –> Rebuild Project 메뉴 클릭

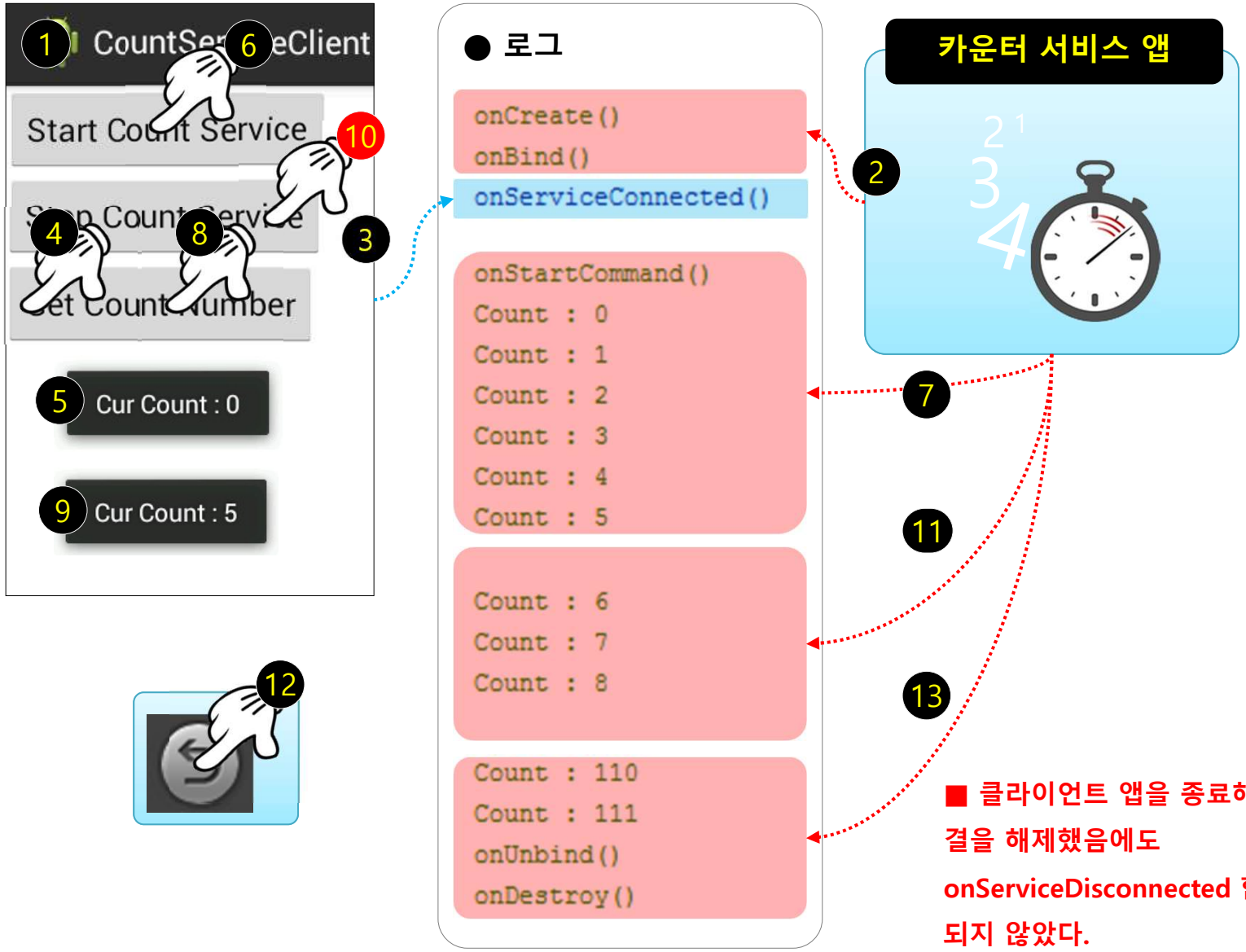
바운드 서비스 – 바인더와 AIDL

```
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



■ 클라이언트 앱을 종료해 서비스 연결을 해제했음에도 onServiceDisconnected 함수가 호출되지 않았다.

바운드 서비스 – BIND_AUTO_CREATE 플래그

- 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

- 하지만 bindService() 메소드의 인자로 BIND_AUTO_CREATE 플래그를 전달하면 스타티드 서비스가 실행되지 않은 상태에서도 바운드 서비스의 사용이 가능해진다.

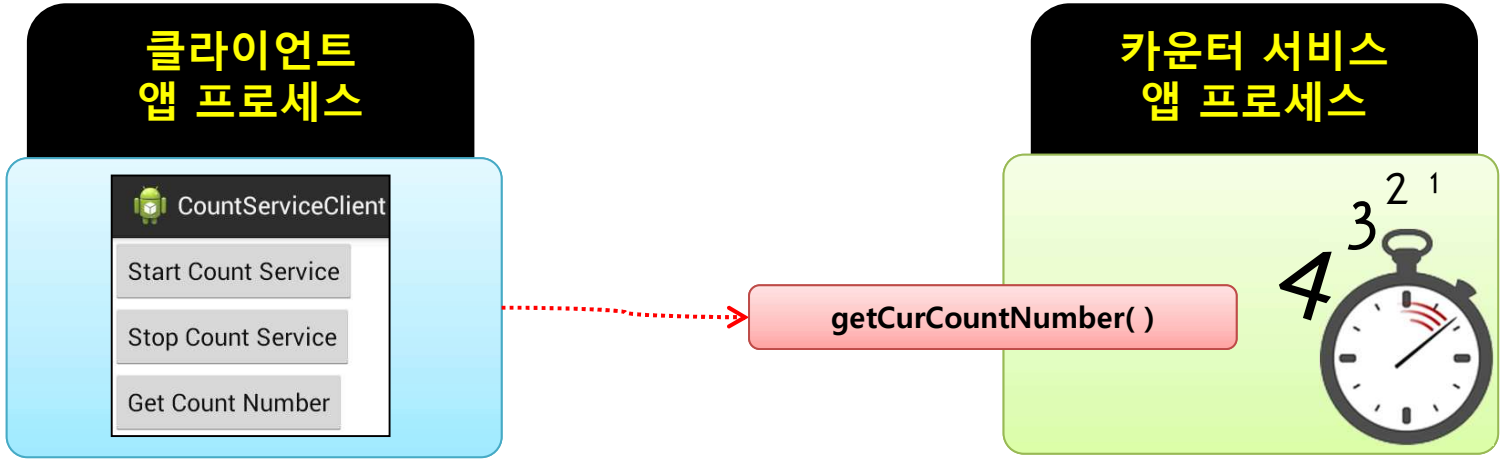
- **따라서 스타티드 & 바운드 서비스가 아니라면 BIND_AUTO_CREATE 옵션은 필수적이다.**
  - **이렇게 분리되어 독립적으로 동작하는 바운드 서비스는 스타티드 서비스와 상관없이 언제든지 사용할 수 있다.**

- 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

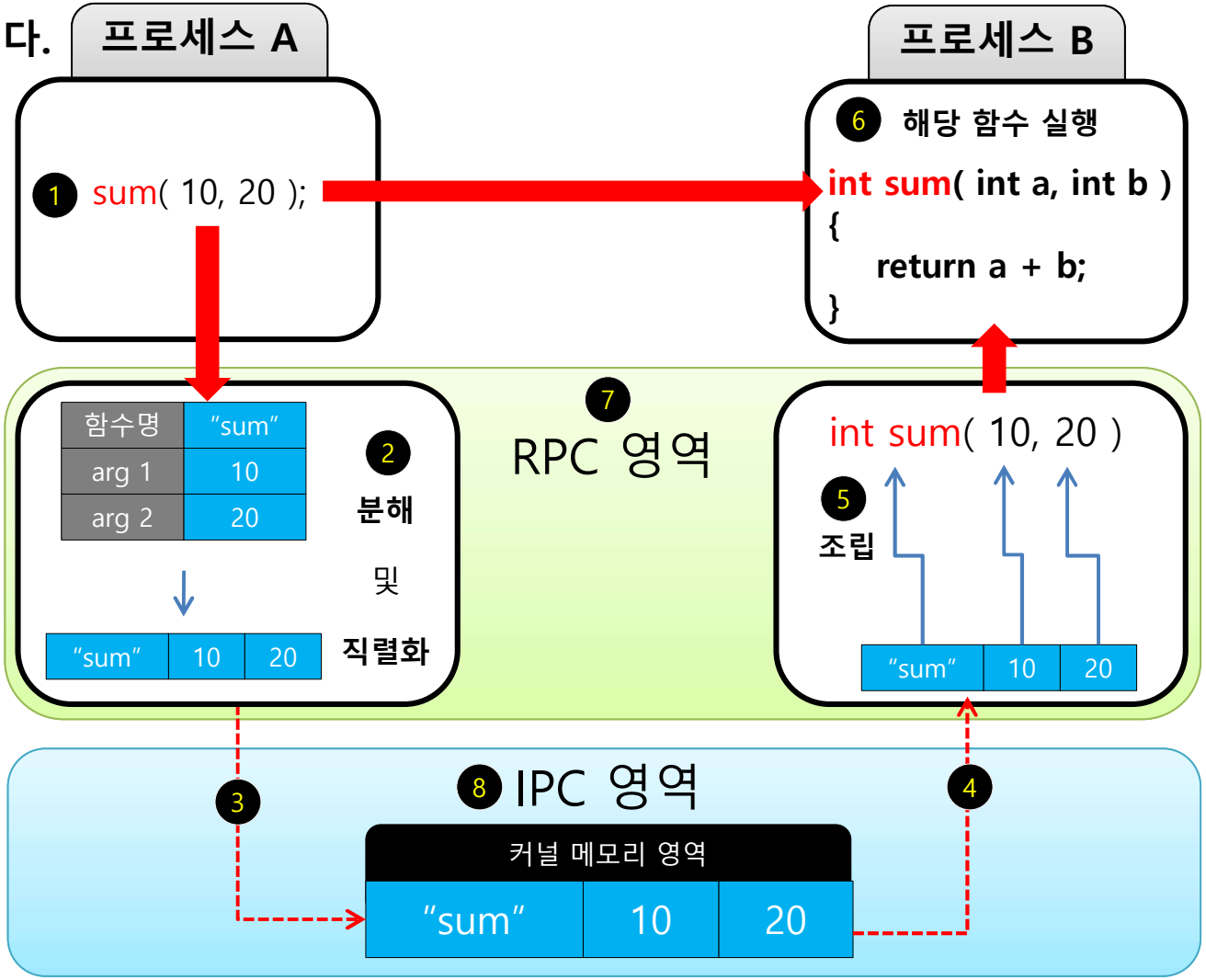
- AIDL을 통해 자동으로 생성된 바인더 객체에 대해서 좀더 살펴보자.



- 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

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

```
ICountService
  Stub
    Proxy
      Proxy(IBinder)
      asBinder(): IBinder
      getInterfaceDescriptor(): String
      getCurCountNumber(): int
      mRemote: IBinder
    Stub()
    asInterface(IBinder): ICountService
    asBinder(): IBinder
    onTransact(int, Parcel, Parcel, int): boolean ↑ Binder
    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

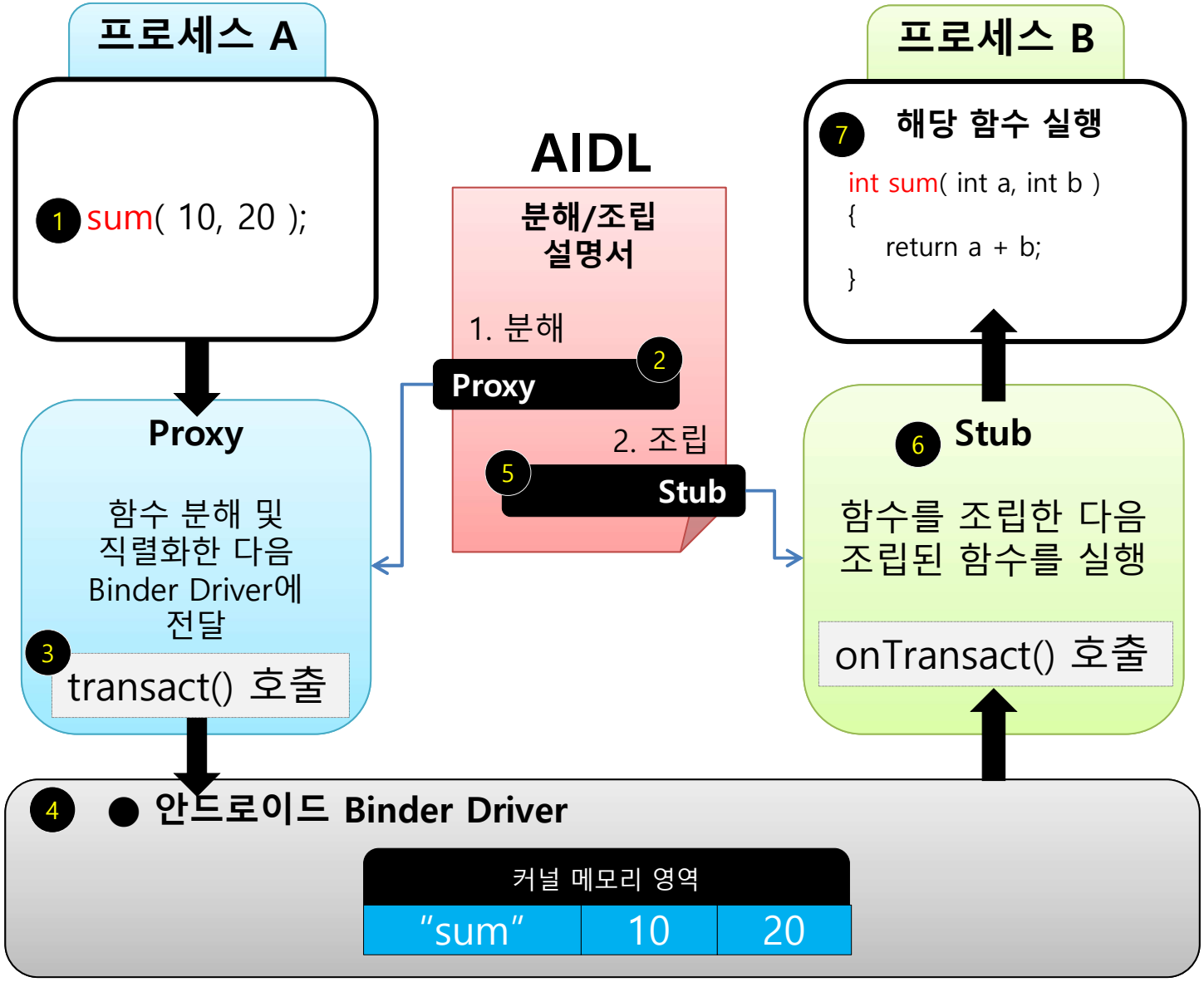
ICountService.java

```
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


```
        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

- 바운드 서비스의 메소드는 어느 스레드에서 동작할까?

- 리모트 클라이언트에서 호출하는 경우에는 별도의 작업 스레드에서 동작한다.

Threads | File Ex... | Heap | Alloca... | Netwo...

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

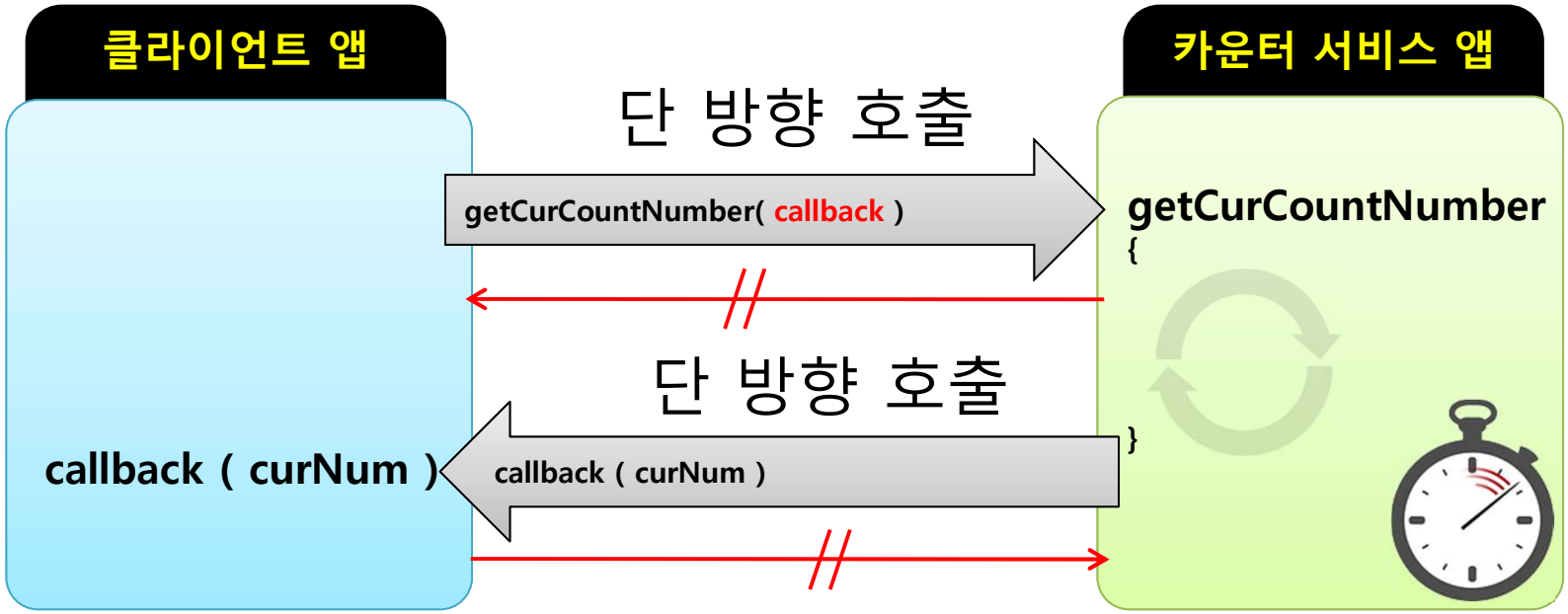
- 서비스의 공개된 인터페이스의 메소드가 오래 걸리는 작업을 수행한다면 그 만큼 해당 인터페이스를 사용한 클라이언트는 대기해야 한다.

```
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

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



바운드 서비스 – 바운드 서비스의 콜백 인터페이스

- 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.

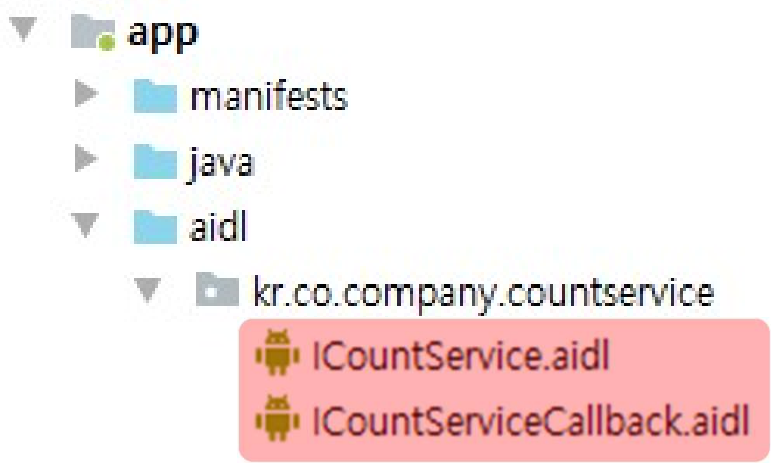
- 기존의 ICountService.aidl을 수정한다.

동일한 패키지에 있는 클래스도 import해야 한다.

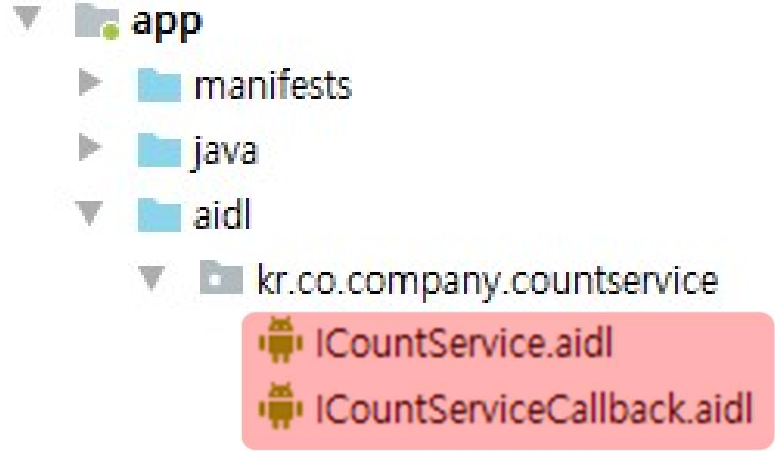
```
// ICountService.aidl
package kr.co.company.countservice;

import kr.co.company.countservice.ICountServiceCallback;

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

바운드 서비스 – 바운드 서비스의 콜백 인터페이스

● **카운트 서비스 패키지**



● **카운트 서비스 클라이언트 패키지**



바운드 서비스 – 바운드 서비스의 콜백 인터페이스

- 카운트 서비스를 수정한다.

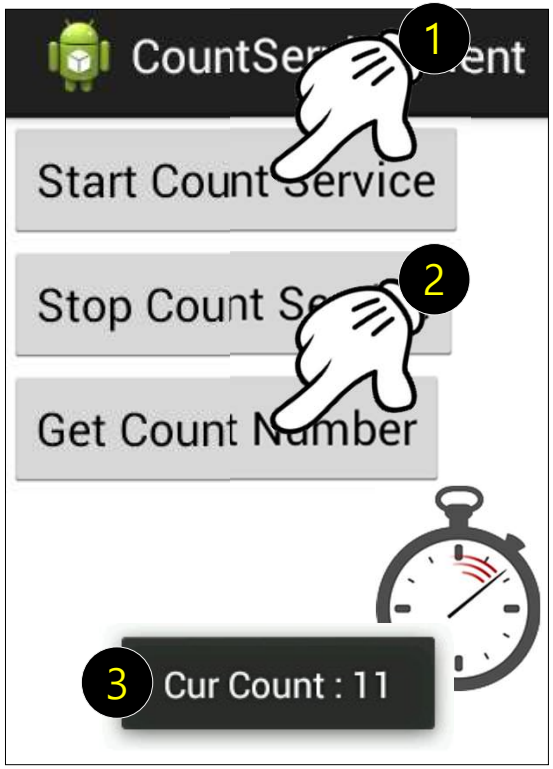
```
public class CountService extends Service {

    ...

    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

- **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



● 클라이언트 앱의 로그

PID	TID	Text
1124	1136	Cur Count Number : 11

4

● 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

5

바운드 서비스 – 바운드 서비스의 생존

- 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.

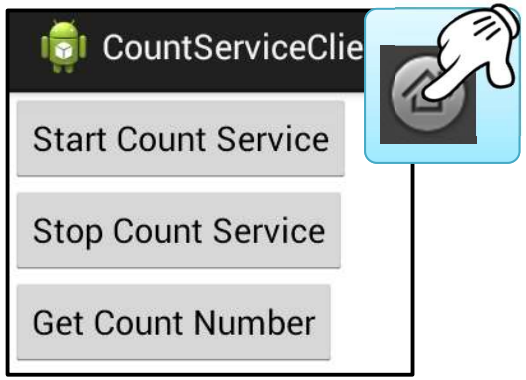
```
public class CountService extends Service {
    ...
    @Override
    public void onCreate() {
        super.onCreate();
        Log.i("superdroid", "onCreate()");

        ...

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

바운드 서비스 – 바운드 서비스의 생존

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



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

- 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);
    }
}
```

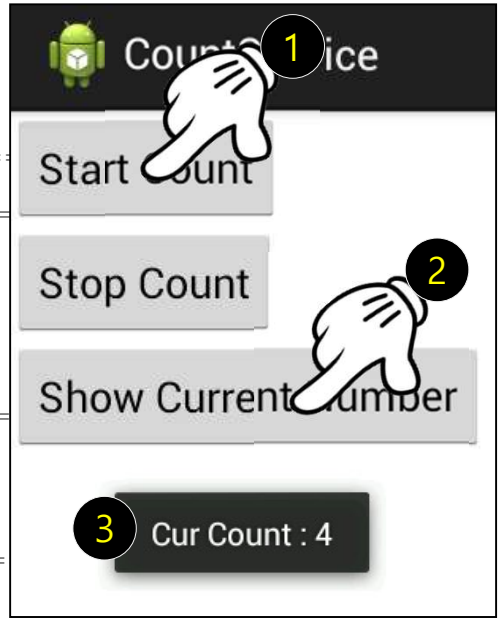
내부 바운드 서비스

- 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;
      }
      // ==================================================================
   }
}
}
```



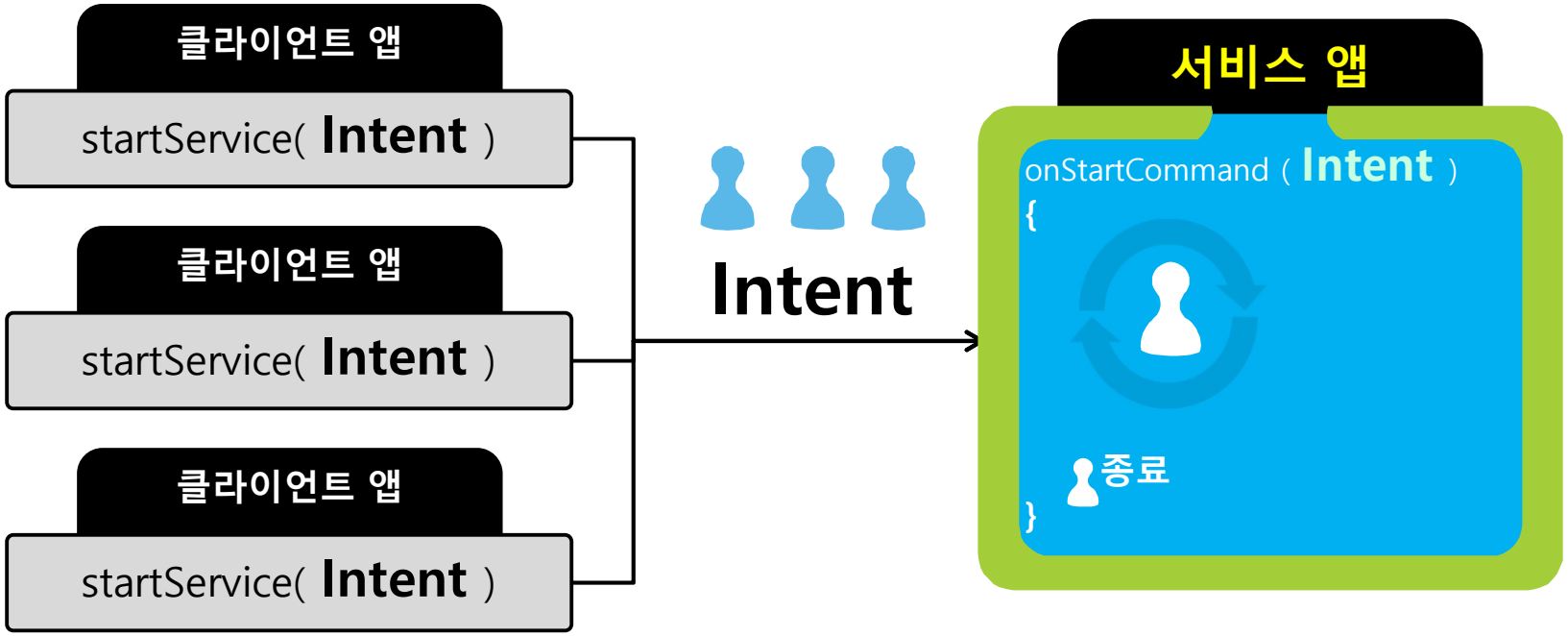
인텐트 서비스와 메신저를 이용한 서비스

- 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

- 인텐트 서비스IntentServie는 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

- **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>
```

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

- 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

- 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;
            }
        }
    }
}
```

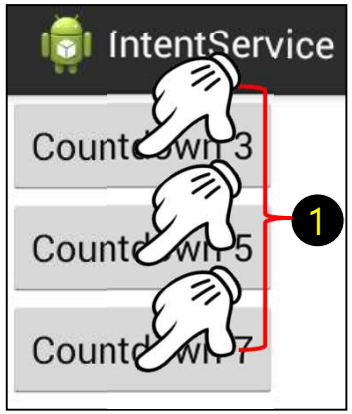
인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 인텐트 서비스

● 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()
```

2 3 4

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



● 로그

```
onCreate()                                  2
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                    3
start Id 1 Countdown : 2
start Id 2 Countdown : 1
stopSelf : 3                                4
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

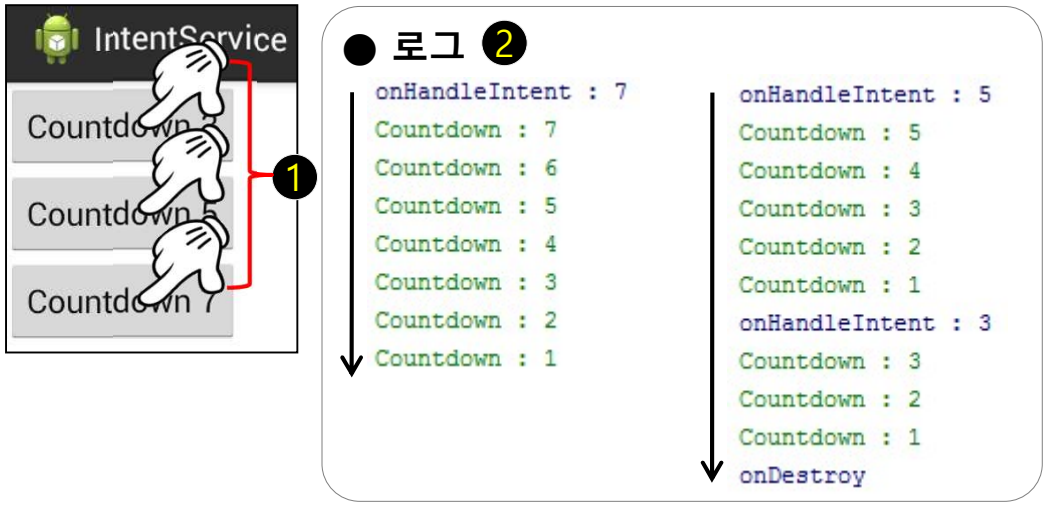
인텐트 서비스와 메신저를 이용한 서비스 - 스타티드 서비스의 파생 클래스 인텐트 서비스

- **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

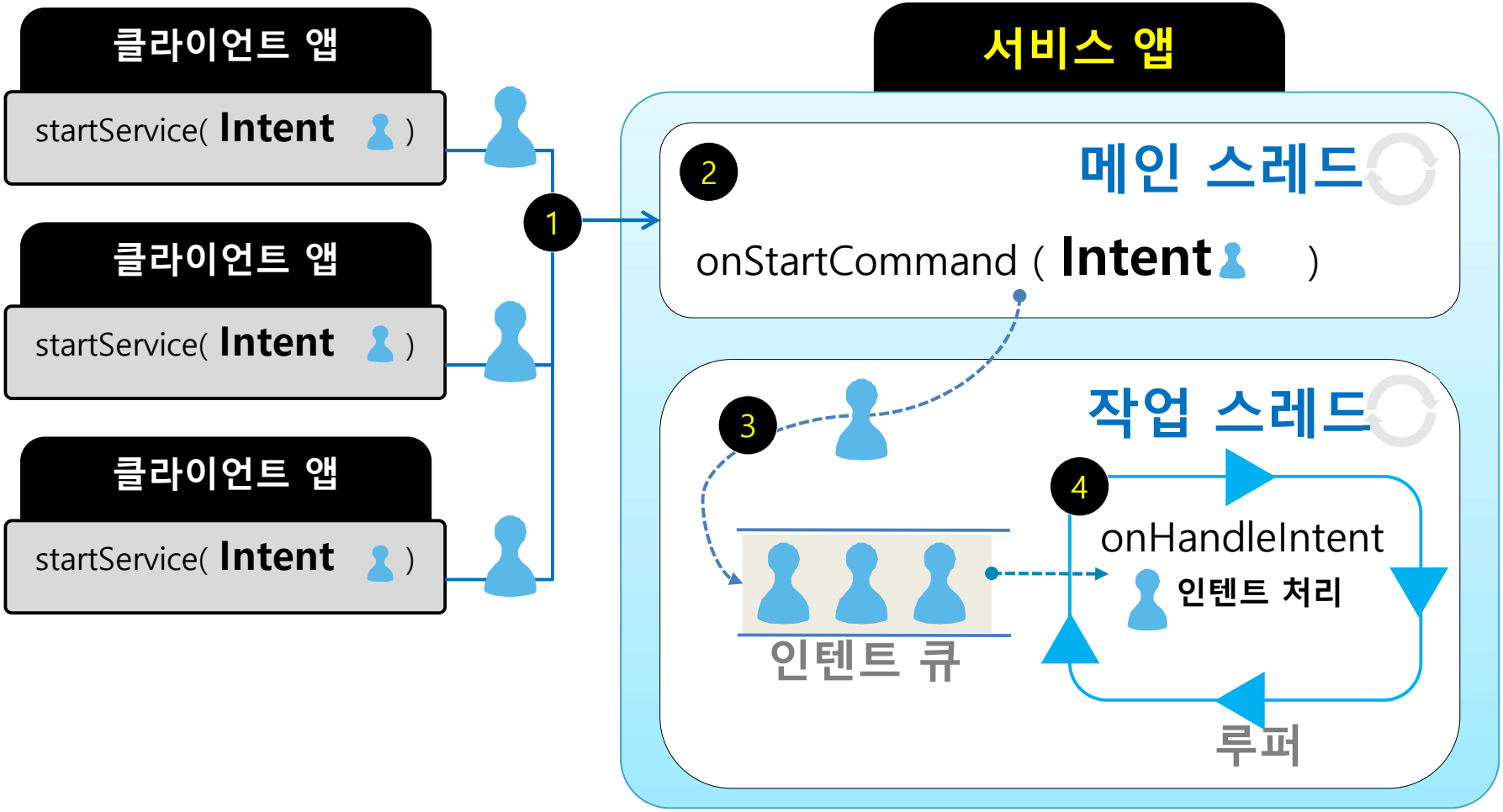


■ **너무 간단하지 않은가!**

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

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

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



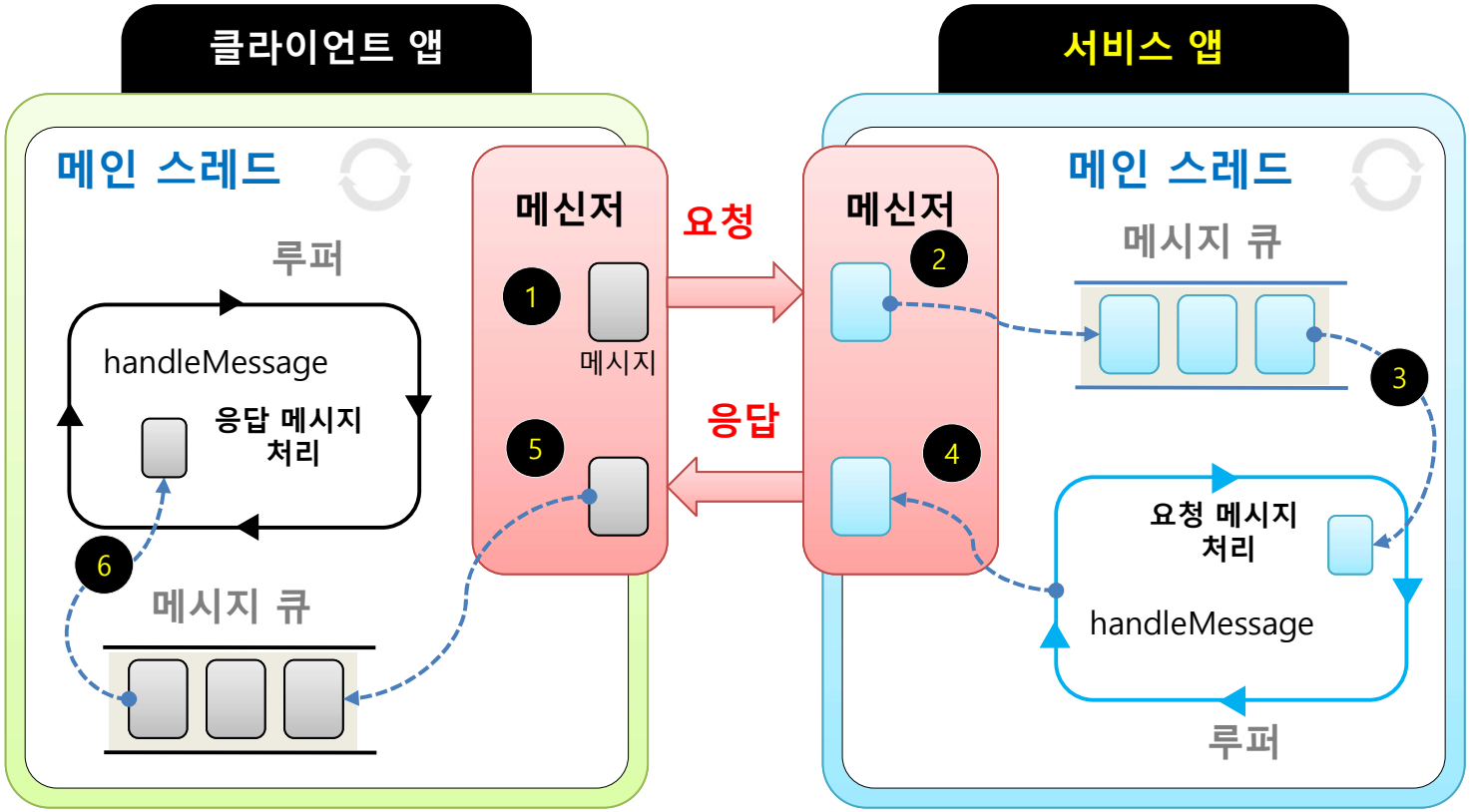
■ 인텐트 서비스에서도 내부적으로 HandlerThread를 사용한다.

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

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

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

- 메신저Messenger의 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

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

</manifest>
```

인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저


```
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 차을 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


```
    @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

- **정말 메신저는 AIDL 파일 없이 프로세스 통신을 하는 것일까?**
  - 사실 안드로이드 프레임워크 내부에는 AIDL 파일이 존재한다.

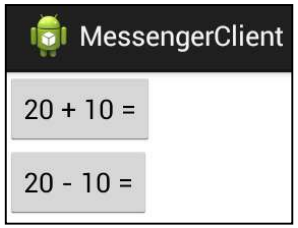
```
package android.os;
import android.os.Message;

/** @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

- **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>
```



인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저


```
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


```
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();
}
```

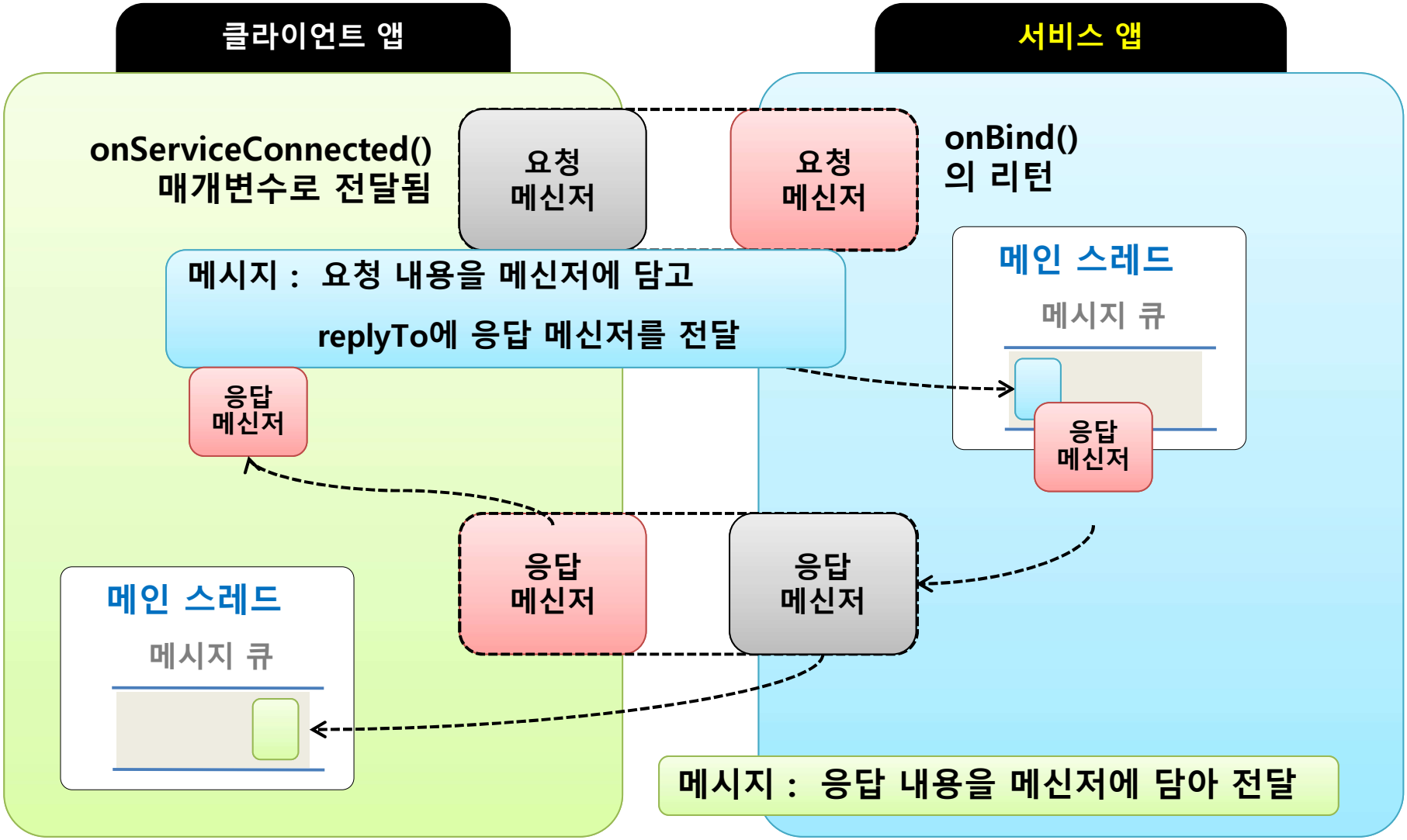
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저


```
// 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;
        }
    }
}
```

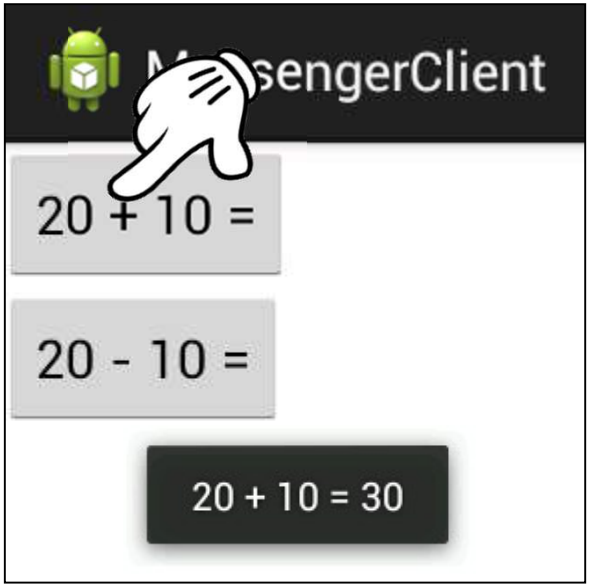
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

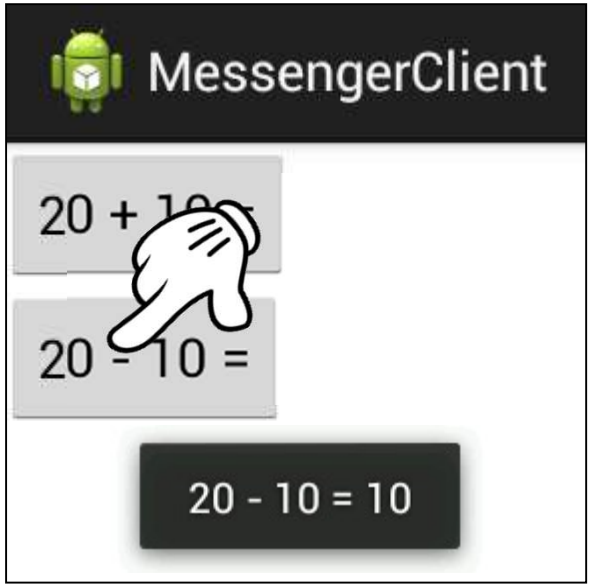

```
private Handler mMessengerResponseHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            // 두 수의 합의 결과 처리
                // ==========================================================
            case RESPONSE_MSG_PLUS_VALUE:
            {
                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



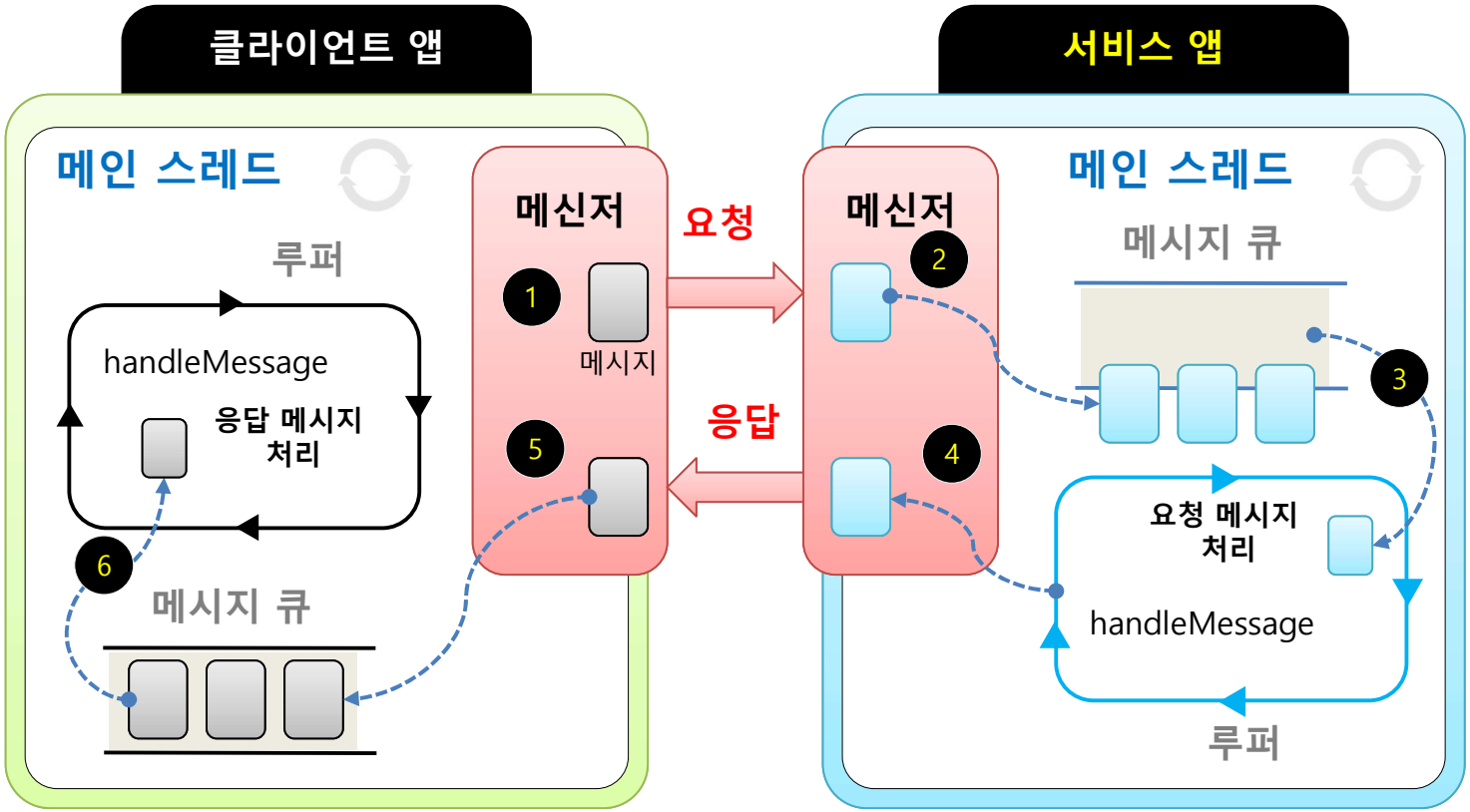
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저





인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

- **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

- 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.

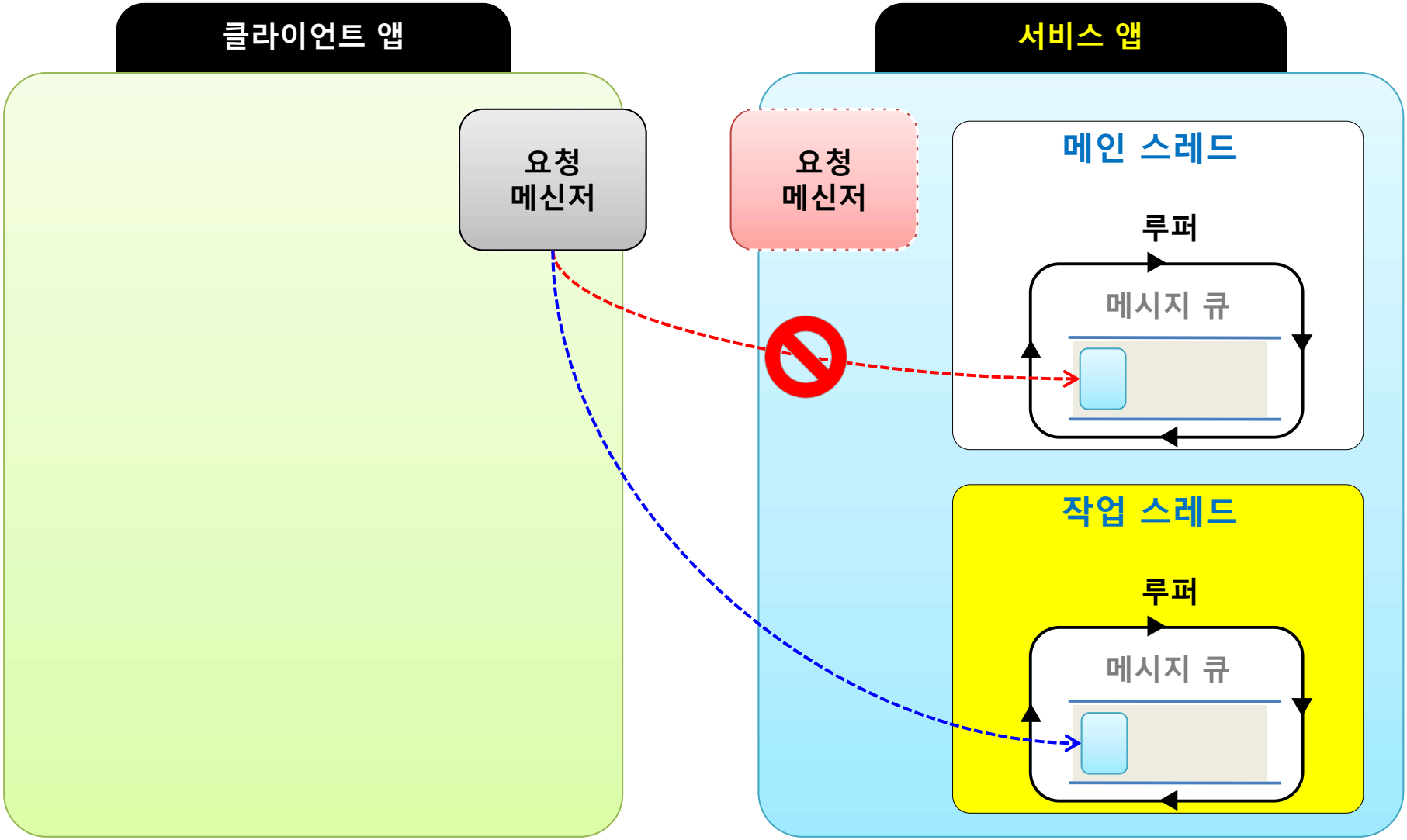
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

- 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.

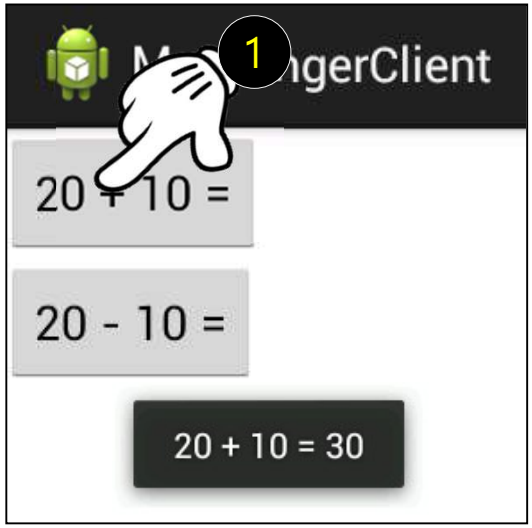
인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저

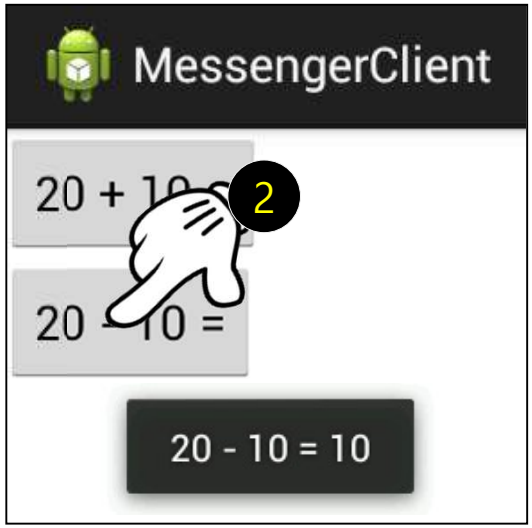

```
public class MessengerService extends Service {
    ...

    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



인텐트 서비스와 메신저를 이용한 서비스 - 바운드 서비스를 이용하는 메신저





● **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	VmWait			
*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

3

● **로그**

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

단말기에서 실행 중인 서비스 정보 보기

- 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{...
```