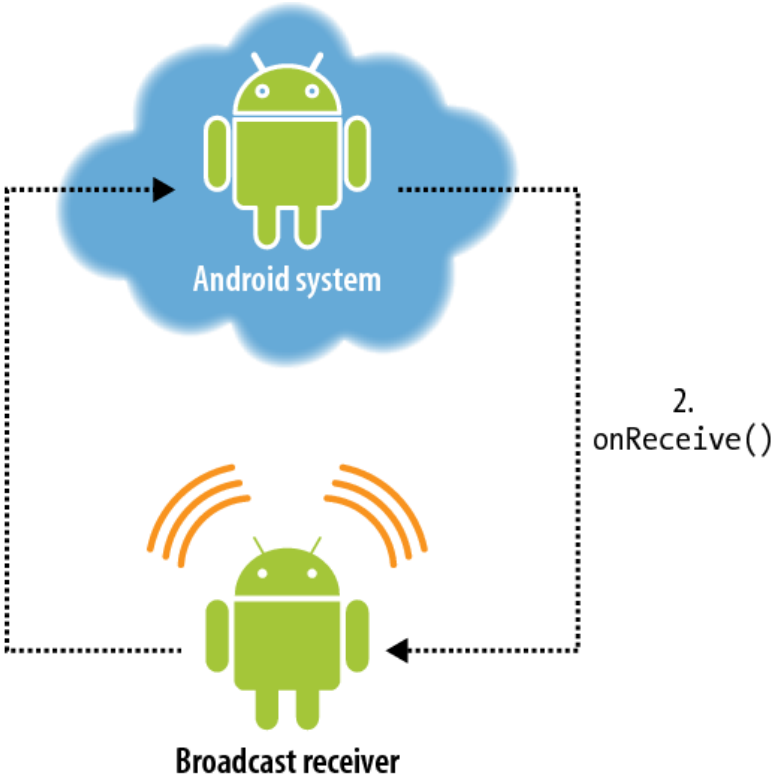
BroadcastReceiver

Programming the Android Platform

CS 282
Principles of Operating Systems II
Systems Programming for Android

BroadcastReceiver Overview

- Components that listen for broadcast events & receive/react to the events
  - Events implemented as Intent instances
  - Events are broadcast system-wide
  - Interested BroadcastReceivers receive Intent via onReceive()
- BroadcastReceivers have no user interface (& other limitations)
- Android's Intents framework supports a wide range of notification models
- BroadcastReceivers can be used for both user-defined & system events



http://developer.android.com/reference/android/content/BroadcastReceiver.html

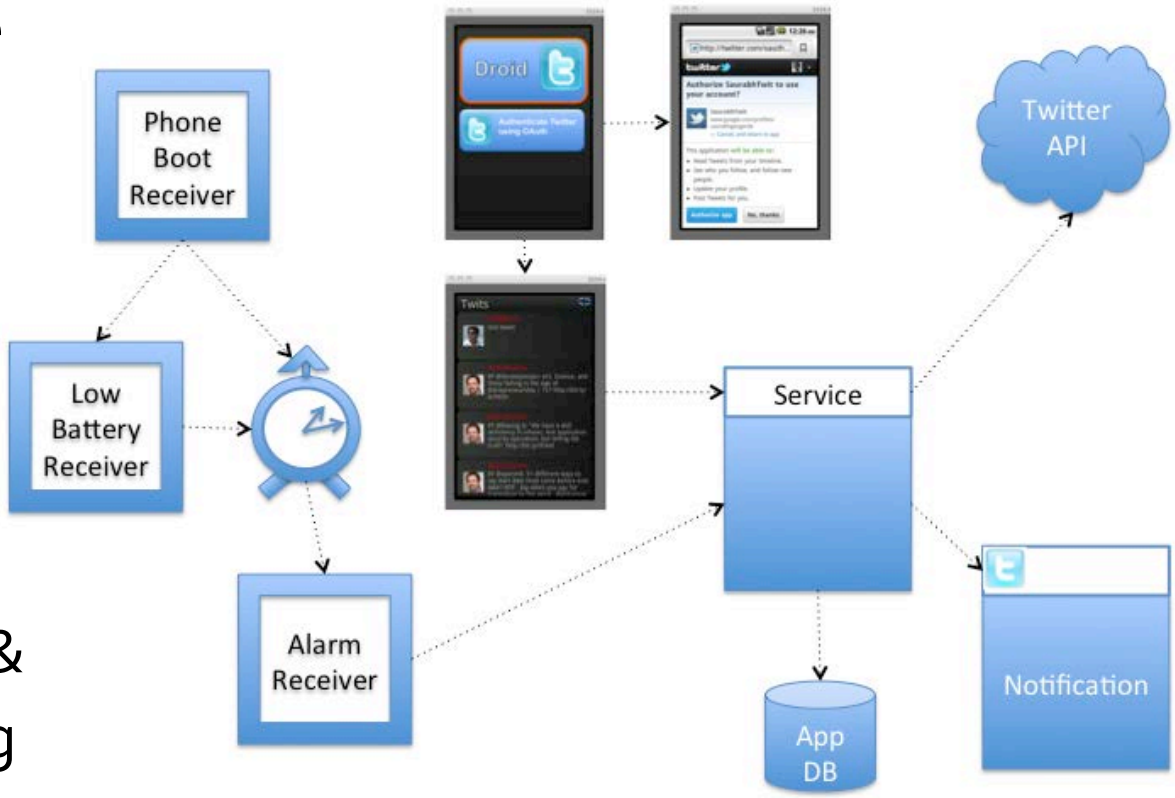
System Events

- Several system events defined as final static fields in the Intent class
  - Other Android system classes also define events, e.g. the TelephonyManager defines events for the change of the phone state
- The following table lists a few important system events

Event	Description
Intent.ACTION_BOOT_COMPLETED	Boot completed. Requires the android.permission.RECEIVE_BOOT_COMPLETED permission
Intent.ACTION_POWER_CONNECTED	Power got connected to the device
Intent.ACTION_POWER_DISCONNECTED	Power got disconnected to the device
Intent.ACTION_BATTERY_LOW	Battery gets low, typically used to reduce activities in your app which consume power
Intent.ACTION_BATTERY_OKAY	Battery status good again

Typical BroadcastReceiver Use Case

- BroadcastReceivers are registered to receive specific Intents
- Some component broadcasts an Intent
- Activity Manager Service identifies appropriate recipients & delivers event by calling onReceive() on BroadcastReceiver
- Event handled in onReceive()



Registering BroadcastReceivers

- BroadcastReceivers can be register in two ways
  - *Statically* via AndroidManifest.XML
    - Include <receiver> in AndroidManifest.xml

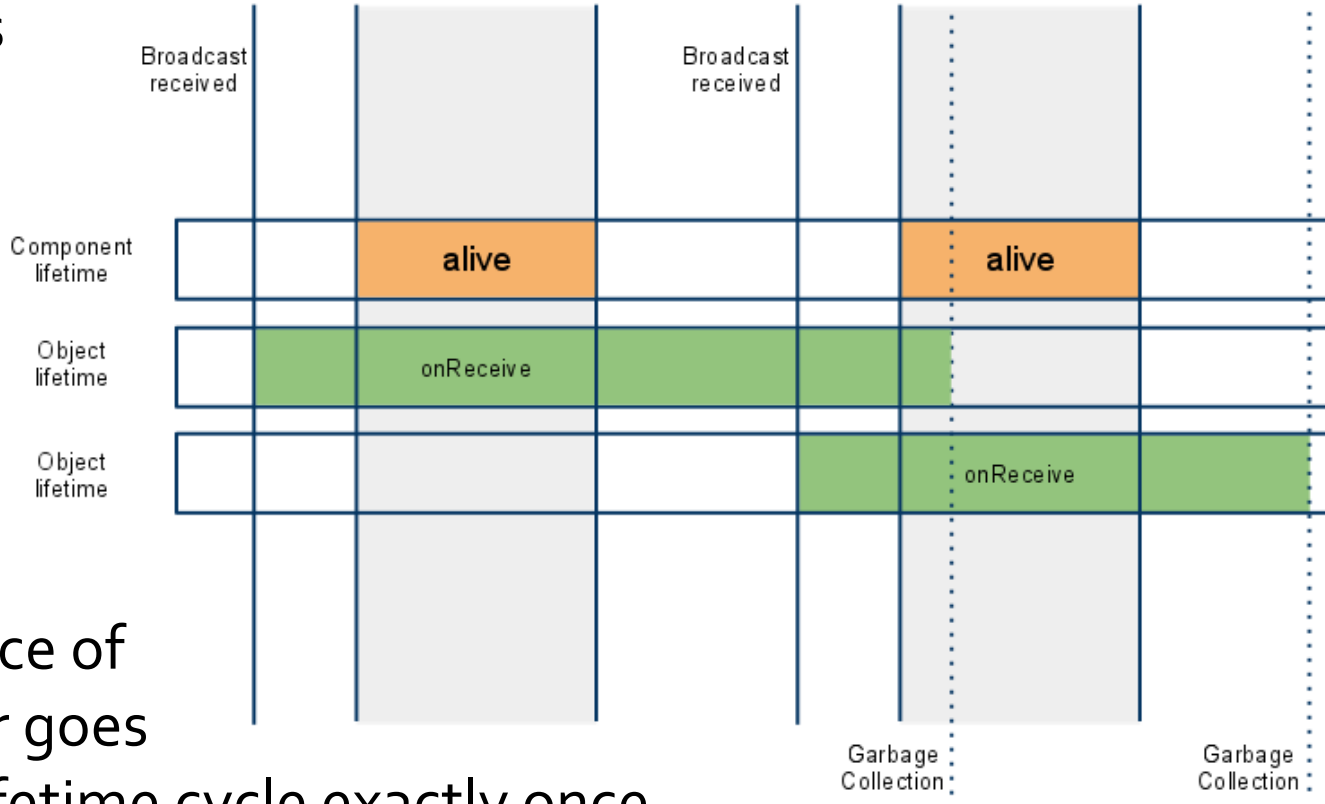
```
<application>
  <receiver receiver_specs >
    <intent-filter> event_specs
    </intent-filter>
  </receiver>
</application>
```

    - Receiver registered at boot time or when application package is added at runtime
  - *Dynamically* via Context. registerReceiver()
    - Create an IntentFilter
    - Create a BroadcastReceiver
    - Register BroadcastReceiver to receive Intents that match the IntentFilter using Context. registerReceiver()
    - Call Context.unRegisterReceiver() to unregister BroadcastReceiver

http://www.vogella.com/articles/AndroidBroadcastReceiver/article.html

Static BroadcastReceivers

- Static BroadcastReceiver instantiated when broadcast is received
  - The object is abandoned when onReceive() returns
- If new broadcast is received, new object is created & onReceive() is called on that new instance
  - After that object is also abandoned
- Every static instance of BroadcastReceiver goes thru component lifetime cycle exactly once



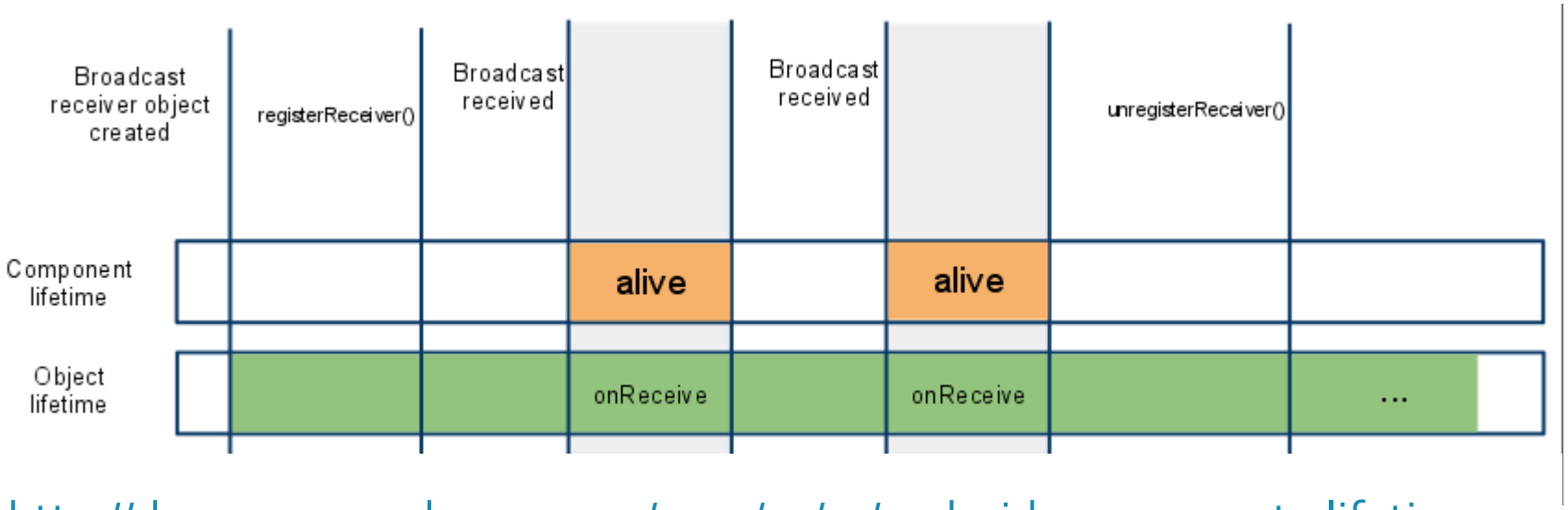
http://devmaze.wordpress.com/2011/07/17/android-components-lifetime

Static BroadcastReceiver Example

```
<application ...>
    <activity android:name=".SimpleBroadcast" ...>  ...  </activity>
     <receiver android:name=".Receiver2">
        <intent-filter android:priority="5">
          <action android:name=
              "course.examples.BroadcastReceiver.intent.
               action.TEST2">
         </action>
        </intent-filter>
     </receiver>
 </application>
 <uses-permission
          android:name="android.permission.VIBRATE">
 </uses-permission>
```

Dynamic BroadcastReceiver

- Android system doesn't control dynamic BroadcastReceiver objects
- Dynamic receivers can be instantiated by application at any time before calling registerReceiver() & they are not destroyed after onReceive() returns
- Dynamic BroadcastReceiver objects may go through several component lifetime cycles



http://devmaze.wordpress.com/2011/07/17/android-components-lifetime

Dynamic BroadcastReceiver Example

```
public class SingleBroadcast extends Activity {
   public static final String CUSTOM_INTENT =
    "course.examples.BroadcastReceiver.intent.action.TEST1";

   public void onCreate(Bundle savedInstanceState) {
      …
      registerReceiver(new Receiver1(),
                              new IntentFilter(CUSTOM_INTENT));
    }
}
```

Android Event Broadcast Models

- Normal vs. Ordered
  - A *normal* broadcast Intent is sent *asynchronously* & ordering of delivery to set of BroadcastReceivers eligible to receive it is undefined
  - An *ordered* broadcast Intent is delivered sequentially to each member of the set of BroadcastReceivers eligible to receive it in the order defined by *priority* of associated IntentFilters
- Sticky vs. Non-Sticky
  - A broadcast Intent specified as *sticky* will be retained by system after it has been sent
  - A non-sticky intent will be discarded after its initial broadcast
- With or without permissions
  - An app can specify a permission when sending a normal or ordered broadcast Intent

http://www.vogella.com/articles/AndroidBroadcastReceiver/article.html

Normal Broadcasts

```
//public abstract class Context ...

// send Intent to interested BroadcastReceivers
void sendBroadcast (Intent intent)

// send Intent to interested BroadcastReceivers
// if they have the specified permissions
void sendBroadcast (Intent intent, String receiverPermission)
```

http://developer.android.com/reference/android/content/Context.html

Normal Broadcasts (cont.)

```
public class SimpleBroadcast extends Activity {
  public static final String CUSTOM_INTENT =
    "course.examples.BroadcastReceiver.intent.action.TEST2";
  public void onCreate(Bundle savedInstanceState) {
      …
      Button button = (Button) findViewById(R.id.button);
      button.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
             sendBroadcast(new Intent(CUSTOM_INTENT),
                                 android.Manifest.permission.VIBRATE);
         }
      });
…
```

Ordered Broadcasts

- An ordered broadcast Intent can have additional data associated with it: a code (an int), data (a String), & extras (a Bundle)
- The initial values of the additional data can be specified by the sender of the ordered broadcast Intent

```
//public abstract class Context …

// send Intent to interested BroadcastReceivers in priority order
void sendOrderedBroadcast (Intent intent,  String receiverPermission)

// send Intent to interested BroadcastReceivers in priority order
// sender can provide various parameters for greater control
void sendOrderedBroadcast (Intent intent, String receiverPermission,
                           BroadcastReceiver resultReceiver,
                           Handler scheduler, int initialCode,
                           String initialData, Bundle initialExtras)
```

http://developer.android.com/reference/android/content/Context.html

Ordered Broadcasts (cont.)

```
public class CompoundOrderedBroadcast extends Activity {
  …
  public static final String CUSTOM_INTENT =
        "course.examples.BroadcastReceiver.intent.action.TEST4";
  public void onCreate(Bundle savedInstanceState) {
    …
    Button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        sendOrderedBroadcast(new Intent(CUSTOM_INTENT),
                             android.Manifest.permission.VIBRATE);
      }
  });
  …
```

Ordered Broadcasts (cont.)

```
public class CompOrdBcastWithResultReceiver extends Activity {
  public void onCreate(Bundle savedInstanceState) {
  …
    button.setOnClickListener(new OnClickListener() {
      public void onClick(View v) {
        sendOrderedBroadcast(new Intent(CUSTOM_INTENT), null,
            new BroadcastReceiver() {
              public void onReceive(Context context, Intent intent) {
                System.out.println("Final Result is:" + getResultData());
              }
            },  null, 0, null, null);
      }
    });
…
```

Sticky Broadcasts

- A normal broadcast Intent isn't available after being sent/processed
- sendStickyBroadcast(Intent) makes the Intent sticky, meaning the Intent stays around after broadcast is complete
- When BroadcastReceivers are dynamically registered
  - Cached sticky Intents matching the specified IntentFilter are broadcast to the BroadcastReceiver
  - One matching sticky Intent is returned to the caller
- Sticky broadcast Intent can be retrieved at any time after being sent without registering a BroadcastReceiver
  - A sticky broadcast Intent can be removed after it has been sent
- The Android system uses sticky broadcast for certain system information
  - e.g., the battery status is send as sticky Intent & can get received at any time

http://www.vogella.com/articles/AndroidBroadcastReceiver/article.html#broadcastreceiver_sticky

Sticky Broadcasts (cont.)

```
//public abstract class Context …

// send sticky Intent to interested BroadcastReceivers
void sendStickyBroadcast (Intent intent)

// send sticky Intent to interested BroadcastReceivers in priority order
// sender can provide various parameters for greater control
void sendStickyOrderedBroadcast (Intent intent,
                                 BroadcastReceiver resultReceiver,
                                 Handler scheduler,
                                 int initialCode,
                                 String initialData,
                                 Bundle initialExtras)
```

- Broadcaster must have BROADCAST_STICKY permission to send sticky Intents

BroadcastReceiver Permissions

- An app can specify a permission when sending a normal or ordered broadcast Intent
- BroadcastReceiver can't receive a normal or ordered broadcast Intent sent with an associated permission if the app that registered the BroadcastReceiver hasn't been granted that permission
- An app can specify a permission when registering BroadcastReceiver
- BroadcastReceiver registered with an associated permission can't receive any normal or ordered broadcast Intent sent by an app that has not been granted that permission
- As of Android 3.1 BroadcastReceivers won't receive *Intents* if corresponding app has never been started by user or if user explicitly stopped the application via the Android menu in *Manage Application*

http://developer.android.com/reference/android/content/BroadcastReceiver.html#Security

Intent Resolution

- Intents are divided into 2 groups:
  - *Explicit intents* designate the target component by its name (the component name field has a value set)
    - Since component names are generally not known to developers of other apps, explicit intents are typically used for app-internal messages, e.g., an activity starting a subordinate service or launching a sister activity
  - *Implicit intents* do not name a target (field for component name is blank)
    - Implicit intents are often used to activate components in other applications
- Some debugging tips
  - Log BroadcastReceivers that match an Intent
    - Intent.setFlag(FLAG_DEBUG_LOG_RESOLUTION)
  - List BroadcastReceivers registered to receive intents
    - Dynamic registration
      - % adb shell dumpsys activity b
    - Static registration
      - % adb shell dumpsys package

http://developer.android.com/guide/components/intents-filters.html#ires

Intent Resolution (cont'd)

- BroadcastReceivers can have one or more intent filters to indicate which implicit intents they can handle
  - Each filter describes a set of intents the component is willing to receive
- Implicit intent is delivered to a component only if it can pass thru one of the component's filters
  - Explicit intent is always delivered to its target & filter is not consulted
- A filter has fields that parallel the action, data, & category fields of an Intent object
  - An implicit intent is tested against the filter in all three areas
    - To be delivered to component that owns filter, it must pass all three tests
  - If it fails even one of them, the Android system won't deliver it to the component
    - Since a component can have multiple intent filters, an intent that doesn't pass through one of a component's filters might make it through another

http://developer.android.com/guide/components/intents-filters.html#ires

Intent Resolution (cont'd)

- Only three aspects of an Intent object are consulted when the object is tested against an intent filter: action, data (both URI & data type), category
- The extras & flags play no part in resolving which component receives an intent
- Action test example: an <intent-filter> element in the manifest file lists actions as

```
<intent-filter . . . >
        <action android:name="com.example.project.SHOW_CURRENT" />
        <action android:name="com.example.project.SHOW_RECENT" />
        <action android:name="com.example.project.SHOW_PENDING" /> . . .
</intent-filter>
```

http://developer.android.com/guide/components/intents-filters.html#ires

Event Handling in onReceive()

- Events delivered by calling onReceive() & passing Intent as a parameter
- onReceive() should be short-lived
  - Hosting process has high priority while onReceive() runs & often terminats when onReceive() returns
- BroadcastReceivers should beware of asynchronous operations
  - e.g., showing a dialog, binding to a Service, starting an Activity via startActivityForResult()

```
public class MyReceiver extends
                              BroadcastReceiver
{
  public void onReceive(Context context,
                                 Intent intent) {
    Intent service =
        new Intent(context,
                        MyService.class);

    // We're starting an unbound service
    context.startService(service);
  }
}
```

- If you have potentially long running operations you should trigger a Service for that

Handling a Normal Broadcast

```
public class Receiver1 extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    System.out.println(this + ":GOT THE INTENT");
    // emulator doesn't support vibration
    Vibrator v = (Vibrator) context.getSystemService(
                                        Context.VIBRATOR_SERVICE);
   v.vibrate(500);
  }
}
```

Handling an Ordered Broadcast

- **Passing results**

```
public class Receiver1 extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    String tmp = getResultData() != null ? getResultData() : "";
    setResultData(tmp + ":Receiver 1:");
  }
}
```

Handling an Ordered Broadcast

- **Aborting a broadcast**

```
public class Receiver2 extends BroadcastReceiver {
  public void onReceive(Context context, Intent intent) {
    if (isOrderedBroadcast()) {
      abortBroadcast();
   }
   System.out.println(this + ":GOT THE INTENT");
   // emulator doesn't support vibration
   Vibrator v = (Vibrator) context.getSystemService(
                                        Context.VIBRATOR_SERVICE);
   v.vibrate(500);
  }
}
```

Handling a Sticky Broadcast

```
public class StickyIntentBroadcastReceiverActivity extends Activity {
  public void onCreate(Bundle savedInstanceState) {
    registerReceiver(new BroadcastReceiver() {
      public void onReceive(Context context, Intent intent) {
        if (intent.getAction().equals(
                              Intent.ACTION_BATTERY_CHANGED)) {
          String age = "Reading taken recently";
          if (isInitialStickyBroadcast()) { age = "Reading may be stale”; }
            state.setText("Current Battery Level" +  String.valueOf(
              intent. getIntExtra(BatteryManager.EXTRA_LEVEL, -1))   + "\n" + age);
        }
      }
    }, new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
  }
}
```

Source Code Examples

- BroadcastReceiverCompoundBroadcast
- BroadcastReceiverCompoundOrderedBroadcast
- BroadcastReceiverCompoundOrderedBroadcast WithResultReceiver
- BroadcastReceiverSingleBroadcast DynamicRegistration
- BroadcastReceiverSingleBroadcastStaticRegistration
- BroadcastReceiverStickyIntent