Programação para Dispositivos Móveis

Leopoldo Teixeira
lmt@cin.ufpe.br | @leopoldomt

Até agora temos usado **`Intent`** para...

Como enviar um *broadcast*?

Crie um **`Intent`** e chame **`sendBroadcast()`**

O que acontece?

BroadcastReceiver

BroadcastReceiver

- Classe base para componentes que:
  - Esperam pela ocorrência de certos eventos
  - Recebem estes eventos
  - Reagem a estes eventos

`BroadcastReceiver`

- Broadcasts podem ser recebidos (registrados)
  - estaticamente (via manifest)
  - dinamicamente (direto em uma **`Activity`**)

Como isso funciona?

- Receivers se registram para receber eventos específicos de interesse
- Por exemplo:
  - ACTION_BOOT_COMPLETED
  - ACTION_BATTERY_LOW

Como eventos são representados?

- Um componente pode fazer algum tipo de processamento em que é interessante informar algum(ns) broadcast receiver(s)
- O evento é representado como um Intent
- Este Intent é *broadcasted* para o sistema

Finalmente…

- Android roteia os Intents para os BroadcastReceivers que se registraram para recebê-los

- Os BroadcastReceivers recebem o Intent por meio do método **`onReceive()`**

Fluxo Geral

- Registrar BroadcastReceivers
- Fazer o broadcast de um Intent
- Android entrega Intent aos recipientes registrados, chamando o método **`onReceive()`**
- Evento é tratado dentro do método **`onReceive()`**

Implementando um BroadcastReceiver

- Criar subclasse de **`BroadcastReceiver`**
  - classe abstrata
- Implementar o método **`onReceive()`**
  - **`context`** - objeto que utilizamos para acessar informação extra, iniciar services ou activities
  - **`intent`** - o objeto com a ação que foi usada para registrar o receiver. O intent pode carregar informação adicional que podemos utilizar na implementação

Eventos do sistema

Evento	Uso
Intent.ACTION_BATTERY_LOW	Nível caiu abaixo de um certo limite
Intent.ACTION_BATTERY_OKAY	Nível subiu acima do limite
Intent.ACTION_BOOT_COMPLETED	Android está rodando!
Intent.ACTION_DEVICE_STORAGE_LOW	Espaço de armazenamento limitado
Intent.ACTION_DEVICE_STORAGE_OK	Espaço de armazenamento ok
Intent.ACTION_HEADSET_PLUG	Headset foi plugado ou desplugado
Intent.ACTION_LOCALE_CHANGED	Usuário mudou língua do dispositivo
Intent.ACTION_MY_PACKAGE_REPLACED	App foi atualizado
Intent.ACTION_PACKAGE_ADDED	Novo app foi instalado
Intent.ACTION_POWER_CONNECTED	Dispositivo conectado à fonte de energia
Intent.ACTION_POWER_DISCONNECTED	Dispositivo desconectado da fonte de energia
AudioManager.ACTION_AUDIO_BECOMING_NOISY	O alto-falante está prestes a ser utilizado

Receiver simples

*Receiver.java

```java
package course.examples.BroadcastReceiver.singleBroadcastStaticRegistration;
import android.content.BroadcastReceiver;..

public class Receiver extends BroadcastReceiver {
    private final String TAG = "Receiver";
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.i(TAG, "INTENT RECEIVED");
        Vibrator v = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);
        v.vibrate(500);
        Toast.makeText(context, "INTENT RECEBIDO pela classe Receiver", Toast.LENGTH_LONG).show();
    }
}
```

Registrar BroadcastReceiver

- Duas maneiras
  - Estaticamente, via **`AndroidManifest.xml`**
  - Dinamicamente, via chamada ao método **`registerReceiver()`**

Registro Dinâmico

- Criar um **`IntentFilter`**
- Criar um **`BroadcastReceiver`**
- Registrar usando **`registerReceiver()`**
  - **`LocalBroadcastManager`** - intra-app
  - **`Context`** - inter-app
- Chamar **`unRegisterReceiver()`** para cancelar o registro do **`BroadcastReceiver`**

Só recebe os broadcasts enquanto a **`Activity`** está ativa…

`registerReceiver()`

- Chamamos **`registerReceiver()`** no objeto de **`Context`** passando os seguintes parâmetros:
  - **`receiver`** - o **`BroadcastReceiver`** que desejamos registrar
  - **`intent`** - o objeto **`IntentFilter`** que especifica qual evento o *receiver* estará escutando

Registro Dinâmico

- Ao registrar *receivers* dinamicamente, estes vivem enquanto o componente viver, e Android envia eventos até que este componente seja destruído.

- Devemos gerenciar o ciclo de vida corretamente. Devemos lembrar de ‘de-registrar’ o mesmo receiver quando o método **`onPause()`** for chamado

- Uma opção é registrar no método **`onResume()`** e retirar o registro no método **`onPause()`**

Registro Estático

- Incluir tags **`<receiver>`** e **`<intent-filter>`** no arquivo AndroidManifest.xml
  - **`<intent-filter>`** vai dentro da tag **`<receiver>`**
- A informação presente em **`<intent-filter>`** vai determinar quais os eventos que aquele receiver está interessado
- As regras de **`<intent-filter>`** são as mesmas já vistas na aula de Intents

Formato de **<receiver>**

```
<receiver
  android:enabled=["true" | "false"]
  android:exported=["true" | “false"]
  android:icon="drawable resource”
  android:label="string resource”
  android:name=“string"
  android:permission=“string"
  android:process=“string">
  <intent-filter>…</intent-filter>
...
</receiver>
```

https://developer.android.com/guide/topics/manifest/receiver-element.html

Registro Estático

- O registro acontece quando:
  - sistema termina de dar boot
  - pacote da aplicação é adicionado em tempo de execução

Exemplo

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

<application
    android:icon="@drawable/icon"
    android:label="@string/app_name" >
    <activity
        android:name=".ServiceConsumerActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

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

    <receiver android:name="MyScheduleReceiver" >
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>
    <receiver android:name="MyStartServiceReceiver" >
    </receiver>
</application>
```

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

Exemplo

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

<application
```

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class MyReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    // assumes WordService is a registered service
    Intent intent = new Intent(context, WordService.class);
    context.startService(intent);
  }
}
```

```
    <receiver android:name="MyStartServiceReceiver" >
    </receiver>
</application>
```

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

Registro Estático

- A instância usada para processar a mensagem só vive enquanto executa o método onReceive()
- Ao finalizar execução o objeto é descartado
- Este método roda na main thread!
- Se demorar demais, congela UI e pode ser terminado

Por qual razão registrar dinamicamente um BroadcastReceiver?

Quando devemos usar cada método?

Escolhendo cada método

- Existem duas razões para ficar ciente de eventos de sistema:
  - Você deseja oferecer algum tipo de serviço associado a estes eventos
    - app precisa iniciar o trabalho assim que o dispositivo for iniciado, ou inicia alguma tarefa assim que é instalado (*registrar no manifest*)
  - Você deseja reagir *graciosamente* à mudanças de estado
    - eventos que sinalizam mudanças em circunstâncias que seu app depende. Por exemplo, seu app pode usar conectividade enquanto ativo. Neste caso, registro dinâmico seria o mais indicado
- Existem alguns eventos que não é possível se registrar estaticamente para observar. Por exemplo: ACTION_TIME_TICK é disparado a cada minuto.

Cuidados

- Devemos evitar qualquer tipo de tarefa que dure muito tempo nos **`BroadcastReceivers`**.

- Os *receivers* registrados estaticamente e dinamicamente são tratados de forma ligeiramente diferente.

- Em ambos os casos, devemos apenas fazer pequenas tarefas no receiver. Para qualquer tarefa mais longa, um **`Service`** deve ser iniciado.

- Receivers são chamados na thread de UI. Isto significa que qualquer tratamento de UI é bloqueado!

Stop!

Saindo do Stopped State

- Algum outro app tem de usar um Intent explícito para invocar um dos componentes

- Geralmente ao clicar no ícone do app no launcher

Entrando em Stopped State

- Desinstalar o app
- Force-quit ou force-stop no app Settings
- Um reboot não move o app para stopped state…

Ativando e desativando BroadcastReceivers

- BroadcastReceivers são bons quando queremos ser notificados sobre eventos de sistema.

- Em alguns casos, queremos saber sobre algum evento apenas uma vez ou por um curto período de tempo.

- Um receiver registrado dinamicamente nem sempre é aplicável, pois a Activity pode ter sido destruída quando o evento ocorre.

- Para resolver este problema, podemos habilitar e desabilitar os receivers por meio do código.

Ativando e desativando BroadcastReceivers

- Habilitamos ou desabilitamos receivers utilizando métodos da classe PackageManager.

- Com esta classe podemos habilitar e desabilitar componentes em tempo de execução.

```
PackageManager pm = getPackageManager();
ComponentName cName = new
   ComponentName(getApplicationContext(), Receiver.class);
pm.setComponentEnabledSetting(cName,
   PackageManager.COMPONENT_ENABLED_STATE_DISABLED,
   PackageManager.DONT_KILL_APP);
```

Estados suportados por **`setComponentEnabledSetting()`**

Evento	Uso
`COMPONENT_ENABLED_STATE_DEFAULT`	Define o estado para o valor no arquivo de manifest.
`COMPONENT_ENABLED_STATE_DISABLED`	Define estado como desabilitado.
`COMPONENT_ENABLED_STATE_ENABLED`	Define estado como habilitado.

Desabilitando por padrão

- Se desejamos ativar e desativar os receivers em tempo de execução, podemos setar o estado como desabilitado inicialmente.

- Fazemos isto no arquivo de manifest:

```
<receiver
   android:name=".Receiver"
   android:enabled="false" >
   <!-- intent filter -->
</receiver>
```

Quando é útil fazer isto?

Preservar recursos do dispositivo.

Alguns exemplos

- Podemos estar interessado no boot, mas apenas no próximo.

- Neste caso, não podemos utilizar registro dinâmico.

- Desejamos registro estático, mas não queremos que o receiver rode a cada boot.

- Por isso, precisamos desabilitar após a primeira execução.

Alguns exemplos

- Se um ou mais receivers dependem do estado de um serviço específico de sistema…
- podemos desabilitá-los durante o tempo em que o sistema não estiver no estado desejado.
- Rede, bluetooth, GPS, etc.

Alguns exemplos

- Você deseja utilizar notificação, mas apenas se o app não estiver ativo.

- Neste caso, o receiver deve ser habilitado por padrão.

- Desabilitado ao iniciar as Activities da aplicação (**`onResume`**)...

- ...e habilitado novamente no método **`onPause()`**

Outra variação

- *Normal*
  - São completamente assíncronos. Todos os receivers rodam em ordem indefinida, muitas vezes ao mesmo tempo.
  - Mais eficiente, mas significa que não podemos confiar na ordem que os receivers tratarão os eventos.
- *Ordered*
  - São entregues para um receiver de cada vez. A cada receiver, o resultado pode ser propagado ou o broadcast pode ser abortado.
  - A ordem é controlada com o atributo **`android:priority`**. Receivers com a mesma prioridade rodam em ordem arbitrária.

LocalBroadcastManager