Android - Cards de Referência

Broadcast Receivers

Bem vindo ao nosso décimo card sobre Android! :-) Agora, vamos conversar sobre os Broadcast Receivers. Eles são componentes interessantes, pois permitem que sua aplicação se registre para receber eventos do sistema. Mais do que isso, sua aplicação pode lançar também eventos que podem ser capturados por outras aplicações!

BROADCAST RECEIVER

São componentes que permitem que sua aplicação se registre para receber eventos do sistema ou até mesmo de outras aplicações. Uma vez que um evento é lançado, através de uma Intent, todos Broadcast Receivers registrados para "ouvir" este evento serão notificados.

É possível registrar Broadcast Receivers para receber eventos do sistema, como o recebimento de uma chamada de telefone, ou eventos de outras aplicações. Sua própria aplicação pode lançar eventos que serão capturados por outras aplicações.

IMPLEMENTAÇÃO

Um Broadcast Receiver nada mais é do que um objeto do tipo **android.content.BroadcastReceiver**. Você deve criar uma classe nova, estender de **BroadcastReceiver**, implementar o método **onReceive()** e registrar essa classe no **AndroidManifest.xml** do seu projeto.

```
public class PowerBroadcastReceiver
                    extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Broadcast",
           Toast.LENGTH_LONG).show();
   }
}
```

O trecho de código acima é um exemplo de um Broadcast Receiver bastante simples, que só exibe uma mensagem. O próximo trecho de código é como ele deve ser registrado no arquivo de manifesto.

```
<application>
   <receiver android:name=".PowerBroadcastReceiver" >
      <intent-filter>
         <action android:name="com.alienlabz.broadcast" />
      </intent-filter>
   </receiver>
</application>
```

Usamos a tag <receiver> com o atributo name informando o nome da nossa classe que estende de BroadcastReceiver. Precisamos também informar um <intent-filter>, que funciona como um filtro informando quais eventos este Broadcast Receiver está apto a receber.

Neste caso, criamos uma ação nossa, mas poderíamos usar uma ação do sistema. Agora, como fazer com que esse Broadcast Receiver seja chamado?

```
public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Intent intent = new Intent();
      intent.setAction("com.alienlabz.broadcast");
      sendBroadcast(intent);
   }
}
```

É simples. Toda Activity já vem com o método **sendBroadcast(Intent)**. Precisamos apenas criar uma Intent e definir a Action dela como sendo aquela que informamos no AndroidManifest.xml.

ATENÇÃO!

É necessário ter alguns cuidados quando se cria Broadcast Receivers. Primeiro, nunca realize tarefas demoradas. O Android lhe dá um timeout de 10 segundos para concluir as tarefas. Caso contrário, vai eleger seu Broadcast Receiver como "Candidato para Morrer"! :-)

Também não faça operações assíncronas nos Broadcast Receivers quando eles são declarados através do arquivo de manifesto. Após chamar o método **onReceive()**, o Android faz uma "reciclagem" e o Broadcast Receiver não estará mais disponível. Para este tipo de operação, use os Serviços.

Sim, há uma outra forma que podemos usar para registrar Broadcast Receivers: usando o método **registerReceiver(BroadcastReceiver, IntentFilter)** da classe Activity.

```
public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      PowerBroadcastReceiver receiver =
           new PowerBroadcastReceiver();

      IntentFilter intentFilter = new IntentFilter();
      intentFilter.addAction("com.alienlabz.broadcast");
      registerReceiver(receiver, intentFilter);
   }
}
```

O método registerReceiver recebe dois parâmetros. O primeiro é o seu Broadcast Receiver, o segundo, o IntentFilter.

BROADCAST DE SISTEMA

O próprio Android lança alguns eventos que podem ser capturados por sua aplicação. Eles estão definidos como constantes na classe Intent. Temos alguns que são bastante usados, são eles:

Intent.ACTION_BOOT_COMPLETED - Lançado assim que o Boot do Android foi completado. Requer uma permissão no manifest (android.permission.RECEIVE_BOOT_COMPLETED).

Intent.ACTION_POWER_CONNECTED - Lançado assim que o usuário conecta o dispositivo a um cabo de energia.

Intent.ACTION_POWER_DISCONNECTED - O inverso do anterior! :-)

Intent.ACTION_BATTERY_LOW - Lançado quando a bateria está pouca.

Intent.ACTION_BATTERY_OKAY - Lançado quando enquanto a bateria está boa.

Existem outras classes que declaram constantes de broadcasts de sistema. Um exemplo é a classe **TelephonyManager**.

TelephonyManager.ACTION_PHONE_STATE_CHANGED - Lançado quando o status do telefone muda. Requer a permissão **READ_PHONE_STATE**. Vamos ver como usar este último.

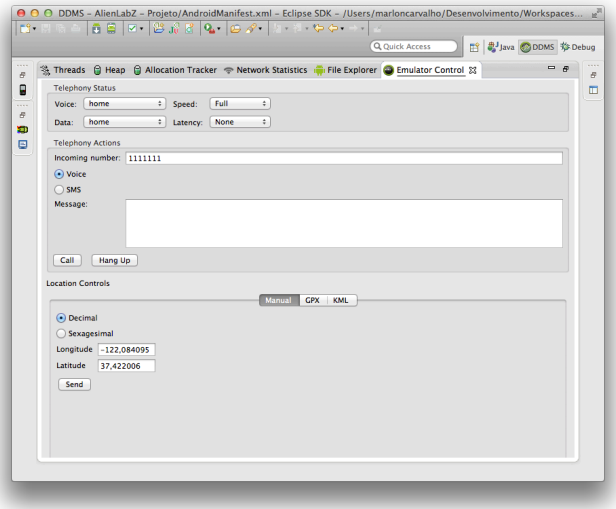
Vamos usar a classe que já criamos, que apenas exibe um Toast. Precisamos modificar apenas duas coisas. Primeiro, a ACTION no AndroidManifest.xml. Neste arquivo, também precisamos pedir permissão para ler o estado do telefone.

```
<manifest xmlns:android
   package="com.alienlabz.projeto"
   android:versionCode="1"
   android:versionName="1.0" >

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

   <application>
      <receiver android:name=".PowerBroadcastReceiver" >
         <intent-filter>
            <action
android:name="android.intent.action.PHONE_STATE" />
         </intent-filter>
      </receiver>
   </application>
</manifest>
```

Estamos omitindo algumas informações no trecho de código acima. Deixamos apenas as informações mais relevantes. Primeiro, temos a permissão com a tag <uses-permission>. Depois, mudamos a tag <action>, colocando o name para **android.intent.action.PHONE_STATE**.

Basta isto, agora, para testar, bastar instalar o aplicativo e realizar uma chamada para o dispositivo. Não sabe como simular uma chamada no emulador do Android? Vamos ver agora.

No Eclipse, abra a Perspectiva chamada DDMS. Você terá uma aba chamada **Emulator Control**. Você verá um campo chamado **Incoming Number**. Logo abaixo tem um botão **Call**. Digite qualquer número e clique em Call. Mude rapidamente para o emulador para ver o resultado!



STICKY INTENTS

O Android também pode lançar Intents (broadcast) que são chamados de Sticky. Isto significa que mesmo após a execução dos Broadcast Receivers, eles continuam “vivos”.

Enfim, eles não são finalizados e estão disponíveis para que os próximos registros com **registerReceiver()** recebam eles. Quando fazemos o broadcast de um Intent, normalmente eles deixam de existir assim que são processados pelos Broadcast Receivers. Não é o caso do Sticky Intent.

Este tipo de Intent é interessante, pois quem se registra para recebê-lo não precisa aguardar um próximo broadcast. Ele é retornado imediatamente. Um exemplo de uso? Receber o status da bateria. Você não precisa registrar um Broadcast Receiver e aguardar o próximo broadcast para saber como está a bateria. Você obtém de imediato o último status.

E aqui vem mais um detalhe interessante. Você sequer precisa passar um Broadcast Receiver para o método **registerReceiver()**. Pode passar **NULL** que lhe será retornado o último Intent. Vamos fazer um exemplo para obter o status da bateria?

```
public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      IntentFilter filter =
           new IntentFilter(Intent.ACTION_BATTERY_CHANGED);

      Intent batteryStatus = this.registerReceiver(null, filter);
      int status =
            batteryStatus.getIntExtra
                  (BatteryManager.EXTRA_STATUS, -1);
      // Fazer algo aqui com o status!
   }
}
```

No trecho acima, estamos apenas criando um IntentFilter e passando NULL para **registerReceiver()**. De imediato, já temos o último status da bateria. Você também pode registrar um Broadcast Receiver, que processará este mesmo Intent retornado.

Quem normalmente lança broadcasts? Serviços. Eles ficam executando em background no seu dispositivo. Quando algum evento importante acontece, ele lança um evento para notificar todos os Broadcast Receivers interessados nele. Existe um serviço que checa o status da bateria e de tempos em tempos lança um Sticky Intent atualizando essa informação.

Um último detalhe: para lançar Sticky Intents a partir de sua aplicação, é necessário pedir permissão no AndroidManifest: BROADCAST_STICKY

ALARM MANAGER

Existe um serviço no Android chamado **AlarmManager**. Ele é uma ótima forma para você agendar tarefas. Você deve usar ele com Broadcast Receivers e Pending Intents. Vamos ver um exemplo. Primeiro, precisamos criar nosso BroadcastReceiver.

```
public class AlarmeBroadcastReceiver
                     extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "ALARME!",
            Toast.LENGTH_LONG).show();
   }
}
```

Vamos declarar ele no nosso AndroidManifest.xml. Não precisamos informar INTENT FILTERS, neste caso.

```
<manifest xmlns:android
   package="com.alienlabz.projeto"
   android:versionCode="1"
   android:versionName="1.0" >
   <application>
      <receiver android:name=".AlarmBroadcastReceiver" >
      </receiver>
   </application>
</manifest>
```

Agora, vamos registrar esse Broadcast Receiver no serviço de Alarme do Android.

```
public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Intent intent =
           new Intent(this, AlarmBroadcastReceiver.class);

      PendingIntent pendingIntent =
         PendingIntent.getBroadcast
            (this.getApplicationContext(), 1, intent, 0);

      AlarmManager alarmManager =
            (AlarmManager) getSystemService(ALARM_SERVICE);

      alarmManager.set(AlarmManager.RTC_WAKEUP,
            System.currentTimeMillis() + (2 * 1000),
            pendingIntent);
   }
}
```

Precisamos criar uma Intent de forma parecida como fazemos para chamar uma nova Activity. A diferença é que passamos a classe do nosso Broadcast Receiver.

Ainda lembra sobre as Intents? Estamos fazendo uma Intent Explícita, informando que quem tratará ela será nosso Broadcast Receiver.

Agora, precisamos de uma Pending Intent. Consegue entender o motivo? É uma tarefa que será executada depois e também passamos as permissões de execução de nossa aplicação.

Agora, basta obter o serviço de Alarme chamando **getSystemService(ALARM_SERVICE)**. Com esse serviço em mãos, temos o método **set()** que recebe três parâmetros.

Tipo - Define a forma como o tempo decorrido será observado e como o Android irá se comportar quando o alarme for acionado. Pode assumir quatro valores.

Constante	Descrição
ELAPSED_REALTIME	Tempo a partir do boot, incluindo o tempo de sleep (usa o tempo de **SystemClock.elapsedRealTime()**). Não despertará o dispositivo para entregar a Intent. Aguarda a próxima vez que o dispositivo for despertado.
ELAPSED_REALTIME_WAKEUP	Igual ao ELAPSED_TIME, com a diferença de que o dispositivo será despertado.
RTC	Usa o tempo de **System.currentTimeMillis()** e não desperta o dispositivo para entregar a Intent. Aguarda a próxima vez que o dispositivo for despertado.
RTC_WAKEUP	Usa o tempo de **System.currentTimeMillis()** e desperta o dispositivo.

Tempo - Tempo para o alarme ser disparado. Em milissegundos.

Ação - A ação a ser executada quando o tempo for atingido. É uma Intent.

BROADCASTS INTERNOS

Já observou uma peculiaridade na forma que fizemos broadcasts até agora? Eles estão sempre disponíveis para todas as aplicações, certo?

Se faço um broadcast de nossa aplicação, ele pode ser capturado em outra aplicação. Será que é isso que sempre queremos? Nem sempre.

Há situações em que você quer garantir que seus broadcasts não passem a barreira de sua própria aplicação, por questões de segurança.

Um broadcast local também é mais rápido. O que muda? Apenas poucos detalhes. Não usamos mais o método **sendBroadcast(Intent)** que está na classe Activity. Usamos um método de mesma assinatura, mas que se encontra na classe **LocalBroadcastManager**. Também passamos a usar o método **registerReceiver()** desta classe.

Mais alguns detalhes. O LocalBroadcastManager foi adicionado apenas a partir da versão 3.X do Android. Portanto, lembre-se de adicionar a biblioteca de compatibilidade para poder usar esta funcionalidade. Por último, você deve usar o método **registerReceiver()** dessa nova classe. Caso registre o BroadcastReceiver no **AndroidManifest.xml**, ele estará registrado para receber eventos de outras aplicações.

```
public class MainActivity extends Activity {
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      IntentFilter filter =
            new IntentFilter("com.alienlabz.broadcast");

      LocalBroadcastManager.getInstance(this).
            registerReceiver(new PowerBroadcastReceiver(), filter);

      Intent intent = new Intent();
      intent.setAction("com.alienlabz.broadcast");

      LocalBroadcastManager.getInstance(this).
            sendBroadcast(intent);
   }
}
```

BROADCASTS ORDENADOS

A forma como disparamos broadcasts até agora implica que eles serão tratados em qualquer ordem pelos Broadcast Receivers. Podem ser até ao mesmo tempo. E caso você não queira isso? E se quiser que seja feita uma fila, no qual cada Broadcast Receiver trata e repassa para o próximo? E um pode até cancelar o envio para o próximo!

Isso é possível e muda apenas UM detalhe. É o método que usamos. Neste caso, devemos usar **sendOrderedBroadcast()**. A ordem você define com o atributo Priority da **IntentFilter** ou o atributo **android:priority** na tag <intent-filter>

Sobre o Autor

Marlon Silva Carvalho

É um programador de longa data. Já peregrinou por diversas linguagens e hoje se considera um programador agnóstico. Atualmente, está fascinado pelo mundo mobile e suas consequências para a humanidade. Está mais focado no desenvolvimento para Android, embora também goste de criar aplicativos para iOS.

Trabalha em projetos sobre mobilidade no SERPRO, tendo atuado no projeto Pessoa Física, para a Receita Federal do Brasil. Você pode encontrar ele no Twitter, no perfil @marlonscarvalho, em seu blog, no endereço http://marlon.silvacarvalho.net e através do e-mail marlon.carvalho@gmail.com.

Sobre os Cards

Caso você já tenha visto os excelentes RefCards da DZone, deve ter percebido a semelhança. E é proposital. A ideia surgiu após acompanhar estes RefCards por um bom tempo. Contudo, eles são mais gerais. O objetivo destes Cards é tratar assuntos mais específicos. Trazendo informações relevantes sobre eles.

Caso tenha gostado, continue nos acompanhando! Tentaremos trazer sempre novos cards sobre os mais variados assuntos. Inicialmente, focaremos no Android.

O ícone principal deste card é de autoria de Wallec e foi obtido em seu profile no DeviantArt.
http://wwalczyszyn.deviantart.com/