Android — Serviços

Flávio Velloso Laper

Universidade Fumec

28 de maio de 2015

Programa da aula

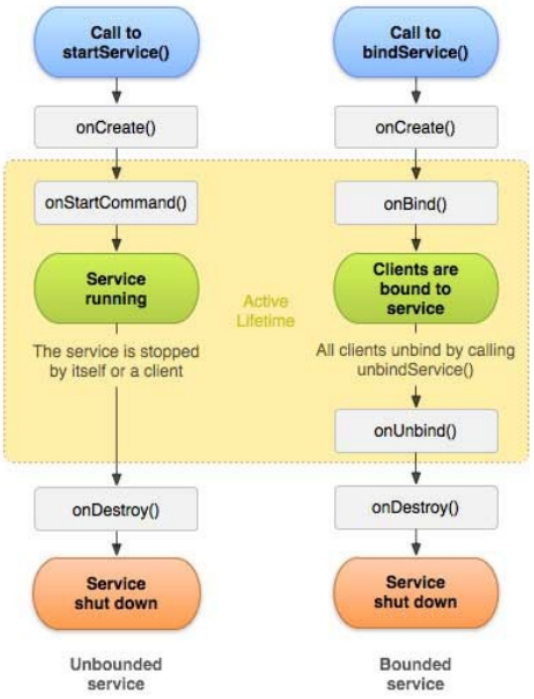
Conteúdo

Serviços

- Serviço: uma tarefa de uma aplicação executada em *background*.
  - Exemplo: *Google Play Music* executa músicas usando um serviço.
  - Exemplo: um navegador executa um serviço de *download* para recuperar um arquivo.
  - Útil para tarefas de longa duração e/ou prover funcionalidades que possam ser usadas por outras aplicações.
- Android tem dois tipos de serviços:
  - serviços padrão: para tarefas longas; permanecem em execução após o fechamento da aplicação;
  - serviços de *intent*: para tarefas curtas; são lançados pela aplicação via *intents*.
- Quando (ou se) o serviço terminar seu trabalho, ele pode divulgar isto fazendo um *broadcast* para quaisquer receptores (*receivers*) que estejam na escuta.

Ciclo de vida de um serviço

- Um serviço é iniciado pela atividade de uma aplicação usando um *intent*.
- Modos de operação:
  - *start*: o serviço continua rodando até ser manualmente parado.
  - *bind*: o serviço continua rodando até que não haja nenhuma aplicação "ligada" (*bound*).
- Os métodos de ciclo de vida de um serviço são semelhantes aos de uma atividade.
  - *onCreate*, *onDestroy*.



Template de classe de serviço

O *Android Studio* possui um *wizard* para a criação de serviços.

```
public class ServiceClassName extends Service {
    /* this method handles a single incoming request
       */
    @Override
    public int onStartCommand(Intent intent, int flags
        , int id) {
        // unpack any parameters that were passed to
           us
        String value1 = intent.getStringExtra("key1");
        String value2 = intent.getStringExtra("key2");
        // do the work that the service needs to do
           ...
        return START_STICKY;
        // stay running
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
        // disable binding
    }
}
```

Mudanças em *AndroidManifest.xml*

- Para obter permissão para usar o serviço, a *tag* a seguir deve ser adicionada à configuração em *AndroidManifest.xml*.
  - O atributo *exported* indica se outras aplicações têm permissão de usar o serviço.
  - O ponto (·) antes do nomes da classe do serviço é obrigatório.

```
<application ...>

  <service
      android:name=".ServiceClassName"
      android:enabled="true"
      android:exported="false" />
```

Inicialização de serviços

- Na classe da atividade:

```
Intent intent = new Intent(this,
    ServiceClassName.class);
intent.putExtra("key1", "value1");
intent.putExtra("key2", "value2");
startService(intent);
// not startActivity!
```

- De um fragmento:

```
Intent intent = new Intent(getActivity(),
                           ServiceClassName.class);
```

Ações dos *intents*

- Frequentemente, um serviço possui diversas "ações" ou comandos que pode executar.
  - Um serviço *music player* pode tocar, pausar, parar, ...
  - Um serviço de *chat* pode envia, receber, ...
- Android implementa isto com métodos *set/getAction* no *intent*.
  - Na classe da atividade:

```
Intent intent = new Intent(this, ServiceClassName
    .class);
intent.setAction("some constant string");
intent.putExtra("key1", "value1");
startService(intent);
```

  - Na classe do serviço:

```
String action = intent.getAction();
if (action.equals("some constant string")) { ...
    } else { ... }
```


Divulgação de resultado

- Quando um serviço completar uma tarefa, ele pode notificar a aplicação enviando um *broadcast* que seja ouvido por ela.
  - Um *action* pode ser usado no *intent* para distinguir entre diferentes tipos de resultados.

```
public class ServiceClassName extends Service {
    @Override
    public int onStartCommand(Intent tent, int flags,
        int id) {
        // do the work that the service needs to do
            ...
        ...
        // broadcast that the work is done
        Intent done = new Intent();
        done.setAction("action");
        done.putExtra("key1", value1); ...
        sendBroadcast(done);
        return START_STICKY;
    }
```

Recepção de *broadcast*

- A atividade pode escutar *broadcasts* usando um *BroadcastReceiver*.
  - Estender *BroadcastReceiver* com o código para manipular a mensagem.
  - Quaisquer parâmetros extras na mensagem vêm do *intent* do serviço.

```
public class ActivityClassName extends Activity {
    ...
    private class ReceiverClassName extends
        BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent
            intent) {
            // handle the received broadcast message
            ...
        }
    }
```

Audição de *broadcasts*

- Configurar a atividade para ser notificada quando uma determinada ação de *broadcast* ocorrer.
  - Um *filtro de intent* deve ser passado especificando as ações de interesse.

```
public class ActivityClassName extends Activity {
    @Override
    protected void onCreate(Bundle
        savedInstanceState) {
        ...
        IntentFilter filter = new IntentFilter();
        filter.addAction("action");
        registerReceiver(new ReceiverClassName(),
            filter);
    }
```

Conteúdo

Serviços e *Threads*

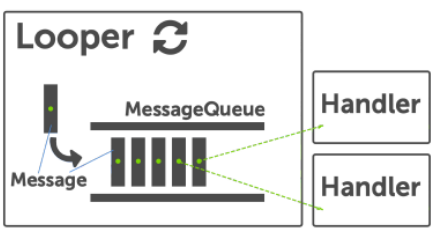
- Por padrão, um serviço vive no mesmo processo e na mesma *thread* da atividade que o criou.
  - Isto não é interessante para tarefas de longa duração.
  - Se o serviço estiver ocupado, a interface da aplicação pode congelar.
  - Exemplo: durante um *download* de um arquivo grande, outros elementos da interface do usuário podem parar de responder.
- Para deixar o serviço e a aplicação mais independentes e responsivos, o serviço deveria manipular suas tarefas em *threads* separadas.

Serviços com *Threads*

```
public class ServiceClassName extends Service {
  /* this method handles a single incoming request */
  @Override
  public int onStartCommand(Intent intent,
                            int flags, int id) {
    // unpack any parameters that were passed to us
    String value1 = intent.getStringExtra("key1");
    Thread thread = new Thread(new Runnable() {
      public void run() {
        // do the work that the service needs to do
      }
    });
    thread.start();
    return START_STICKY;
  }
```


Classes auxiliares para *threads*

- Fila de *jobs* (ou mensagens): padrão comum em Android.
  - Novas tarefas (*jobs*) entram em serviço via *intents* da aplicação.
  - *Jobs* são enfileirados em algum tipo de estrutura para processamento posterior.
  - A(s) *thread*(s) processam os *jobs* de acordo com a ordem de chegada.
  - Ao terminar, o *job* faz um *broadcast* do resultado para a aplicação.
- Android fornece diversas classes para auxiliar na implementação de filas *multithread*:
  - *Looper*, *Handler*, *HandlerThread*, *AsyncTask*, *Loader*, *CursorLoader*, ...
  - Vantagens: facilidade na submissão e finalização de *jobs*; sincronização simplificada; possibilidade de cancelamento; melhor integração com questões do ciclo de vida do Android; ...



HandlerThread

- *HandlerThread*: *thread* que possui dados internos representando uma fila de *jobs* a executar.
  - *Looper*: vive dentro de um *handler thread* e executa um laço *while* de longa duração que aguarda por *jobs* e os processa.
  - É possível fornecer novos *jobs* para o *handler thread* via seu *looper*.

```
HandlerThread hThread = new HandlerThread("name");
hThread.start();
Looper looper = hThread.getLooper();
...
```

Handler

- Representa uma peça de código para manipular um *job* na fila.
  - Ao construir um *handler*, deve-se passar o *Looper* do *handler thread* no qual o *job* deve ser executado.
  - Submeter o *job* para o *handler* chamando o método *post*, passando um objeto *Runnable* indicando o código a executar.

```
Handler handler = new Handler(looper);
handler.post(new Runnable() {
  public void run() {
    // the code to process the job
    ...
  }
});
```

Exercício

Desenvolva uma pequena aplicação para enviar *alarmes* para o usuário:

- Inicialmente, a aplicação deve apresentar uma tela para que o usuário informe uma mensagem e um intervalo de tempo.
  - Alternativamente, o usuário pode informar também uma data/hora.
- Após o pressionamento de um botão, a mensagem deve ser registrada para exibição após o tempo requisitado.
- Transcorrido o intervalo de tempo, a aplicação deve exibir a mensagem para o usuário.
  - Utilize uma notificação ou um *Toast*.
- Entregue o exercício na data estipulada pelo professor.