Programação Dispositivos Móveis

Prof. Rogério Fontes
@rogeriofontes
ADS - Unipac

Activity


Activity
Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack .
When a new activity starts, it is pushed onto the back stack and takes user focus.

Activity


Activity
An Activity is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
Each activity is given a window in which to draw its user interface.
An application usually consists of multiple activities that are loosely bound to each other.

Activity

- Activity é uma classe que representa uma/ou mais tarefas, muito focada, no que um usuário pode fazer em uma aplicação Android.

- A maioria das atividades fazem interação com usuário.

- A classe de atividade controla todo processo de interação com S.O e o usuário, como criar telas e dimensões, conectar banco de dados, e demais ações de integração com usuário.

Ciclo de Vida

- Atividades no sistema são gerenciadas como um activity stack ou pilha de atividades em português. Quando uma atividade é iniciada, ela é colocada no topo da pilha e se torna a atividade corrente - a atividade anterior sempre permanece abaixo na pilha e não vai ser mostrada enquanto a atividade corrente não terminar.

Ciclo de Vida

- Uma atividade tem quatro estados essenciais:

- Se uma atividade está sendo executada e está sendo mostrada na tela (que é o topo da pilha), ela está em modo active ou running.

Ciclo de Vida

- Uma atividade tem quatro estados essenciais:

- Se uma atividade perdeu o foco mas ainda assim está visível (ou seja, uma nova atividade está sendo mostrada na tela mas não ocupando-a completamente - atividades em janelas flutuantes, por exemplo, ela está em modo paused. Uma atividade em modo paused está completamente viva (ela mantém todos os estados e informações e mantém-se relacionada ao gerenciador de janelas), mas pode ser encerrado pelo sistema em situações de memória baixa extremas.

Ciclo de Vida

- Uma atividade tem quatro estados essenciais:

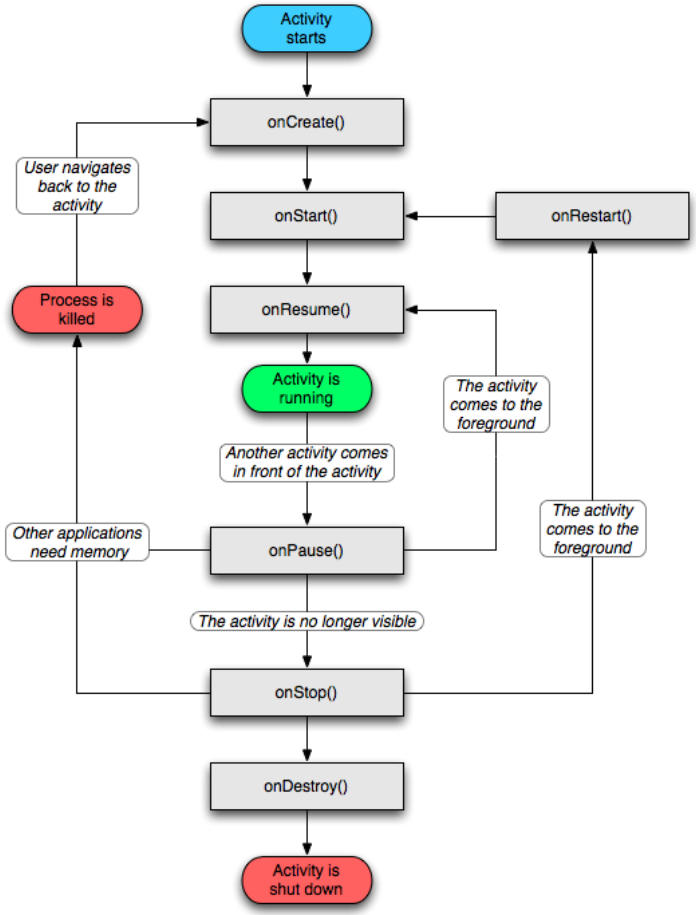
- Se uma atividade é completamente obscurecida por outra atividade, ela está em modo stopped. Ela ainda retém o seu estado e informações, contudo não é mais visível pelo usuário e sua janela está escondida e pode acontecer de ser encerrada pelo sistema quando for necessário liberar memória.

Ciclo de Vida

- Uma atividade tem quatro estados essenciais:

- Se uma atividade está em modo paused ou stopped, o sistema pode retirar a atividade da memória simplesmente pedindo a ela que seja finalizada ou simplesmente matando o seu processo. Quando é mostrada novamente ao usuário, ela terá de ser novamente reiniciada e restaurada para seu estado anterior.

Ciclo de Vida



Ciclo de vida

- Existem três laços que você pode achar interessante monitorar dentro de sua atividade:

- O ciclo de vida completo de uma atividade acontece entre a primeira chamada do onCreate(Bundle) até o onDestroy(). Uma atividade vai fazer toda a configuração do estado "global" no onCreate() e liberar os recursos remanescentes em onDestroy(). Por exemplo, se você tem uma thread rodando em background para fazer o download de dados da rede, você deve criar essa thread em onCreate() e então pará-la em onDestroy().

Ciclo de vida

- Existem três laços que você pode achar interessante monitorar dentro de sua atividade:

- O vida visível de uma atividade acontece entre o onStart até o onStop. Durante esse tempo o usuário pode ver a atividade na tela, apesar de às vezes não estar completamente visível ao usuário. Entre esses dois métodos você pode manter os recursos que são necessários para mostrar a atividade ao usuário. Por exemplo, você pode registrar um BroadcastReceiver no onStart() para monitorar as mudanças que impactam a interface. Os métodos onStart() e onStop() podem ser chamados múltiplas vezes enquanto a atividade se torna visível e escondida do usuário.

Ciclo de vida

- Existem três laços que você pode achar interessante monitorar dentro de sua atividade:

- O ciclo de vida de uma atividade é definida pelos métodos de atividade mostrados abaixo. Todos eles podem ser sobrescritos para fazer o trabalho apropriado quando um estado de atividade mude. Todas as atividades vão implementar onCreate(Bundle) para que as mudanças de dados sejam persistidas e, de outra maneira, preparar para encerrar a interação com o usuário. Voc}e pode sempre chamar a superclasse quando implementando um desses métodos.

Ciclo de vida

```
public class Activity extends ApplicationContext {
        protected void onCreate(Bundle savedInstanceState);
        protected void onStart();
        protected void onRestart();
        protected void onResume();
        protected void onPause();
        protected void onStop();
        protected void onDestroy();
}
```

o movimento através do ciclo de vida das atividades

Método	Método	Pode ser morto?	Próximo
onCreate()	Chamado quando a atividade é criado. Aqui é quando você deve fazer todas as funções como: criar as views, linkar os dados às listas, etc.	Não	onStart()
onRestart()	Chamado após a atividade ser parada e antes de ser reinciada. Sempre seguida por onStart()	Não	onStart()
onStart()	Chamado quando a atividade se torna visível ao usuário. Seguido pelo onResume() se a atividade roda na frente ou por onStop() se ela se torna invisível.	Não	onResume()ou onStop()

o movimento através do ciclo de vida das atividades

Método	Método	Pode ser morto?	Próximo
onResume()	Chamado quando a atividade vai iniciar a interação com o usuário. Nesse ponto, sua atividade está no topo da pilha de atividades e quaisquer dados que sejam inseridos serão feitos aqui. Sempre seguido pelo onPause().	Não	onPause()

o movimento através do ciclo de vida das atividades

Método	Método	Pode ser morto?	Próximo
onPause()	Chamado quando o sistema está por resumir a atividade anterior. Isso é tipicamente usado para persistir quaisquer mudanças ainda não efetivadas, parar animações e oturas coisas que possam consumir a CPU, etc. Implementações desse método devem ser rápidos pois a próxima atividade não será mostrada até que esse método seja finalizado. Seguido por onResume() se a atividade retornar para a frente ou onStop() se ela se tornar invisível ao usuário.	Sim	onResume()ou onStop()

o movimento através do ciclo de vida das atividades

Método	Método	Pode ser morto?	Próximo
onStop()	Chamado quando a atividade não mais estiver visível ao usuário, pois outra atividade foi resumida e está na frente desta. Isso pode acontecer porque outra atividade está sendo iniciada, uma atividade existende está sendo trazida para a frente ou essa atividade estiver sendo finalizada. Seguida pelo onRestart() se essa atividade está voltando para interagir com o usuário ou onDestroy() se a atividade estiver sendo encerrada.	Sim	onRestart()ou onDestroy()

o movimento através do ciclo de vida das atividades

Método	Método	Pode ser morto?	Próximo
onDestroy()	A chamada final que você receberá antes que a atividade seja destruída ou finalizada. Isso pode acontecer porque a atividade está, de fato, sendo encerrada (alguém chamou finish() nela) ou porque o sistema está temporariamente destruindo a instância da atividade para aumentar o espaço na memória. Você pode distinguir entre esses cenários com o método isFinishing()	Sim	*nada*

Android overview

Gerenciador de Layouts

Gerenciador de Layouts

- No Android todos os componentes gráficos são gerados através de subclasses da classe View, tudo que é gráfico vem da classe View.

Gerenciador de Layouts

- Os principais Layouts no Android são:
  - Linear Layout (horizontal ou vertical)
  - Absolute Layout
  - Relative Layout
  - Table Layout ou Frame Layout
  - Grid Layout

Gerenciador de Layouts

- Layouts em Android são construidos em XML.
- Qualquer arquivo XML de interface do Android necessita obrigatoriamente de um Layout raiz. Sendo possível vários layout aninhados.
- todo Layout contem dois atributos básicos: Width (Largura) e Height (Altura).

Gerenciador de Layouts

- Todo Layout obedece seu pai, ou seja, Ao inserir Layouts e Widgets dentro de Layouts os mesmos devem obedecer o tamanho de seu pai. Para isso podemos deixar que ele se ajuste na largura máxima até seu pai ou somente a largura necessária para englobar seu conteúdo.

Gerenciador de Layouts

- Os atributos básicos que auxiliam nessas configurações são:
- fill_parent
- wrap_content

- O atributo FILL_PARENT indicará que o componente utilizará toda altura ou largura de seu pai.
- O atributo WRAP_CONTENT indicará que o componente deve utilizar apenas tamanho suficiente para receber seu conteúdo.

Resoluções para várias telas

- No Android é preciso um esforço para tentar encaixar as imagens para as mais variadas telas, entretanto, algumas restrições tem que ser respeitadas, para garantir que tanto o tamanho, quanto a resolução, ou seja, a densidade de informação que cabe na tela sejam respeitados.

Os Gerenciadores de Layout

- Os gerenciadores de layout estão sob o pacote android.widget do Android SDK. Esse pacote nos permite exibir seus componentes de layout. Esta classe está disponível desde a API Level 1. Veja sua posição na hierarquia de classes do Android:

  java.lang.Object

  android.view.View

  android.view.ViewGroup

  android.widget.LinearLayout

  android.widget.FrameLayout

  ...

Principais gerenciadores de Layout

- AbsoluteLayout : Posiciona os componentes fornecendo coordenadas x e y.
- FrameLayout : Tipo mais comum e simples de layout. Usado quando um componente deve preencher a tela toda.
- LinearLayout : Utilizado para organizar os componentes na horizontal ou na vertical.

Principais gerenciadores de Layout

- TableLayout : Pode ser usado para organizar os componentes em uma tabela, com linhas e colunas.
- RelativeLayout : Permite posicionar um componente relativo a outro, por exemplo, abaixo, acima ou ao lado de um componente já existente.

LinearLayout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
</LinearLayout>
```

LinearLayout

```
package arquivo.estudos;
//imports....

public class EstudosActivity extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
        // vamos criar o gerenciador de layout
        LinearLayout layout = new LinearLayout(this);
        layout.setOrientation(LinearLayout.VERTICAL);

        layout.setLayoutParams(new LinearLayout.LayoutParams( LayoutParams.FILL_PARENT,
        LayoutParams.FILL_PARENT));

          // vamos atribuir este layout à janela
         setContentView(layout);
}
```

Absolute Layout (depreciado)

- ```
  <?xml version="1.0" encoding="utf-8"?>
  <AbsoluteLayout
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     xmlns:android="http://schemas.android.com/apk/res/android"
     >
     <!-- Button 1 position mentioned using layout x and layout y attributes-->
     <Button
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:text="simplecode"
        android:layout_x="126dp"
        android:layout_y="361dp"
        />
       <!-- Button 2 position mentioned using layout x and layout y attributes-->

   <Button
        android:layout_width="113dp"
        android:layout_height="wrap_content"
        android:text="stuffs.com"
        android:layout_x="12dp"
        android:layout_y="361dp"
        />
  </AbsoluteLayout>
  ```
  ```

FrameLayout Example

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
android:id="@+id/framelayout" >
<TextView
    android:id="@+id/frameText" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="This is a text"
    android:textSize="15sp"
    android:textStyle="bold"
    android:visibility="gone" android:layout_gravity="center" />
</FrameLayout>
```

TableLayout

```
<?xml version="1.0" encoding="utf-8"?>
 <TableLayout
        xmlns:android="http://schemas.android.com/apk/res/
        android" android:id="@+id/tableLayout1"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent" >
<TableRow android:id="@+id/tableRow5"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="5dip" >
         <Button android:id="@+id/button5"
        android:layout_column="1"
        android:text="Column 2" />
    </TableRow>
    </TableLayout>
```

- **<?xml** version="1.0" encoding="utf-8"**?>**
  <RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent" **>**
- **</RelativeLayout>**

demos

- http://github.com/rogeriofontes/androidgraphicdemos

Resoluções para várias telas

Conceitos:

- **Screen size**
  - Tamanho físico da tela medido a partir da diagonal. Para simplificar, o Android dividiu em alguns grupos: pequeno, normal, grande e extra grande.

Resoluções para várias telas

Conceito:

- **Screen density**
  - Quantidade de pixels que uma porção da tela, para ter um parâmetro, é adotado **dpi (dots per inch)**. Isso significa que uma tela com uma baixa densidade tem poucos pixels em uma parcela da tela. Para simplificar, o Android dividiu em: baixo, médio, alto e extra alto

Resoluções para várias telas

Conceito:

- **Orientation**
  - A orientação da tela depende do ponto de vista, mas consideramos a partir da visão do usuário podendo ser paisagem e retrato, é importante ressaltar que além de as telas poderem trabalhar em diferentes orientações, algumas telas mudam automaticamente com os acelerômetros.

Resoluções para várias telas

Conceito:

- **Resolution**
  - O número total de pixels físicos na tela. Para criar um aplicação para várias telas diferentes devemos nos preocupar com a densidade e tamanho de tela e não resolução.

Resoluções para várias telas

Conceito:

- **Density-independent pixel (dp)**
  - É uma proporção, um pixel virtual utilizado para normalizar as dimensões no momento em que for feito o layout do seu software e para que se adeque da melhor maneira possível à maioria das interfaces utiliza-se para uma tela de densidade média. Em tempo real o sistema trata essas variações de pixels virtuais transformando em pixels reais. Para uma tela média consideramos que a tela tem 160 dp, então a conversão para pixels reais é: px = dp * (dpi / 160)..

Resoluções para várias telas

Conceito:

- **Density-independent pixel (dp)**
  - > px = dp * (dpi / 160).
  - As interfaces são separadas por tamanho em:
    - small, normal, large e xlarge
    - escala de densidade:
    - ldpi, mdpi, hdpi e xhdpi

Obrigado