Linux Kernel: Camada de abstração entre hw e as camadas restantes, fornecendo serviços de SO genéricos • Permissões, gerenciamento de memória e processos, IO de arquivos e rede. DRM, armazenamento externo, Gráficos, entrada, media = meios de comunicação, sensores e tv, drives: Áudio, bluetooth, câmera, teclado, WIFI, display, binder (IPC) = encadernador;

Hardware Abstraction Layer (HAL) = Camada de Abstração de Hardware: Fornece interfaces padronizadas para expor hardware à camadas superiores ● Múltiplos módulos especiﬁcando componentes de hardware, como câmera, bluetooth ● Ao fazer uma chamada que acessa hardware, o sistema carrega o módulo correspondente. Áudio, Bluetooth, câmera, sensores.

Native C/C++ Libraries = Bibliotecas C / C ++ nativas: Componentes “core” Android são construídos a partir de código nativo, que usam bibliotecas C/C++ ● Algumas são expostas a desenvolvedores através do Android Application Framework, como OpenGL ● Aplicativos desenvolvidos com uso de C/C++ podem usar a NDK para acessar estas bibliotecas diretamente. Webkit, OpenMAX AL, Libc, Media Framework, OpenGL ES...

Android Runtime:

Android Runtime (ART)

Core Librieries

Core Libraries:

* Bibliotecas ● Fornecem maioria das funcionalidades disponíveis nas bibliotecas padrão
* Java ● java.\* e javax.\* packages
* pacotes especíﬁcos ao ciclo de vida das aplicações, android.\*
* junit.\* - testes

Android Runtime (ART):

* Ambiente onde cada aplicação Android roda

● Cada aplicação roda seu próprio processo, com sua própria instância de ART

● Escrita para possibilitar a um dispositivo rodar múltiplas VMs eﬁcientemente

● Executa arquivos no formato DEX (bytecode)

● Até Android 5.0 (API 21), Dalvik era padrão

**Java API Framework**

Content Providers Managers

View System Activity Location Package notification

Resource Telephony Window

* APIs fornecem o fundamento para criação de aplicativos, ao permitir, de forma simpliﬁcada, o reuso de componentes e serviços do sistema;
* Desenvolvedores têm acesso total às mesmas APIs utilizadas pelos core apps.

**System Apps**

Dialer Email Calendar Camera ...

* Android fornece um conjunto de aplicativos:

● Cliente de email, SMS, calendário, mapas, navegador, contatos, etc. ● Estas aplicações não tem status especial, podem ser substituídas (exceto apps como Settings)

● Fornecem funcionalidades que podem ser compartilhadas.

**Componentes Android**

Activities, services, contente providers e broadcast receivers = receptores de radiofusão.

Activity

• Classe básica para interação com usuário, análoga a uma janela ou dialog em app desktop

• Representa uma tela com interface para o usuário

• Em geral, pode ser um ponto de entrada da aplicação • Exemplo: Email App

• Em alguns dispositivos, já é possível lidar com mais de uma activity ao mesmo tempo

Service

• Activities tem vida curta e podem ser encerradas a qualquer momento • Services são pontos de entrada de propósito geral que permitem que um app continue rodando em background

•Componentes que foram projetados para continuar rodando independente de UI

•Por exemplo, Music Player app, live wallpaper, serviços de acessibilidade, entre outros…

BroadcastReceiver = Receptor de radiodifusão

• O sistema e outros aplicativos enviam eventos de broadcast de tempos em tempos

• Escuta e responde a eventos que acontecem “fora” do ﬂuxo regular do usuário no app

• Desempenham papel de subscriber no padrão publish/subscribe • Exemplo: Messaging application

ContentProviders = Provedores de conteúdo

• Gerencia um conjunto compartilhado de dados do app que podem estar armazenados no sistema de arquivos, banco de dados SQLite, web, ou em outro local de persistência…

• Por meio deste componente, é deﬁnida uma camada de abstração que permite compartilhar dados entre aplicações

•Interfaces similares à bancos de dados

• Gerenciam interprocess communication

• Permitem controlar acesso aos dados • Exemplo: Browser, contatos

**Apps e Packages**

• Dado que você escreveu toda sua aplicação, as ferramentas de build de Android geram aplicação

• Formato APK é utilizado para distribuição

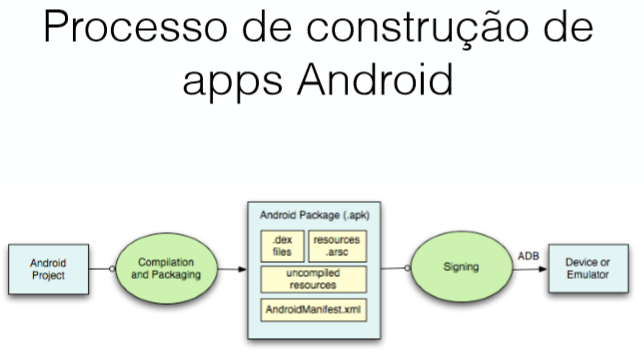
• Todo app tem um package name, também chamado de application id

• nome válido de acordo com regras Java

• único no dispositivo e na play store

• Ao criar um projeto declaramos o package name

Android roda múltiplos apps (activities e services) em paralelo



Criando um app

• Deﬁnir recursos; • Implementar classes da aplicação; • Empacotar o app;

• Instalar e rodar.

Instant Run

• Alguns dispositivos vão permitir o modo “Instant Run”. Ao invés de fazer o build do app inteiro, a IDE faz o hot swap de código e recursos no app rodando no dispositivo ou emulador.

Código Java

• Geralmente há pelo menos uma Activity, no diretório relativo ao package name • Há código gerado em vários outros diretórios • Como, por exemplo, a classe R.java.

Deﬁnindo Recursos

• Diretório res/ • Entidades que não são código-fonte • Muitos tipos diferentes, como layouts, strings, imagens, menus, animação, etc. • Permitem que apps sejam personalizados para diferentes dispositivos e usuários • http://developer.android.com/guide/topics/resources/ index.html

Exemplos

• res/drawable para imagens (PNG, JPEG…)

• res/layout para especiﬁcações de layout em XML

• res/menu para especiﬁcações de menu em XML

• res/raw para arquivos em geral

• res/values para strings, dimensões…

• res/xml para arquivos XML de propósito geral.

Instruções de build

• build.gradle

• Um ou mais arquivos utilizados • Em projetos Eclipse, estava espalhado em arquivos como project.properties e .classpath.

Gradle

• Build system, build automation (make, ant, maven) • Essas ferramentas permitem especiﬁcar como utilizar compiladores, linkers, packagers, etc. • DSL implementada em Groovy (embedded) • similar a uma deﬁnição XML.

build.gradle

• Dois arquivos normalmente

•Project-level

• controla conﬁguração para todos os módulos

• conﬁguração do processo de build

• dependências que precisam ser satisfeitas

• Module-level

• Dois arquivos normalmente

• Project-level

• Module-level

• dependências especíﬁcas de código

• informações de conﬁguração especíﬁcas de Android - diretamente relacionado ao manifest

AndroidManifest.xml

• Base de qualquer aplicação • Geralmente em src/main • O que a sua aplicação tem, activities, services, etc

• Como estas peças se ligam com o sistema • que activity deve aparecer no main menu (launcher)

Elementos em comum AndroidManifest.xml e build.gradle

• Package name / Application ID

• minSdkVersion e targetSdkVersion

• Version Code e Version Name

build.gradle

• compileSdkVersion

• API level a ser utilizada para compilação

• buildToolsVersion

• versão da Android SDK build tools

AndroidManifest.xml

• Nome da aplicação

• Componentes

• Activities, Services

• Intent Filters

• Entre outros: permissões, API mínima, etc.

Strings

• Vários tipos: String, String Array, Plurals

• Armazenadas em res/values/\*.xml, ex.:

<string name=“hello”>Hello World!</string>

• Pode incluir formatação e estilos

• Podem ser acessados por outros recursos (@string/string\_name) e em código Java (R.string.string\_name)

Layouts

• Especiﬁcam interface com o usuário

•Arquivos XML, mas ferramentas permitem editar visualmente

• Armazenados em res/layout/\*.xml

• Também podem ser acessados tanto por outros recursos como em código Java (notação similar à de strings).

Ao iniciar Activity, devemos carregar o layout correspondente no código, utilizando setContentView:

Public void onCreate(Bundle saveInstanceState){

super.onCreate(savedInstanceState);

setContentView(R.layout.main\_layout);

}

Atributos

• Objetos View e ViewGroup suportam diversos atributos XML • Alguns atributos são especíﬁcos a um tipo de objeto, como textSize para TextView • Alguns são comuns a todos os objetos, como id • Outros atributos são considerados parâmetros de layout, como width e height.

ID • Identiﬁcador único, referenciado no código como um inteiro, mas geralmente assinalado como string no XML • Sintaxe: android:id=“@+id/my\_button” • Criamos uma instância do objeto View ao criar nossa Activity:

• Button b = (Button) findViewById(R.id.my\_button); • Importante deﬁnir IDs quando criando layouts relativos.

LinearLayout

• Arranjam views em linhas, horizontais ou verticais • Barra de rolagem é criada caso tamanho da janela exceda tamanho da tela • layout\_weight: ‘importância’ da view em termos de espaço ocupado. RelativeLayout

• Exibe ﬁlhas em posições relativas • A posição pode ser relativa a elementos irmãos ou ao elemento pai • layout\_alignParentTop • layout\_centerVertical • layout\_below.

Button: A ação ocorrida no clique (onClick) pode ser deﬁnida via XML, por meio de atributos • Ou ainda por meio de listeners, como já visto

ToggleButton e Switch:

ToggleButton toggle = (ToggleButton) findViewByid(R.id.togglebutton);

Toggle.setOnCheckedChangedListener(new CompoundButton.OnCheckedChangeListener(){

Public void onCheckedChanged(CompoundButton buttonView, boolean is Checked){

if(isChecked){

//The toggle is enabled

}else{

//The toggle is disabled

}

} }};

RadioGroup • ViewGroup contendo um conjunto de radio buttons (ou checkboxes) • Apenas um botão pode ser selecionado a cada momento • Se não é necessário exibir todas as opções, pode-se usar um Spinner • Responde a eventos assim como os demais elementos já vistos

ATTENDING?

* Yes
* Maybe } Isso uma do lado da outra(horizontal)
* No

Pickers

• Permitem que usuário escolha um horário ou data especíﬁca • Permitem seleção independente • Ajuda a garantir que será utilizada uma data ou hora válida e formatada corretamente.

Outros

• Ratingbar • Ideia de associar estrelas para avaliação • WebView • Container para exibir páginas web • MapView • mostra um mapa do Google Maps ‘embutido’ • Deﬁnir localizações, posição, nível de zoom, etc.

Gerenciando Eventos • Existem várias maneiras de interceptar eventos gerados pela interação com o usuário • Dentro das várias classes View que usamos para montar o layout, existem diversos métodos de callback • Podemos utilizar estas interfaces de ‘event listeners’ para capturar interação com o usuário • Em alguns casos vamos querer estender a classe View, para adicionar algum comportamento mais rebuscado para o seu botão.

Gerenciando Eventos

• Diversas interfaces de listeners deﬁnidas pela classe View • View.onClickListener.onClick() • View.onLongClickListener.onLongClick() • View.onFocusChangeListener.onFocusChange() • View.onKeyListener.onKey() • View.onTouchListener.onTouch()

Intent é algo similar, representa uma ação mais o contexto • existem bem mais verbos… • assim como um navegador processa requisições HTTP, Android sabe como lidar com Intents…

Intents

• Basicamente, uma mensagem passada entre componentes • Estrutura de dados que permite especiﬁcar uma operação a ser executada, ou um evento ocorrido • Permite late binding entre código de diferentes aplicações Android.

Operações

• Podemos visualizar Intents como uma linguagem ﬂexível, que permite especiﬁcar operações desejadas • Um componente que deseja realizar uma operação por meio de outro componente tem apenas de expressar seu intent. • Qualquer outro componente que exista e aﬁrme que pode realizar esta operação por meio de intent-ﬁlters, é invocado pela plataforma. • Os componentes não precisam estar cientes da existência uns dos outros e ainda assim podem trabalhar juntos para fornecer o resultado desejado para o usuário.

Possibilidades

• Misturar e combinar (plug and play) componentes em tempo de execução • Substituir aplicativos padrão Android com aplicações personalizadas • Reuso em nível de componentes dentro da própria aplicação e entre aplicações.

Principais Casos de Uso

• Iniciar uma atividade • startActivity() | startActivityForResult() • Iniciar um serviço

• startService()… • Transmitir uma mensagem (broadcast)

• sendBroadcast()…

Tipos de Intent • Explícitos • especiﬁcam o componente a ser iniciado pelo nome (fully-qualiﬁed) • tipicamente utilizado no contexto do próprio app • Implícitos • não deﬁne um componente especíﬁco, apenas declara uma ação a ser realizada • outros aplicativos podem realizar esta ação.

Como iniciar activities utilizando intents?

Iniciando activities

startActivity(Intent intent, …)

startActivityForResult(Intent intent, …)

Intent Explícito

//fileUrl é uma string como "http://www.example.com/image.png"

Intent downloadIntent = new Intent(this,   DownloadService.class);

downloadIntent.setData(Uri.parse(fileUrl));

startService(downloadIntent);

Intens Implícitos

Android encontra o componente apropriado por meio dos intent ﬁlters • Expressões declaradas no manifest, que especiﬁcam o(s) tipo(s) de intent(s) que os componentes do app recebem

Definindo um Intent:

• Component • Action • Data • Type • Category • Extras • Flags

• O nome do componente que deve receber o Intent • Distingue um intent explícito de um implícito • Se precisamos iniciar um componente especíﬁco, devemos especiﬁcar o nome do mesmo • ex.: com.example.ExampleActivity • Ao iniciar serviços, devemos sempre especiﬁcar o nome do componente, por qual razão?

Intent i = new Intent(Context packageContext,  Class<?> cls);

ou…

Intent i = new Intent();

setComponent(…), setClass(…), setClassName(…)

Action:

• Uma String que especiﬁca a ação genérica a ser executada • No caso de broadcasts, é a ação que aconteceu e está sendo relatada • Este campo determina como o restante do Intent é estruturado, particularmente quais os dados e campos extra que estarão contidos • Podemos especiﬁcar nossas ações, mas em geral devemos usar as constantes deﬁnidas na classe Intent.

Exemplo:

ACTION\_DIAL - discar um número

ACTION\_EDIT - mostrar dados para editar

ACTION\_VIEW - mostrar dados ao usuário

ACTION\_SYNC - sincronizar dados c/ um servidor

ACTION\_MAIN - disparar uma atividade como atividade inicial de uma aplicação

ACTION\_SEND - “share” intente.

Deﬁnindo:

Intent i = new Intent(Intent.ACTION\_DIAL);

ou…

Intent i = new Intent();

i.setAction(Intent.ACTION\_DIAL);

Data:

• Dados associados com o Intent • Formatados como Uniform Resource Identiﬁer (Uri) • O tipo de dado fornecido geralmente é ditado pela ação do Intent • Por exemplo, se a ação é ACTION\_EDIT, o campo Data deve incluir a URI do documento a ser editado • Ao criar um Intent, também é importante deﬁnir o tipo de dados (MIME type), além da UR.

Dados para mapa:

-Uri.parse(“geo:0,0?q=Av.+Jornalista+Anibal +Fernandes,+s/n+Recife+PE”);

Telefone a ser discado:

- Uri.parse(“tel:+558198761234”)

Definindo:

Intent i = new Intent(Intent.ACTION\_DIAL,

Uri.parse(“tel:+5581…”));

ou…

Intent i = new Intent(Intent.ACTION\_DIAL);

i.setData(Uri.parse(“tel:+5581…”));

MIME Type:

• Ao criar um Intent, também é importante deﬁnir o tipo de dados (MIME type), além da URI • Especiﬁcar o MIME type ajuda Android a encontrar o melhor componente para receber o intent • Em alguns casos, o MIME type pode ser inferido a partir da URI • Ex.: no caso de content:, que indica dados localizados no dispositivo, que tornam visível o MIME type.

Exemplos:

image/\*, image/png, image/jpeg, image/jpg

text/html, text/plain.

Definindo:

Intent i = new Intent();  i.setType(String type);

ou…

Intent i = new Intent();  i.setDataAndType(Uri data, String type);

Category:

• Informação adicional sobre o tipo de componente que deve receber o intent • Um intent pode ter várias categorias.

CATEGORY\_BROWSABLE  invocada por um navegador para mostrar dados referenciados por um link (imagem, email…)

CATEGORY\_LAUNCHER  atividade inicial de uma tarefa, listada como ‘app launcher’

CATEGORY\_TAB|TEST|UNIT\_TEST|PREFERENCE|…

Definindo:

Intent i = new Intent();

i.addCategory(Intent.CATEGORY\_XYZ);

Ao ler estas propriedades, o sistema Android é capaz de deﬁnir qual componente deve ser iniciado. Um intent pode carregar informação adicional, que não afeta a resolução do componente que será iniciado.

Extras:

• Informação adicional associada com o Intent • Um mapa de pares chave-valor • Algumas ações utilizam dados extras • Existem várias constantes já deﬁnidas para chaves, padronizando vários tipos de dados • Podemos deﬁnir as nossas próprias.

Ex- EXTRA\_EMAIL:

Intent i = new Intent (Intent.ACTION\_SEND);

i.putExtra(android.content.Intent.EXTRA\_EMAIL, new String[]{ “lmt@cin.ufpe.br”,”posgrad@cin.ufpe.br” } };

putExtra():

putExtra(String name, double[] value) putExtra(String name, int value) putExtra(String name, CharSequence value) putExtra(String name, char value) putExtra(String name, Bundle value) putExtra(String name, Parcelable[] value) putExtra(String name, Serializable value) putExtra(String name, int[] value) putExtra(String name, float value) putExtra(String name, byte[] value) putExtra(String name, long[] value) putExtra(String name, Parcelable value)

putExtra(String name, float[] value) putExtra(String name, long value) putExtra(String name, String[] value) putExtra(String name, boolean value) putExtra(String name, boolean[] value) putExtra(String name, short value) putExtra(String name, double value) putExtra(String name, short[] value) putExtra(String name, String value) putExtra(String name, byte value) putExtra(String name, char[] value) putExtra(String name, CharSequence[] value).

Flags:

Funcionam como metadados para o Intent • Podem instruir o sistema Android sobre como iniciar uma atividade… • a qual task a atividade vai pertencer • …ou como tratá-la após ser iniciada • se vai aparecer na lista de atividades recentes.

Exs: FLAG\_ACTIVITY\_NO\_HISTORY: não coloca atividade na pilha.

FLAG\_DEBUG\_LOG\_RESOLUTION: imprime informação extra de log quando o intent está sendo processado.

Exs:

FLAG\_GRANT\_READ\_URI\_PERMISSION

FLAG\_GRANT\_WRITE\_URI\_PERMISSION

FLAG\_DEBUG\_LOG\_RESOLUTION

FLAG\_FROM\_BACKGROUND

FLAG\_ACTIVITY\_BROUGHT\_TO\_FRONT

FLAG\_ACTIVITY\_CLEAR\_TASK

FLAG\_ACTIVITY\_CLEAR\_TOP

FLAG\_ACTIVITY\_CLEAR\_WHEN\_TASK\_RESET

FLAG\_ACTIVITY\_EXCLUDE\_FROM\_RECENTS

FLAG\_ACTIVITY\_FORWARD\_RESULT

FLAG\_ACTIVITY\_LAUNCHED\_FROM\_HISTORY

FLAG\_ACTIVITY\_MULTIPLE\_TASK

FLAG\_ACTIVITY\_NEW\_TASK

FLAG\_ACTIVITY\_NO\_ANIMATION

FLAG\_ACTIVITY\_NO\_HISTORY

FLAG\_ACTIVITY\_NO\_USER\_ACTION

FLAG\_ACTIVITY\_PREVIOUS\_IS\_TOP

FLAG\_ACTIVITY\_RESET\_TASK\_IF\_NEEDED

FLAG\_ACTIVITY\_REORDER\_TO\_FRONT

FLAG\_ACTIVITY\_SINGLE\_TOP

FLAG\_ACTIVITY\_TASK\_ON\_HOME

FLAG\_RECEIVER\_REGISTERED\_ONLY

Definindo:

Intent i = new Intent(Intent.ACTION\_SEND);

i.setFlags(Intent.FLAG\_ACTIVITY\_NO\_HISTORY);

Intents Implícitos:

• Especiﬁcam ações que podem invocar qualquer app no dispositivo que seja capaz de executar a ação desejada • Útil quando o aplicativo não é capaz de executar uma ação, mas outros aplicativos podem e você gostaria de oferecer a possibilidade ao usuário • Por exemplo, podemos desejar compartilhar conteúdo com outras pessoas, por meio de uma ação ACTION\_SEND • Ao chamar startActivity() com este intent, o usuário pode escolher o aplicativo que utilizará para compartilhar o conteúdo.

Exemplo de Intent Implícito:

//txt = “...”

Intent i = new Intent();

i.setAction(Intent.ACTION\_SEND);

i.putExtra(Intent.EXTRA\_TEXT, txt);

i.setType(HTTP.PLAIN\_TEXT\_TYPE);

//Checa que alguma activity pode receber o intente

If(i.resolveActivity(getPackageManager())!=null)

{

startActivity(sendIntent);

}

Intent Resolution:

• Quando não especiﬁcamos um componente explicitamente, Android tenta descobrir qual Activity casa com o Intent • O processo é chamado de Intent Resolution • Intent descrevendo operação • IntentFilters descrevendo que operações uma Activity pode receber.

Intent Filter:

• Para declarar que intents implícitos seu aplicativo pode receber, declaramos um ou mais elementos do tipo <intent-filter> no manifesto • Especiﬁca os tipos baseados em action, data, e category • Intents só são recebidos pela sua aplicação se passarem por um dos intent ﬁlters • Um ﬁlter para cada tarefa que o componente da aplicação pode fazer.

• Ex.: Visualizar e Editar uma imagem.

<activity ...>

<intent-filter ...>

...

<action … />

<data … />

<category … />

...

</intent-filter>

...

</activity

Ex:

<activity android:name=”ShareActivity”>

<intent-filter>

<action

android:name=”android.intent.action.SEND”/>

<category

Android:name=”android.intent.category.DEFAULT”/>

<data android:mineType=”text/plain”/>

</intente-filter>

</activity>

Intent.ACTION\_DIAL

<activity ...>

<intent-filter...>

...

<action android:name= ”android.intent.action.DIAL” />

...

</intent-filter>

...

</activity>

Dados em um Intent Filter:

<intent-filter ...>

...

<data

android:mimeType="string"

android:scheme="string"

android:host="string"

android:port="string"

android:path="string"

android:pathPattern="string"

android:pathPrefix="string"

/>

...

</intent-filter>

Recebendo intens associados com mapas

<intent-filter ...>

...

<data android:scheme="geo"/>

</intent-filter>

<activity android:name=”MainActivity”>

<!—This activity is the main entry, should appear em app launcher-->

<intente-filter>

<action android:name=”android.intent.action.MAIN” />

<category android:name=”android.intent.category.LAUNCHER”/>

</intente-filter>

</activity>

<activity android:name=”ShareActivity”>

<!—This activity handles “SEND” actions with text data-->

<intente-filter>

<intente-filter>

<action android=”android.intent.action.SEND”/>

<category android:name=”android.intent.category.DEFAULT”/>

<data android:mimeType=”text/plain”/>

</intente-filter>

<!—This activity also handles “SEND” and “SEND\_MULTIPLE” with media data-->

<intent-filter>

<action android:name=”android.intent.actio.SEND/n”>

<action android:name=”android.intent.action.SEND\_MULTIPLE”/>

<category android:name=”android.intent.category.DEFAUL”

<data android:mimeType = application/vnd.google.panorama360+jpn”/>

<data android:mimeType = “image/\*”/>

<data android:mimeType=”video/\*”/>

</intente-filter>

</activity>

Processo de resolução:

• Baseado em três aspectos • Action • Category • Data (URI e MIME type

Action

• Um intent ﬁlter pode declarar zero ou mais elementos <action> • Para passar por um ﬁltro, a ação especiﬁcada no Intent deve casar com uma das ações listadas no ﬁlter • Se um intent ﬁlter não listar nenhuma ação, todos os intents vão falhar no teste.

Category

• Um intent ﬁlter pode declarar zero ou mais elementos <category> • Para passar, todas as categorias declaradas no objeto Intent devem casar com categorias declaradas no ﬁltro • O inverso não é necessário.

Atenção

• Para receber intents implícitos, devemos incluir a categoria CATEGORY\_DEFAULT no manifesto • Android automaticamente aplica esta categoria aos intents implícitos passados para os métodos que chamam atividades.

Data:

• Um intent ﬁlter pode declarar zero ou mais elementos <data> • Cada elemento pode especiﬁcar uma estrutura URI e um tipo de dados (MIME) • Existem atributos separados para cada parte da URI <scheme>://<host>:<port>/<path> content://com.example.project:200/folder/subfolder/etc

Regras

• Um intent que não contém URI nem tipo MIME passa no teste apenas se o ﬁltro não especiﬁcar quaisquer URIs ou tipos MIME. • Um intent que contém URI, mas nenhum MIME (explícito ou inferido a partir da URI) passa no teste apenas se o seu URI corresponde ao formato URI do ﬁltro e o ﬁltro também não especiﬁca um tipo de MIME.

Regras • Um intent que contém um tipo MIME, mas não um URI, passa no teste apenas se o ﬁltro lista o mesmo tipo MIME e não especiﬁca um formato de URI. • Um intent que contém um URI e um tipo MIME (explícito ou inferido a partir da URI) passa a parte “MIME” do teste somente se esse tipo corresponde a um tipo listado no ﬁltro. Só passa a “parte URI” do teste se a sua URI corresponde a um URI no ﬁltro ou se tem uma URI content: ou file: e o ﬁltro não especiﬁca um URI.

Intent Matching

• O Home app popula o launcher procurando por todas as activities com intent ﬁlters com action MAIN + category LAUNCHER • O PackageManager tem um conjunto de métodos do tipo query…() que retornam componentes que aceitam um intent particular.

If you want to share someting in any app you want or open a url via every action, just use this method:

private void shareOrViewUrlViaThisApp(String appPackgeName, String url){

boolean found = false;

Intent intente = new Intent(Intent.ACTION\_VIEW);

List<ResolveInfo> resInfo = getPackageManager().queryIntentActivities(intente, 0)

If(!resInfo.isEmpty()){

for (ResolveInfo info: resInfo){

if(info.activityInfo.packgeName.toLowerCase().contains(appPackageName)

info.activityInfo.name.toLowercase().contains(appPackgeName) )

intente.setPackage(info.activityInfo.packageName);

found = true;

break;

}

}

if(!found)

return;

startActivity(Intent.createChooser(intent, “Select”));

}

}

And simply call:

shareorViewUrlViaThisApp(<your package name>,<your url>);

This anserw was inspired from “this”.

There is a way to directly open the intente you want. You can get the listo f intents and open only one.

See this code:

private void initShareIntent(String type){

boolean found = false;

Intent share = new Intent(android.content.Intent.ACTION\_SEND);

share.setType(“image/jpeg”);

//gets the List of intents that can be loader.

List<ResolveInfo> resinfo = getPackageManager().queryIntentActivities(share,0);

If(!resInfo.isEmpty()){

for(ResolveInfo info : resInfo){

if(info.activityInfo.packageName.toLowerCase().contains(type) || info.activityInfo.name.toLowerCase().contains(type)){

share.putExtra(Intent.EXTRA\_SUBJECT, “subject”);

share.putExtra(Intent.EXTRA\_TEXT, “your text”);

share.SetPackage(info.activityInfo.packageName);

found = true;

break;

}

}

if(!found)

return;

startActivity(Intent.createChooser(share, “Select”));

}

}

If you wanna open twitter, do that:

initShareIntent(“twi”);

if facebook:

initShareIntent(“face”);

if mail:

initShareIntent(“mail”);//or gmail

Bastidores • Ao iniciar um app, Android faz um fork de um processo chamado de zygote • O processo vai conter: • Cópia da VM (Dalvik ou ART) • Cópia das classes do Android Framework • Cópia das classes do app (via APK) • Objetos criados a partir das classes de framework, como as instâncias das subclasses de Activit.

Suponha que seu aplicativo tem apenas uma activity. O usuário vai abrir o app a partir do launcher… Uma vez que esteja rodando, o usuário aperta o botão BACK. O que acontece? Android tenta manter seu processo vivo, ao menos por um tempo, por qual razão? E se o usually aperta o botão HOME… O que acontece? A diferença é o que acontece com a Activity…

BACK vs. HOME • Ao pressionar BACK, a activity em primeiro plano é destruída. • A instância não será mais usada e ﬁca marcada para ser garbage collected • Ao pressionar HOME, a activity não é destruída imediatamente, mas permanece na memória • Se o usuário retornar ao app, o sistema usa a instância existente ao invés de criar novo objeto.

Termination

• Processos não vivem eternamente, pois ocupam espaço de RAM • Eventualmente, Android tem de se livrar de processos para liberar memória • O quanto seu processo vai durar depende de vários fatores, como por exemplo? • O que o dispositivo está fazendo em primeiro plano (aplicativos visíveis) e em segundo plano • Quanto de memória o dispositivo tem • O que ainda roda no processo. No cenário anterior, a diferença entre HOME e BACK é o potencial tempo que o processo ainda viverá… Android tende a manter processos por mais tempo se estes tem componentes ativos. Primeiro plano

• Android não elimina processos arbitrariamente • Processos de primeiro plano são os últimos a serem eliminados (veremos mais detalhes) • Se isso acontece, a coisa está feia… :)

Heap Size • Processos usam RAM, e uma porção signiﬁcativa será usada pelos objetos criados (heap) • Os tamanhos de heap em Android não são grandes, e não controlamos usando -Xmx • É imperativo ter isto em mente enquanto programamos… • Para aplicativos simples como os que estamos vendo até agora, não há o que se preocupar.

Activity

• Interface para interação com o usuário • Cada activity deve dar suporte a apenas uma única tarefa do usuário • Aplicações geralmente consistem de várias activities, cada uma com um propósito • Isso signiﬁca que precisamos ter meios para iniciar activities.

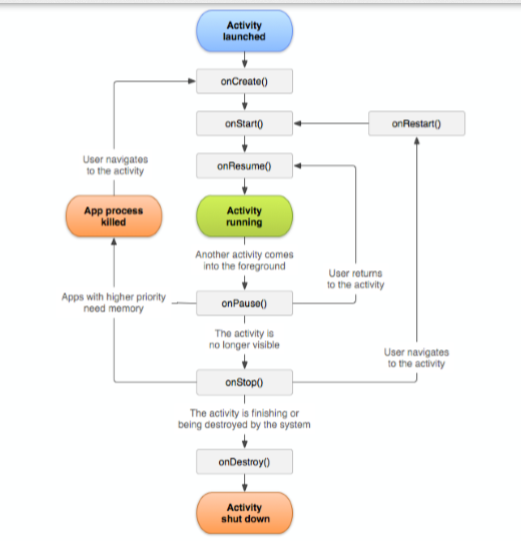
Estados

• Active: activity foi iniciada, está em memória e em primeiro plano • Paused: activity foi iniciada, está em memória, (parcialmente) visível, mas outra activity ocupa o primeiro plano, não é possível interagir • Stopped: activity foi iniciada, está em memória, mas não é mais visível, por conta de activities que foram iniciadas • Dead ou non-existent: activity nunca foi iniciada ou foi destruída, ou seja não está mais em memória.

Ciclo de Vida

• Activities são criadas, suspensas, resumidas e destruídas conforme o necessário, durante execução da aplicação • Algumas ações dependem do usuário • Outras ﬁcam a cargo do sistema. Métodos para mudança de estados

• Android fornece métodos para anunciar a mudança de estado em Activities • De acordo com a necessidade da aplicação.



onCreate():

• Chamado ao criar Activity • Deﬁne estado inicial • pode deﬁnir layout (setContentView…) • inicializa referências para elementos de UI • conﬁgura interface conforme necessário.

onDestroy() • Activity está prestes a ser destruída, por conta do botão BACK, ou chamada a finish() • Liberar recursos • Pode não ser chamado se • ocorrer um crash com exceção não tratada • usuário usa force-quit via Settings • Android matar sua aplicação para liberar memória.

onStart()

• Activity está prestes a se tornar visível • por estar sendo iniciada pela primeira vez • voltando ao primeiro plano após um tempo • Iniciar comportamento que deve estar pronto quando aplicação se tornar visível • Carregar dados da aplicação (persistência).

onRestart()

• Chamado se Activity foi pausada e está prestes a ser iniciada novamente • Útil para processar ações que são necessárias apenas quando a aplicação foi pausada e reiniciada.

onStop()

• Chamado assim que Activity não é mais visível para o usuário • Pode ser reiniciada mais tarde • Cache state • Pode não ser chamado pelas mesmas razões que onDestroy().

onResume()

• Activity está visível e prestes a iniciar interação com o usuário em primeiro plano • Iniciar comportamento visível, refresh da UI.

onPause()

• Prestes a perder o foco, como por exemplo, ao iniciar outra activity, ou ao aparecer um dialog • Interromper comportamento visível • Salvar estado.

Trabalhe com os pares

• Se inicializou algo com X, limpe com Y

• x = onCreate(); y = onDestroy();

• x = onStart(); y = onStop();

• x = onResume(); y = onPause();

Quando Activities morrem?

• Ao pressionar o botão BACK • Chamando finish() a partir da própria activity • Evite deﬁnir botões de “sair”/“exit”/“quit” • use as opções de Android, como botão BACK.

• Se nenhuma activity de um app está em primeiro plano, o processo é candidato para ser eliminado • pode ser que não execute onDestroy() • Se o dispositivo passa por mudança de conﬁguração, como por exemplo, orientação • Se ocorrem exceções não tratadas.

Iniciando Activities:

Dois cenários

• Sabemos exatamente que activity desejamos iniciar

• em geral, outra activity do mesmo app

• Temos uma referência para algo, como por exemplo, um link para página da web

• queremos permitir que o usuário visualize, mas não sabemos exatamente quais opções temos disponíveis…

Explicit Intents

• Um Intent encapsula uma requisição, feita ao sistema, para algum componente fazer algo • Se sabemos exatamente o que queremos, usamos intents explícitos, nomeando o componente que desejamos carregar

new Intent(this, ListActivity.class);

Implicit Intents

• Intents explícitos funcionam bem quando sabemos exatamente que componente utilizar • No entanto, podemos iniciar activities a partir do SO ou de aplicativos de terceiros • Nestes casos não temos um objeto Class que representa a Activity… • Usamos a estrutura de intents implícitos, que se parecem com requisições Web (URIs).

Exemplos • Imagine que você obteve latitude e longitude a partir de uma mensagem… • Podemos desejar exibir um mapa • Uma opção é embutir Google Maps… • A outra é simplesmente terceirizar para o sistema • “Ei, Android, abre aí uma activity que consegue exibir mapas para estas coordenadas…”

Exemplo Explicit vs. Implicit:

Uri.parse()

• No exemplo, vimos como usar este método para fazer o parsing de uma URL http(s) • Existem outros schemes que podem ser usados • file:// - via Uri.fromFile() • content:// - content providers.

startActivity() vs. startActivityForResult()

Navegação entre Activities

• Tasks • Task Backstack • Suspendendo e resumindo Activities

Tasks

• Um task é um conjunto de Activities relacionadas • Estas activities não precisam ser parte da mesma aplicação • Geralmente iniciados a partir de um clique no ícone do app.

Task Backstack

• Quando iniciamos uma Activity, esta reside no topo da pilha • Quando a Activity é destruída, ela é removida (pop) da pilha.

Processos

Iniciando componentes

• Quando um componente inicia e a aplicação não tem nenhum componente sendo executado, o sistema inicia um novo processo • Por padrão, todos os componentes de uma mesma aplicação rodam no mesmo processo e thread (chamada de ‘main thread’).

Processos em Android • As entradas do manifest de cada um dos tipos de componentes (<activity>, <service>, <receiver>, <provider>) possuem um atributo android:process, que permite especiﬁcar o processo que o componente irá rodar • É possível fazer com que cada componente rode em um processo separado, ou fazer com que apenas alguns compartilhem o processo • Também é possível fazer com que componentes de diferentes aplicações compartilhem o mesmo processo.

Matando Processos

• Android pode decidir matar um processo, quando há pouca memória e há necessidade de uso da memória por outros processos • Ao matar um processo, todos os componentes da aplicação rodando no processo são destruídos • Para decidir que processos serão desligados, o sistema leva em consideração a importância relativa do processo para o usuário.

Ciclo de vida • Android tenta manter um processo rodando pelo maior tempo possível, mas eventualmente precisa remover processos antigos para liberar memória • Para determinar quais processos devem ser mantidos e quais devem ser eliminados, há uma ‘hierarquia de importância’ • Processos menos importantes são eliminados primeiro, e assim sucessivamente, até atingir o necessário.

Níveis de Importância

• Foreground • Visible • Service • Cached

Foreground • Um processo que está associado com o que o usuário está fazendo atualmente. Um processo é considerado foreground se alguma das condições abaixo for satisfeita: • Hospeda uma Activity que o usuário está interagindo no momento • Hospeda um Service que está executando um dos métodos de callback do ciclo de vida (onCreate(), onStart(), ou onDestroy()) • Hospeda um BroadcastReceiver que está executando o método onReceive().• Geralmente, apenas alguns processos estão neste nível em um dado momento • Só são eliminados como última manobra - memória está tão baixa que não podem continuar a ser executados • Geralmente, neste ponto, o dispositivo está em estado de paginação de memória, e precisa eliminar alguns processos para manter a interface com o usuário ‘responsiva’.

Visible • Processo que faz algo que o usuário esteja ciente. • Hospeda Activity que não está em primeiro plano, mas ainda está visível (onPause() foi chamado) • Hospeda Service que está rodando como foreground • Hospeda Service usado para uma feature que o usuário está ciente, como um live wallpaper… etc • Um processo visível é considerado importante e não é eliminado, a não ser que isto seja necessário para manter os processos de primeiro plano rodando.

Service • Um processo que roda um Service que foi iniciado com startService(), mas não se encaixa nas categorias anteriores • Embora os processos de serviços não estejam diretamente ligados com algo que o usuário vê, geralmente estão fazendo algo que o usuário se importa, como um download • Quanto mais tempo rodando, a importância é reduzida.

Cached

• Estes processos não tem impacto direto na experiência do usuário e o sistema pode eliminálos a qualquer momento para liberar memória • Em um sistema funcionando normalmente, estes são os afetados pelo gerenciador de memória • Tendência é manter múltiplos processos em cache e regularmente eliminar os mais antigos.

Um processo que hospeda uma Activity que não é visível ao usuário (método onStop() foi chamado) • Se uma Activity implementa o ciclo de vida corretamente, e salva o estado atual, eliminar o processo não tem efeito visível para o usuário. • São mantidos em uma (pseudo)lista de processos recentes (LRU), para garantir que o processo associado com a Activity mais recentemente vista pelo usuário seja o último a ser eliminado.

Hierarquia de Processos • Android tenta classiﬁcar processos com o maior nível possível, baseado na importância dos componentes atualmente ativos no processo. Por exemplo, se um processo hospeda um Service e uma Activity visível, o processo é classiﬁcado como Visible, ao invés de Service. • Além disso, o nível de um processo pode ser aumentado pelo fato de outros processos serem dependentes do mesmo - um processo que está servindo a outro processo não pode ser deﬁnido com nível inferior ao do processo servido.

• Processos que rodam Services são superiores a processos com Activities de background • Portanto, qualquer operação que tome mais tempo deve iniciar um Service, principalmente se for durar mais que a Activity • Usar Services garante que pelo menos há uma certa prioridade do processo, independente do que acontece com a Activity • Esta também é a razão pela qual BroadcastReceivers devem iniciar Services ao invés de realizar operações que demandam muito tempo.

Vamos criar um app mais interessante…

Leitor RSS:

• Capturar o conteúdo de um feed RSS público • Extrair os dados importantes (título, link, etc) • Apresentar uma lista de itens RSS • Ao clicar em um item, devemos abrir o navegador no link associado • Deve ser possível trocar o feed durante a execução do aplicativo.

RSS • Formato de dados que permite que produtores de conteúdo publiquem informações, áudio/vídeo, etc. • A publicação se dá geralmente por meio de um feed, que é regularmente atualizado • É possível ‘assinar’ feeds, para receber novidades automaticamente • O formato é baseado em XML, mais detalhes em http://www.rssboard.org/rss-speciﬁcation.

Qual o primeiro passo?

Leitor RSS

• Criar um projeto com uma blank activity e colocar um TextView só para testarmos inicialmente • outra opção é usar fragmentos… • No método onStart(), atualizar o conteúdo com uma string qualquer em um text view • (no futuro serão utilizados os dados provenientes do feed).

• Agora vamos capturar o conteúdo de um feed RSS • Atualize o conteúdo do fragmento com um feed RSS disponível online • Sugestão: <http://rss.cnn.com/rss/edition.rss>

The following code sample shows a typical use, with an XML layout and code to modify the contents of the text view:

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:id="@+id/text\_view\_id"

android:layout\_height="wrap\_content"

android:layout\_width="wrap\_content"

android:text="@string/hello" />

</LinearLayout>

This code sample demonstrates how to modify the contents of the text view defined in the previous XML layout:

public class MainActivity extends Activity {

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

final TextView helloTextView = (TextView) findViewById(R.id.text\_view\_id);

helloTextView.setText(R.string.user\_greeting);

}

}

Iniciando conexão HTTP:

URL url = new URL(“http://www.android.com/”);

HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();

try{

InputStream in = new BufferedInputStream(urlConnection.getInputStream());

readStream(in);

finally{

urlConnection.disconnect();

}

}

...............................................................................................................

getRssFeed(..):

private String getRssFeed(String feed) throws IOException{

InputStream in = null;

String rssFeed = null;

try{

URL url = new URL(feed);

HttpURLConnection conn = (HttpURLConnection) url.openConnection();

In = conn.getInputStream();

ByteArrayOutputStream out = new ByteArrayOutputStream();

byte[] buffer = new byte[1024];

for(int count; (count = in.read(buffer))!= -1;){

out.write(buffer, 0 count);

}

byte[] response = out.toByteArray();

rssFeed = new String(response, “UTF-8”);

finally{

if(in!= null){  
 in.close();

Dá erro, pois:

Operações de rede devem ser realizadas fora da thread principal!

http://android-developers.blogspot.com.br/2010/07/ multithreading-for-performance.html.

O que acontece ao chamarmos label.setText(…)?

Main Thread:

ANR

Jank

Mas o que são processos mesmo?

Processos

• Ambientes de execução auto-contidos • Gerenciam recursos de forma separada de outros processos, como • memória • arquivos abertos • conexões de rede.

O que é uma thread?

• Visão Conceitual

• É uma de muitas computações paralelas rodando em um processo do SO;

• Visão de implementação

• Um program counter e uma stack • Compartilha a heap e áreas de memória estáica com outras threads rodando em um processo.

Threads em Java

• Representadas por um objeto do tipo java.lang.Thread

• Threads devem implementar a interface Runnable

• void run().

Alguns métodos

• void start()

• inicia a thread

• void sleep (long time)

• coloca a thread ‘para dormir’ durante dado período

• void wait()

• thread aguarda até que outra thread invoque o método notify() neste objeto

• void notify()

• acorda uma thread que está esperando.

Fluxo Geral

• Instanciar um objeto do tipo Thread

• Chamar o método start()

• que por sua vez chama run()

• A thread termina quando o método run() retorna.

Running example

• Aplicação com dois botões:

• Carregar Ícone (operação “demorada”)

• carrega uma imagem a partir de um arquivo

• exibe a imagem

• Outro

• exibe um texto (toast)

Sem Threads

Com Threads (incorreto)

listener do botão dispara uma thread separada para carregar a imagem e exibi-la.

O que houve?

UI Thread

• Aplicações tem uma thread principal (UI Thread)

• Componentes da aplicação que compartilham o processo usam a mesma UI thread

• Interação com o usuário, callbacks do sistema e métodos do ciclo de vida de componentes são todos tratados na UI thread

• UI thread não é thread-safe.

Implicações

• Bloquear UI Thread prejudica uso e capacidade de resposta da aplicação • operações que levem tempo devem rodar em threads de background • Não devemos acessar UI de uma thread que não seja UI.

Então como fazer para atualizar a UI a partir de uma background thread?

Melhorando a Solução

• Precisamos realizar alguma tarefa em segundoplano, mas depois é necessário atualizar a UI • Android fornece métodos que são garantidos de rodar na UI Thread

• boolean View.post(Runnable)

• void Activity.runOnUiThread(Runnable)

• boolean View.postDelayed(Runnable, long)

Com Threads e view.post(…)

Com Threads e …runOnUi…(…)

Alternativa: AsyncTasks.

AsyncTask

• Divide o trabalho em

• Background thread

• realiza a tarefa

• indica progresso

• UI Thread

• conﬁguração inicial

• publica progresso intermediária

• usa o resultado ﬁnal

Como faz er isto?

Usando AsyncTask

• Classe Genérica

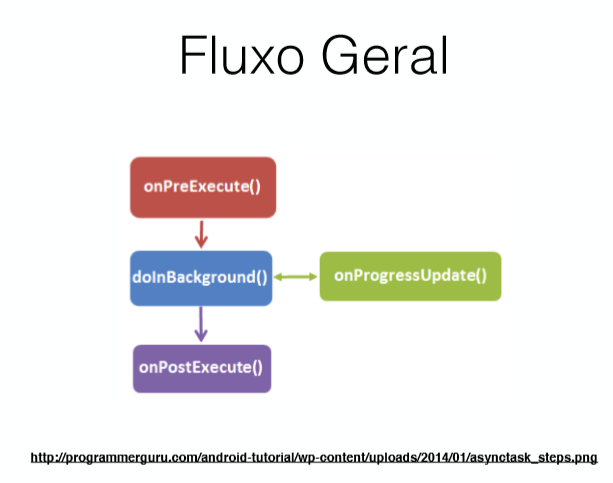
• class AsyncTask<Params,Progress,Result> {…}

• Parâmetros

• Params - tipo usado para tarefa em background

• Progress - tipo usado para indicar progresso

• Result - tipo do resultado



Fluxo Geral

• void onPreExecute()

• roda na UI Thread antes de doInBackground()

• Result doInBackground(Params… params)

• realiza a tarefa na thread separada

• pode chamar void publishProgress(Progress… values)

• void onProgressUpdate(Progress… values)

• invocado nas chamadas de publishProgress(…)

• void onPostExecute(Result result)

• roda após doInBackground()

private class DownloadFilesTask extends RL, Integer, Long>{

protect Long doInBackground (URL... urls){

int count = urls.length;

long totalSize = 0;

for(int i = 0; i < cont; i++){

totalSize += Donloader.downloadFile(urls[i]);

publishProgress((int) ((int) ((i/(float)count)\*100));

//escape early if cancel() is called

if(isCancelled()) break;

}

return totalSize;

}

protect void onProgressUpdate(Integer progress){

setProgressPercent(progress[0]);

}

Protect void onPostExecute(Long result){

showDialog(“Donwloaded ” + result + “bytes”);

}

}

}

New DownloadFilesTask().execute(url1, url2, url3);

<https://developer.android.com/reference/android/os/AsyncTask>

Carregando imagem com AsyncTask:

O que você não precisa fazer.

Estados:

• AsyncTask.Status getStatus()

• retorna o estado atual de um objeto AsyncTask

• PENDING

• ainda não foi executado

• RUNNING

• rodando no momento

• FINISHED

• indica que onPostExecute(Result) ﬁnalizou.

Cancelando:

• task.cancel(boolean)

• tenta cancelar execução da tarefa (interrupção depende do booleano passado como argumento)

• seta uma ﬂag, ao chamar isCancelled()

• após doInBackground() chama onCancelled() ao invés de onPostExecute()

• boa prática: chamar isCancelled() periodicamente dentro de doInBackground().

Services

Canivete suíço!

Utilidade: • Operações que precisam continuar mesmo que o usuário deixe Activity, como fazer downloads longos ou tocar música • Operações que precisam existir independente das activities estarem ativas ou não, como manter uma conexão de chat online • Fornecer uma API local para APIs remotas, como as fornecidas por web services • Realizar tarefas periódicas sem intervenção do usuário.

Até mesmo widgets da tela inicial geralmente envolvem services para auxiliar em tarefas longas.

Não são AsyncTasks melhorados…

• Não interagem diretamente com o usuário (sem UI) • Principais usos • Processar tarefas em segundo-plano • Dar suporte a comunicação inter-processos • remote method execution

Como services são criados e sobrevivem?

• Iniciados manualmente (via API Android) • Uma activity conecta ao Service via inter-process communication (IPC) • Vivem até que sejam explicitamente desligados ou caso o sistema precise de memória e o destrua • Cuidado com o tempo, para não detonar a bateria.

Implementando…

• Criar subclasse de Service e implementar alguns métodos

• onCreate() • onStartCommand() • onBind() • onDestroy()

Métodos do ciclo de vida

• onCreate()

• Chamado quando o Service é criado pela primeira vez, para procedimentos de conﬁguração • Se o Service já está rodando, este método não é chamado

• onDestroy()

• Chamado quando o Service não está mais sendo usado e vai ser destruído • Útil para liberar recursos como threads, listeners e receivers registrados, etc.

onStartCommand()

• Chamado sempre que algum componente requisita que o Service seja iniciado - startService().

• É de responsabilidade de quem implementa deﬁnir quando o Service deve ser interrompido (opções são os métodos stopSelf e stopService).

onBind()

• Chamado quando um componente faz binding com o Service - bindService(). • Deve ser fornecida uma interface para clientes se comunicarem com o Service, com base na interface IBinder • Este método sempre deve ser implementado. Se não for desejável permitir binding, deve retornar null.

Declarando Service

• Assim como outros componentes, AndroidManifest.xml

• O nome da classe corresponde ao nome do Service, similar a uma API pública, portanto devemos evitar mudar os nomes

• Segurança: explicit intents e android:exported

<manifest ... >

...<application ... >

<service android:name=".TesteServico" />

...

</application>

</manifest>

<http://android-developers.blogspot.com.br/2011/06/things-that-cannot-change.html>

Sintaxe

<http://developer.android.com/guide/topics/manifest/service-element.html>

Sintaxe

<service android:enabled = [“true” |”false”]

android:exported = [“true” |”false”]

android:icon = “drawable resource”

android:isolatedProcess = [“true” |”false”]

android:label = “string resource”

android:permission = “string”

android:process = “string” >

...

</service>

http://developer.android.com/guide/topics/manifest/service-element.html

Duas formas de usar e comunicar com services

• Um cliente (geralmente uma Activity) tem duas maneiras de enviar requisições ou informações a um Service

• enviar um comando e não estabelecer nenhum tipo de conexão entre o cliente e o Service

• binding com o Service: estabelece canal de comunicação bidirecional que dura o tempo que o cliente necessitar

startService()

• Recebe um Intent como parâmetro, assim como startActivity(), com o papel de: • Identiﬁcar o Service a ser iniciado • Carregar informação adicional para o Service • Usar Intent explícito! • Chamada é assíncrona, não bloqueia o cliente.

onStartCommand()

• Ao chamarmos startService() o Service é criado, se já não estiver rodando, e recebe o Intent via chamada a onStartCommand() • O método onStartCommand() roda na thread principal, portanto… • Qualquer tarefa que demore deve ser delegada a uma thread de segundo plano.

Atenção: evite ANRs!

Atenção…

• Um Service não necessariamente roda em um processo separado • Um Service não necessariamente corresponde a uma thread separada

Quando usar services vs. threads?

onStartCommand()

• O método onStartCommand() deve retornar um inteiro • START\_NOT\_STICKY - se o sistema mata o Service após este método, o Service não é criado novamente, a não ser que ainda existam intents pendentes. • START\_STICKY - se o sistema interrompe o Service após este método, o Service é criado novamente, e onStartCommand() é chamado com um intent nulo. • START\_REDELIVER\_INTENT - se o sistema interrompe o Service após este método, o Service é criado novamente, e onStartCommand() é chamado com o último intent entregue.

Encerrando Services

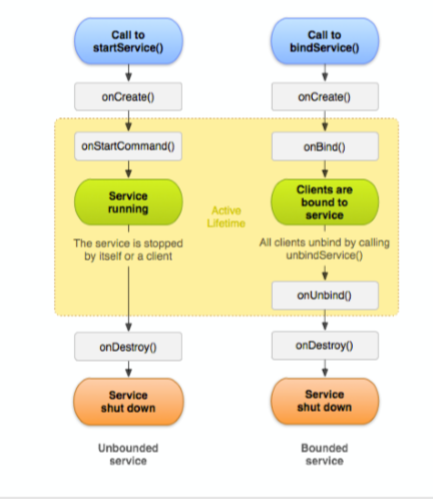
• Por padrão, startService() inicia um Service que roda até que seja encerrado explicitamente • Uma maneira de encerrar Service é por meio de stopService(), passando o mesmo Intent utilizado para iniciá-lo • Outra possibilidade é o próprio Service chamar stopSelf() ao encerrar o trabalho realizado.

Comunicando-se a partir de um Service

• Fazer Broadcast de intents • Pending intents • Event Buses • Messengers • Notiﬁcations

Ciclo de Vida de Service

• O ciclo de vida é mais simples que o de Activity • No entanto, importante prestar atenção em como Service é criado e destruído, dado que geralmente roda em background • O ciclo varia de acordo com o tipo de Service.



public class ExempleService extends Service{

int mStartMode; //indicate how to behave if the servisse is killed

Ibinder mBinder; //interface for clientes that bind

Boolean mAllowRebind; //indicates whether onRebind shoud be used

@override

public void onCreate(){

//the servisse is being created

}

@overrride

public int onStartCommand(Intent intente, int flag, int stratId){

//the servisse is starting, due to a call to startService()

return mStartMode;

}

@override

Public IBinder(Intent intent){

//A cliente is bindind to the servisse with bindService()

Return mBinder;

}

@override

public boolean onUnbind(Intent intent){

// All clientes have unbound with unbindService()

return mAllowRebind;

}

@Override

public void onRebind(Intent intent){

// A cliente is binding to the servisse with bindService(),

// after onUnbind () has already been called

}

@Override

public void onDestroy(){

///The servisse is no longed used and is being destroyed

}

}

Mudanças de Conﬁguração

• Service por padrão é alheio a mudanças de conﬁguração, continua rodando simplesmente • Se desejar observar mudanças pode sobrescrever o método onConfigurationChanged() • Importante estar ciente apenas no caso de armazenar localmente ou fazer caching de informações de conﬁguração, como locale, por ex.

Dica: IntentService:

• Usa uma thread de background, então pode fazer operações de rede e demorar mais…

• Automaticamente se encerra ao terminar tarefa

• Uma Activity pode, basicamente, enviar um comando via startService() e ‘esquecer’

• Obviamente, existem situações onde a Activity desejará sa

ber quando terminou a tarefa.

Ao pensarmos em um download, imaginamos que armazenaremos o arquivo em uma pasta, possivelmente pública…

Particularidades…

• Se estivermos em Android 6.0+, precisaremos ter permissão antes de iniciar o Service • Um Service não pode pedir permissão (pode checar, no entanto…)

Matando Serviços

• O sistema interrompe Service apenas quando a memória está baixa e é necessário recuperar recursos

• Se o Service está ligado a uma activity que interage com o usuário, é menos provável que o Service seja interrompido

• Da mesma forma, Service declarado como de primeiro plano quase nunca é eliminado

• Um Service eliminado pelo sistema é reiniciado assim que recursos se tornam disponíveis novamente.

Bound Services

• Permite que outros componentes de sua aplicação ou de outras aplicações interajam com o Service

• Binding permite que o Service exponha uma API, na forma de um objeto “binder”, com métodos de sua escolha • Isto pode ser feito intra-aplicação ou entre aplicações, permitindo comunicação entre processos

Criando Bound Services

• Ao criar um Service que permite binding, é necessário fornecer um IBinder que deﬁne a interface que clientes podem usar para interagir:

• Estendendo a classe Binder • Usando um Messenger • Usando AIDL

Implementando Binding

• Requer esforço nos dois lados, cliente e Service • O Service precisa implementar o método onBind(), que retorna null normalmente… • O cliente precisa solicitar o binding, ao invés de (ou além de) iniciar o Service

O lado do Service

• O binding é fornecido por meio da implementação de onBind() • O Service implementa uma subclasse de Binder que representa a API exposta • Para Service local, o Binder pode ter métodos, parâmetros, tipos de retorno e exceções como desejável

• Para Service remoto, a implementação do Binder é mais restrita para dar suporte a IPC

• O método onBind() retorna uma instância do Binder

O lado do cliente

• Clientes chamam bindService(), fornecendo um Intent e implementação de ServiceConnection

• O cliente não sabe nada sobre o status do binding até que o sistema chame o método onServiceConnected() de ServiceConnection

• Este método indica que o binding foi estabelecido, e no caso de local services, retorna o IBinder retornado no método onBind(), que o cliente usa para se comunicar com o Service

BIND\_AUTO\_CREATE ﬂag

unbindService()

• Eventualmente, cliente chama unbindService() para indicar que não precisa mais se comunicar com o Service • Normalmente chamado no método onStop() de Activity • Após chamar este método, não é seguro utilizar o objeto Binder no cliente • Se não existem outros clientes ligados ao Service, Android o encerra também, liberando memória • O objeto ServiceConnection também deﬁne o método onServiceDisconnected().

Estendendo Binder

• Quando o Service é usado apenas pela aplicação local e não precisa lidar com comunicação entre processos

• Implementa a própria classe Binder, que fornece acesso direto aos métodos públicos do Service

public class LocalService extends Service{

//Binder given to clientes

private final IBinder = new LocalBinder();

//Random numer generator

private final Random mGenerator = new Randon();

/\*\*

\*Class used for the cliente Binder. Because we know this servisse always

\*run is the same process as its clientes, we don’t need to deal with IPC.

\*/

public class LocalBinder extend Binder{

LocalService getService(){

//Return this instance of localService so clientes can call public methods

return LocalService.this;

}

}

@override

Public IBinder onBind(Intent intent){

Return mBinder;

}

/\*\*method for clients\*/

Public int getRandomNumber(){

return mGenerator.nexInt(100);

}

}

public class BindingActivity extends Activity{

LocalService mService;

Boolean mBound = false;

@override

protected void onCreat(Bundle savedInstanceState){

super.onCreate(saveInstanceState);

setContentView(R.layout.main);

}

@override

protect void onStart(){

super.onStart();

//Bind to LocalService

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

bindService(intente, mConnection, Context.BIND\_AUTO\_CREATE);

}

@override

Protected void onStop(){

super.onStop();

//Unbind from the service

if (mBound){

unbindService (mConnection);

mBound = false;

}

}

//\*\* Called when a button is clicked (the button in the layout file attaches to \* this method with the android:onClick attribute \*/

public void onButtonClick(View v){

if(mBound){

//Call a method from the LocalService.

// However, if this call were something that migth hang, then this request should occur in a separate thread to avoid slowing down the activity performance.

int num = mService.getRandomNumber();

Toast.makeText(this, “number: ” + num, Toast.LEGTH\_SHORT).show();

}

}

/\*\*Defines call-backs for service biding, passed to bindService() \*/

private ServiceConnection mConnection = new ServiceConection(){

@override

Public void onServiceConnected(ComponentName className, IBinder service){

//We’ve bound to LacalService, cast the IBinder and get LocalService instance

LocalBinder binder = (LocalBinder) service;

mService = binder.getService();

mBound = true;

}

@override

Public void onServiceDisconnected(ComponentName arg0){

mBound = false;

}

}

};