Intents e filtros de intents

bookmark_border

O Intent é um objeto de mensagem que pode ser usado para solicitar uma ação de outro componente do aplicativo. Embora os intents facilitem a comunicação entre os componentes de diversas formas, há três casos fundamentais de uso:

- **Como iniciar uma atividade**

  Uma Activity representa uma única tela em um aplicativo. É possível iniciar uma nova instância de uma Activity passando um Intent para startActivity(). O Intent descreve a atividade a iniciar e carrega todos os dados necessários.

  Se você quiser receber um resultado da atividade quando ela finalizar, chame startActivityForResult(). Sua atividade recebe o resultado como um objeto Intent separado no callback de onActivityResult() da atividade. Para saber mais, consulte o guia Activities.

- **Como iniciar um serviço**

  O Service é um componente que realiza operações em segundo plano sem interface do usuário. Com o Android 5.0 (API nível 21) e posteriores, é possível iniciar um serviço JobScheduler. Para saber mais sobre JobScheduler, consulte a API-reference documentation.

  Para versões anteriores ao Android 5.0 (API nível 21), é possível iniciar um serviço usando os métodos da classe Service. É possível iniciar um serviço para realizar uma operação que acontece uma vez (como fazer o download de um arquivo) passando um Intent a startService(). O Intent descreve o serviço a iniciar e carrega todos os dados necessários.

  Se o serviço for projetado com uma interface servidor-cliente, é possível vincular ao serviço em outro componente passando um Intent a bindService(). Para mais informações, consulte o guia Serviços.

- **Como fornecer uma transmissão**

  Transmissão é uma mensagem que qualquer aplicativo pode receber. O sistema fornece diversas transmissões para eventos do sistema, como quando o sistema inicializa ou o dispositivo inicia o carregamento. Você pode fornecer uma transmissão a outros aplicativos passando um Intent ao sendBroadcast() ou ao sendOrderedBroadcast().

O restante desta página explica como os intents funcionam e como usá-los. Para informações relacionadas, consulte Interação com outros aplicativos e Compartilhamento de conteúdo.

Tipos de intents

Há dois tipos de intents:

- Os **intents explícitos** especificam qual aplicativo atenderá ao intent, fornecendo o nome do pacote do aplicativo de destino ou o nome da classe de um componente totalmente qualificado. Normalmente, usa-se um intent explícito para iniciar um componente no próprio aplicativo porque se sabe o nome de classe da atividade ou do serviço que se quer iniciar. Por exemplo, iniciar uma nova atividade em resposta a uma ação do usuário ou iniciar um serviço para fazer o download de um arquivo em segundo plano.
- Os **intents implícitos** não nomeiam nenhum componente específico, mas declaram uma ação geral a realizar, o que permite que um componente de outro aplicativo a processe. Por exemplo, se você quiser exibir ao usuário uma localização em um mapa, pode usar um intent implícito para solicitar que outro aplicativo capaz exiba uma localização especificada no mapa.

A figura 1 mostra como um intent é usado ao iniciar uma atividade. Quando o objeto Intent nomeia um componente específico da atividade de forma explícita, o sistema inicia o componente de imediato.

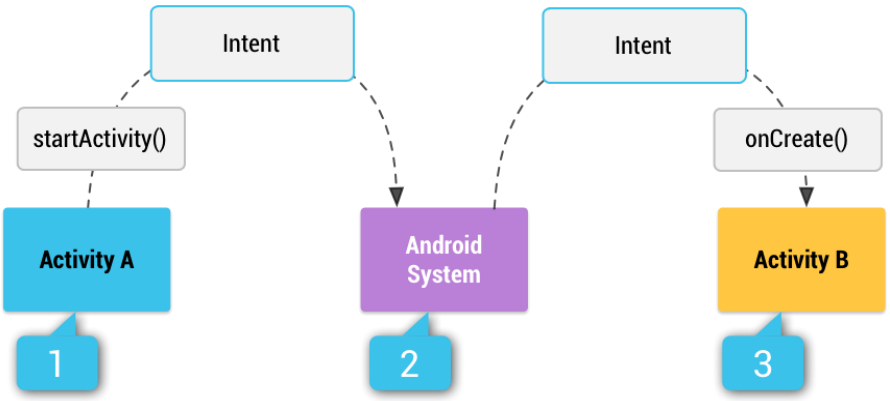

Figura 1. Ilustração de como um intent implícito é fornecido pelo sistema para iniciar outra atividade: **[1]** *A atividade A* cria um Intent com uma descrição de ação e a passa para startActivity(). **[2]** O sistema Android busca, em todos os aplicativos, um filtro de intents que corresponda ao intent. Ao encontrar uma correspondência, **[3]** o sistema inicia a atividade correspondente (*Atividade B*) chamando seu método onCreate() e passando-lhe o Intent.

Ao criar um intent implícito, o sistema Android encontra o componente adequado para iniciar, comparando o conteúdo do intent aos *filtros de intents* declarados no arquivo de manifesto de outros aplicativos no dispositivo. Se o intent corresponder a um filtro de intents, o sistema iniciará

esse componente e entregará o objeto Intent. Se diversos filtros de intents corresponderem, o sistema exibirá uma caixa de diálogo para que o usuário selecione o aplicativo que deseja usar.

O filtro de intents é uma expressão em um arquivo de manifesto do aplicativo que especifica o tipo de intents que o componente gostaria de receber. Por exemplo, ao declarar um filtro de intents para uma atividade, outros aplicativos se tornam capazes de iniciar diretamente sua atividade com o determinado tipo de intent. Do mesmo modo, se você *não* declarar nenhum filtro de intents para uma atividade, ela poderá ser iniciada somente com um intent explícito.

Atenção: para garantir a segurança do seu aplicativo, sempre use um intent explícito ao iniciar um `Service` e não declare filtros de intents para os serviços. O uso de um intent implícito para iniciar um serviço representa um risco de segurança, porque não é possível determinar qual serviço responderá ao intent e o usuário não poderá ver que serviço é iniciado. A partir do Android 5.0 (API de nível 21), o sistema lança uma exceção ao chamar `bindService()` com um intent implícito.

Criação de um intent

Um objeto Intent carrega informações que o sistema Android usa para determinar o componente a iniciar (como o nome exato do componente ou categoria do componente que deve receber o intent), além de informações que o componente receptor usa para realizar a ação adequadamente (como a ação a tomar e os dados a usar).

As informações principais contidas em um Intent são as seguintes:

Nome do componente

> É o nome do componente a iniciar.
>
> É opcional, mas é a informação fundamental que torna um intent *explícito*, o que significa que o intent deve ser entregue somente ao componente do aplicativo definido pelo nome do componente. Sem nome de componente, o intent será *implícito*, e o sistema decidirá qual componente deve receber o intent com base nas informações de outro intent (como a ação, os dados e a categoria — descritos abaixo). Portanto, se for necessário iniciar um componente específico no seu aplicativo, você deverá especificar o nome do componente.
>
> **Observação:** ao iniciar um `Service`, deve-se *sempre especificar o nome do componente*. Caso contrário, não será possível determinar qual serviço responderá ao intent e o usuário não poderá ver que serviço é iniciado.
>
> Esse campo do Intent é um objeto ComponentName que pode ser especificado usando um nome de classe totalmente qualificado do componente-alvo, inclusive o nome do pacote do aplicativo. Por exemplo, com.example.ExampleActivity. É possível definir o nome do

componente com setComponent(), setClass(), setClassName() ou com o construtor Intent.

Ação

String que especifica a ação genérica a realizar (como *exibir* ou *selecionar*).

No caso de um intent de transmissão, essa é a ação que entrou em vigor e que está sendo relatada. A ação determina amplamente como o resto do intent é estruturado — especificamente, o que está contido nos dados e em extras.

É possível especificar as ações para uso por intents dentro do aplicativo (ou para uso por outros aplicativos para chamar componentes no seu aplicativo), mas normalmente são usadas constantes de ação definidas pela classe Intent ou por outras classes de biblioteca. Veja algumas ações comuns para iniciar uma atividade:

ACTION_VIEW

Use essa ação em um intent com startActivity() quando houver informações de que uma atividade possa ser exibida ao usuário, como uma foto para exibição em um aplicativo de galeria ou um endereço para exibição em um aplicativo de mapa.

ACTION_SEND

Também conhecida como o intent de *share*, deve ser usada em um intent com startActivity() quando houver alguns dados que o usuário possa compartilhar por meio de outro aplicativo, como um app de e-mails ou de compartilhamento social.

Consulte a referência da classe Intent para saber mais constantes que definem ações genéricas. Outras ações são definidas em outros locais na biblioteca do Android, como nas Settings para ações que abrem telas específicas no aplicativo de Configurações do sistema.

É possível especificar a ação para um intent com setAction() ou com um construtor Intent.

Se você definir as próprias ações, verifique se incluiu o nome do pacote do seu aplicativo como prefixo, conforme o exemplo a seguir:

KotlinJava

```
const val ACTION_TIMETRAVEL = "com.example.action.TIMETRAVEL"
```

Dados

É o URI (um objeto Uri) que referencia os dados a serem aproveitados e/ou o tipo MIME desses dados. O tipo dos dados fornecidos geralmente é determinado pela ação do intent. Por exemplo, se a ação for ACTION_EDIT, os dados deverão conter o URI do documento a editar.

Ao criar um intent, em geral, é importante especificar o tipo de dados (seu tipo MIME) em adição ao URI. Por exemplo: uma atividade capaz de exibir imagens provavelmente não será capaz de reproduzir um arquivo de áudio, mesmo que os formatos do URI sejam similares. Portanto, especificar o tipo MIME dos dados ajuda o sistema Android a encontrar o melhor componente para receber o intent. Contudo, o tipo MIME, às vezes, pode ser inferido a partir do URI — especificamente quando os dados são um URI de content:. A URI de content: indica que os dados se localizam no dispositivo e são controlados por um ContentProvider, que torna o tipo MIME dos dados visível para o sistema.

Para definir só um URI de dados, chame setData(). Para definir só o tipo MIME, chame setType(). Se necessário, é possível definir ambos explicitamente com setDataAndType().

Atenção: se você desejar definir o URI e o tipo MIME, *não* chame `setData()` e `setType()`, pois um anula o outro. Sempre use `setDataAndType()` para definir o URI e o tipo MIME juntos.

Categoria

É uma string que contém informações adicionais sobre o tipo de componente que deve processar o intent. Qualquer número de descrições de categoria pode ser inserido em um intent, mas a maioria dos intents não requer nenhuma categoria. Veja algumas categorias comuns:

CATEGORY_BROWSABLE

A atividade-alvo permite ser iniciada por um navegador da Web para exibir dados referenciados por um link, como uma imagem ou uma mensagem de e-mail.

CATEGORY_LAUNCHER

A atividade é a atividade inicial de uma tarefa e é listada no inicializador do aplicativo do sistema.

Consulte a descrição da classe Intent para ver a lista completa de categorias.

É possível especificar uma categoria com addCategory().

As propriedades listadas abaixo (nome do componente, ação, dados e categoria) representam as características de definição de um intent. Ao ler estas propriedades, o sistema Android será capaz de definir o componente de aplicativo que ele deve iniciar. Contudo, um intent pode carregar informações adicionais que não afetam o modo com que é tratado para um componente do aplicativo. Os intents também podem fornecer o seguinte:

Extras

São pares de chave-valor que carregam informações adicionais necessárias para realizar a ação solicitada. Assim como algumas ações usam determinados tipos de URIs de dados, outras também usam determinados extras.

É possível adicionar dados extras com diversos métodos putExtra(), cada um aceitando dois parâmetros: o nome principal e o valor. Também é possível criar um objeto Bundle com todos os dados extras e, em seguida, inserir o Bundle no elemento Intent com putExtras().

Por exemplo, ao criar um intent para enviar um e-mail com ACTION_SEND, é possível especificar o recipiente *to* com a chave EXTRA_EMAIL e especificar *subject* com a chave EXTRA_SUBJECT.

A classe Intent especifica diversas constantes EXTRA_* para tipos de dados padronizados. Se for necessário declarar chaves extras (para intents que seu aplicativo receba), certifique-se de incluir o nome do pacote do aplicativo como prefixo, conforme o exemplo abaixo:

Kotlin Java

```
const val EXTRA_GIGAWATTS = "com.example.EXTRA_GIGAWATTS"
```

Atenção: não use dados **`Parcelable`** ou **`Serializable`** ao enviar um intent que você espera que outro aplicativo receba. Caso um aplicativo tente acessar dados em um objeto **`Bundle`**, mas não tenha acesso à classe parcelada ou serializada, o sistema cria um **`RuntimeException`**.

Sinalizadores

Sinalizadores são definidos na classe Intent e funcionam como metadados para o intent. Os sinalizadores podem instruir o sistema Android a inicializar uma atividade (por exemplo, a qual tarefa a atividade deve pertencer) e como tratá-la após a inicialização (por exemplo, se ela pertence a uma lista de atividades recentes).

Para mais informações, consulte o método setFlags().

Exemplo de intent explícito

O intent explícito é usado para inicializar um componente específico de um aplicativo, como uma atividade ou serviço em particular no seu aplicativo. Para criar um intent explícito, defina o nome do componente para o objeto Intent — todas as outras propriedades do intent são opcionais.

Por exemplo, se você criar um serviço no aplicativo chamado DownloadService, projetado para fazer o download de um arquivo da Web, poderá iniciá-lo com o código a seguir:

KotlinJava

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
val downloadIntent = Intent(this, DownloadService::class.java).apply {
    data = Uri.parse(fileUrl)
}
startService(downloadIntent)
```

O construtor Intent(Context, Class) fornece o Context do aplicativo e o componente fornece um objeto Class. Assim, esse intent inicia explicitamente a classe DownloadService no aplicativo.

Para mais informações sobre a criação e inicialização de um serviço, consulte o guia Serviços.

Exemplo de intent implícito

O intent implícito especifica uma ação que possa chamar qualquer aplicativo no dispositivo capaz de realizar a ação. O intent implícito é útil quando o aplicativo não pode realizar a ação, mas outros aplicativos provavelmente podem, e o usuário seleciona que aplicativo usar.

Por exemplo, se você tiver conteúdo que quer que o usuário compartilhe com outras pessoas, crie um intent com a ação ACTION_SEND e adicione extras que especifiquem o conteúdo a compartilhar. Ao chamar startActivity() com esse intent, o usuário poderá selecionar um aplicativo para compartilhar o conteúdo.

Atenção: é possível que um usuário não tenha *nenhum* aplicativo que processe o intent implícito enviado a `startActivity()`. Ou um aplicativo pode estar inacessível por causa das configurações ou restrições do perfil definidas pelo administrador. Se isso ocorrer, há uma falha de chamada e o aplicativo será interrompido. Para verificar se uma atividade receberá o intent, chame `resolveActivity()` no seu objeto `Intent`. Se o resultado for diferente de nulo, há pelo menos um aplicativo que pode processar o intent e é seguro chamar `startActivity()`. Se o resultado for nulo, você não deve usar o intent e, se possível, deve desativar o recurso que o emite. O exemplo abaixo mostra como verificar se o intent resulta em uma atividade. Nesse caso, não será usado nenhum URI, mas o tipo de dados do intent será declarado para especificar o conteúdo carregado pelos extras.

KotlinJava

```
// Create the text message with a string
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    putExtra(Intent.EXTRA_TEXT, textMessage)
    type = "text/plain"
}

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(packageManager) != null) {
    startActivity(sendIntent)
}
```

Quando startActivity() é chamada, o sistema avalia todos os aplicativos instalados para determinar quais deles podem processar esse tipo de intent (um intent com a ação ACTION_SEND e que carrega dados de “texto/simples”. Se houver somente um aplicativo que possa tratá-lo, o aplicativo se abrirá imediatamente e receberá o intent. Se diversas atividades aceitarem o intent, o sistema exibirá uma caixa de diálogo, conforme exibido na Figura 2, para que o usuário selecione que aplicativo usar.

Mais informações sobre iniciar outros aplicativos são fornecidas também em Como enviar o usuário para outro aplicativo.

Figura 2. Caixa de diálogo seletora.

Como forçar um seletor de aplicativo

Quando há mais de um aplicativo que responde ao intent implícito, o usuário pode selecionar o aplicativo que quer usar e tornar esse aplicativo a escolha padrão para a ação. Isso é positivo ao executar uma ação em que o usuário quer usar o mesmo aplicativo todas as vezes, como quando abre uma página da Web (os usuários geralmente usam somente um navegador).

Contudo, se diversos aplicativos puderem responder ao intent e o usuário tiver que ficar livre para usar um aplicativo diferente a cada vez, é preciso exibir uma caixa de diálogo seletora explicitamente. A caixa de diálogo seletora pede que o usuário selecione o aplicativo desejado para a ação todas as vezes (o usuário não pode selecionar um aplicativo padrão para a ação). Por exemplo: quando o aplicativo realiza “compartilhar” com a ação ACTION_SEND, os usuários podem querer compartilhar usando um aplicativo diferente conforme

a situação, portanto, deve-se sempre usar a caixa de diálogo seletora, como ilustrado na figura 2.

Para mostrar o seletor, crie um Intent usando createChooser() e passe-o para startActivity(), conforme exibido no exemplo a seguir: Isso exibirá uma caixa de diálogo com uma lista de aplicativos que respondem ao intent passado ao método createChooser() e usará o texto fornecido como título da caixa de diálogo.

KotlinJava

```
val sendIntent = Intent(Intent.ACTION_SEND)
...

// Always use string resources for UI text.
// This says something like "Share this photo with"
val title: String = resources.getString(R.string.chooser_title)
// Create intent to show the chooser dialog
val chooser: Intent = Intent.createChooser(sendIntent, title)

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(packageManager) != null) {
   startActivity(chooser)
}
```

Como receber um intent implícito

Para anunciar quais intents implícitos o aplicativo pode receber, declare um ou mais filtros de intents para cada um dos componentes do aplicativo com um elemento <intent-filter> no seu arquivo de manifesto. Cada filtro de intents especifica o tipo de intents aceito com base na ação, nos dados e na categoria do intent. O sistema fornecerá um intent implícito ao componente do seu aplicativo somente se ele puder passar por um dos filtros de intents.

Observação: o intent explícito é sempre entregue ao alvo independentemente dos filtros de intents que o componente declare.

Os componentes de um aplicativo devem declarar filtros separados para cada job exclusivo que podem fazer. Por exemplo, uma atividade em um aplicativo de galeria de imagens pode ter dois filtros: um filtro para visualizar uma imagem e outro para editar uma imagem. Quando a atividade se inicia, ela inspeciona o Intent e decide como se comportar com base nas informações no Intent (como para exibir ou não os controles do editor).

Cada filtro de intents é definido por um elemento <intent-filter> no arquivo de manifesto do aplicativo, aninhado no componente correspondente do aplicativo (como um elemento <activity>). Dentro de <intent-filter>, é possível

especificar o tipo de intents aceitos usando um ou mais dos três elementos a seguir:

<action>

Declara a ação do intent aceito, no atributo name. O valor deve ser o valor literal da string de uma ação, e não a constante da classe.

<data>

Declara o tipo de dados aceitos usando um ou mais atributos que especificam diversos aspectos do URI de dados (scheme, host, port, path) e o tipo MIME.

<category>

Declara a categoria do intent aceito, no atributo name. O valor deve ser o valor literal da string de uma ação, e não a constante da classe.

Observação: para receber intents implícitos, *você precisa incluir* a categoria `CATEGORY_DEFAULT` no filtro de intents. Os métodos `startActivity()` e `startActivityForResult()` tratam todos os intents como se eles declarassem a categoria `CATEGORY_DEFAULT`. Se você não a declarar no filtro de intents, nenhum intent implícito retomará sua atividade.

Por exemplo, abaixo há uma declaração de atividade com um filtro de intents para receber um intent ACTION_SEND quando o tipo de dados for texto:

```
<activity android:name="ShareActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
</activity>
```

Você pode criar um filtro que inclua mais de uma instância de <action>, <data> ou <category>. Se fizer isso, será necessário verificar se o componente pode processar todas as combinações dos elementos de filtro.

Para processar diversos tipos de intents, mas somente em combinações específicas de ações, dados e tipos de categoria, será necessário criar diversos filtros de intents.

Os intents implícitos são testados em relação a um filtro por meio da comparação do intent com cada um dos três elementos. Para ser entregue ao componente, o intent deve passar por todos os três testes. Se ele falhar em algum deles, o sistema Android não entregará o intent ao componente. No

entanto, como um componente pode ter diversos filtros de intents, um intent que não passe por um dos filtros de um componente pode passar por outro filtro. Veja mais informações sobre como o sistema resolve intents na seção abaixo sobre Resolução de intents.

Atenção: o uso de filtros de intent não é um modo seguro de evitar que outros aplicativos iniciem componentes. Embora os filtros de intents restrinjam um componente a responder somente a determinados tipos de intents implícitos, outro aplicativo pode iniciar o componente do seu aplicativo usando um intent explícito se o desenvolvedor determinar os nomes dos componentes. Se for importante que *somente seu próprio aplicativo* inicie um dos seus componentes, não declare filtros de intents no manifesto. Em vez disso, defina o atributo `exported` como `"false"` para esse componente.

De modo semelhante, para evitar a execução involuntária de um `Service` diferente do aplicativo, sempre use um intent explícito para iniciar o próprio serviço.

Observação: para todas as atividades, é necessário declarar os filtros de intents no arquivo de manifesto. No entanto, filtros para broadcast receivers podem ser registrados dinamicamente ao chamar `registerReceiver()`. Assim, será possível cancelar o registro do receptor com `unregisterReceiver()`. Isso permitirá que o aplicativo receba transmissões específicas durante um período de tempo especificado somente quando o aplicativo estiver em execução.

Exemplos de filtros

Para demonstrar alguns comportamentos do filtro de intents, veja um exemplo do arquivo de manifesto de um aplicativo de compartilhamento social.

```
<activity android:name="MainActivity">
  <!-- This activity is the main entry, should appear in app launcher -->
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>

<activity android:name="ShareActivity">
  <!-- This activity handles "SEND" actions with text data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
  </intent-filter>
  <!-- This activity also handles "SEND" and "SEND_MULTIPLE" with media data -->
  <intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <action android:name="android.intent.action.SEND_MULTIPLE"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="application/vnd.google.panorama360+jpg"/>
```

```
    <data android:mimeType="image/*"/>
    <data android:mimeType="video/*"/>
  </intent-filter>
</activity>
```

A primeira atividade, MainActivity, é o ponto de entrada principal do aplicativo — a atividade que se abre quando o usuário inicializa o aplicativo pela primeira vez com o ícone na tela de início:

- A ação ACTION_MAIN indica que esse é o ponto de entrada principal e não espera nenhum dado de intent.
- A categoria CATEGORY_LAUNCHER indica que esse ícone da atividade deve ser colocado no inicializador de aplicativo do sistema. Se o elemento <activity> não especificar nenhum ícone com icon, o sistema usará o ícone do elemento <application>.

Esses dois devem ser pareados para que a atividade apareça no inicializador do aplicativo.

A segunda atividade, ShareActivity, destina-se a facilitar o compartilhamento de conteúdo de texto e mídia. Apesar de os usuários poderem acessar essa atividade pela MainActivity, eles também podem acessar ShareActivity diretamente de outro aplicativo que emita um intent implícito que corresponda a um dos dois filtros de intents.

Mapas

Exibir um local em um mapa

Para abrir um mapa, use a ação ACTION_VIEW e especifique as informações de localização nos dados do intent com um dos esquemas definidos abaixo.

Ação

ACTION_VIEW

Esquema do URI dos dados

geo:*latitude*,*longitude*

Exibe o mapa na longitude e na latitude dadas.

Exemplo: "geo:47.6,-122.3"

geo:*latitude*,*longitude*?z=*zoom*

Exibe o mapa na longitude e na latitude dadas em um certo nível de zoom. Um nível de zoom de 1 mostra a Terra inteira, com centro em *lat*,*lng* dadas. O nível de zoom mais alto (mais perto) é 23.

Exemplo: "geo:47.6,-122.3?z=11"

geo:0,0?q=lat,lng(label)

Exibe o mapa na longitude e na latitude dadas com um rótulo de string.

Exemplo: "geo:0,0?q=34.99,-106.61(Treasure)"

geo:0,0?q=my+street+address

Mostra a localização para "meu endereço" (pode ser um endereço específico ou uma consulta de local).

Exemplo: "geo:0,0?q=1600+Amphitheatre+Parkway%2C+CA"

Observação: todas as strings passadas no URI de `geo` precisarão ser codificadas. Por exemplo, a string `1st & Pike, Seattle` precisará tornar-se `1st%20%26%20Pike%2C%20Seattle`. Espaços na string podem ser codificados com `%20` ou substituídos pelo sinal de mais (+).

Tipo MIME

Nenhum

Exemplo de intent:

KotlinJava

```
fun showMap(geoLocation: Uri) {
   val intent = Intent(Intent.ACTION_VIEW).apply {
      data = geoLocation
   }
   if (intent.resolveActivity(packageManager) != null) {
      startActivity(intent)
   }
}
```

Exemplo de filtro de intent:

```
<activity ...>
   <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <data android:scheme="geo" />
      <category android:name="android.intent.category.DEFAULT" />
```

```
    </intent-filter>
</activity>
```

Layouts em visualizações

bookmark_border

Um layout define a estrutura de uma interface de usuário no seu aplicativo, como em uma atividade . Todos os elementos do layout são construídos usando uma hierarquia de objetos Viewe . ViewGroupGeralmente , A Viewdesenha algo que o usuário pode ver e interagir. Considerando que a ViewGroupé um contêiner invisível que define a estrutura de layout de Viewoutros ViewGroupobjetos, conforme mostrado na figura 1.

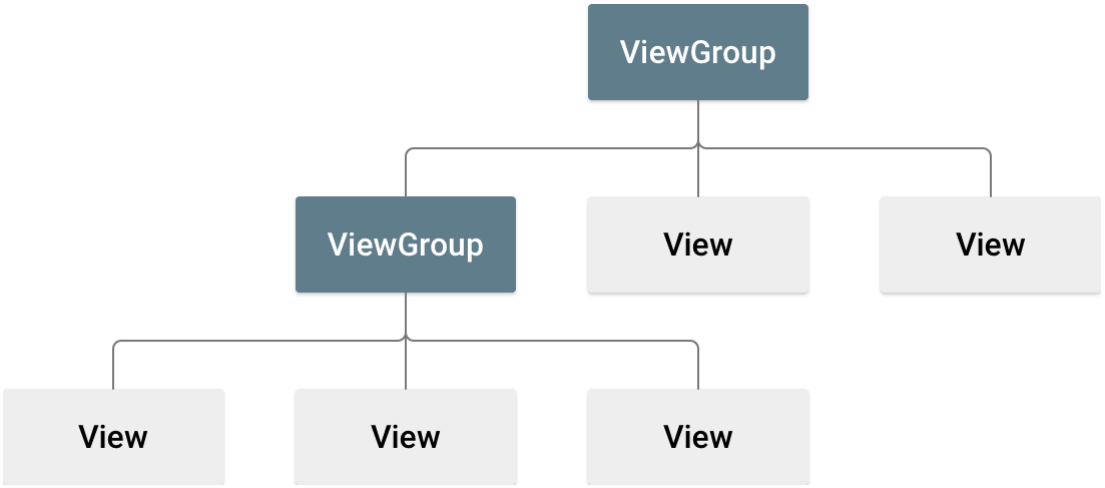

Figura 1. Ilustração de uma hierarquia de visualização, que define um layout de UI

Os Viewobjetos são geralmente chamados de "widgets" e podem ser uma de muitas subclasses, como Buttonou TextView. Os ViewGroupobjetos são geralmente chamados de "layouts" e podem ser de vários tipos que fornecem uma estrutura de layout diferente, como LinearLayoutou ConstraintLayout .

Você pode declarar um layout de duas maneiras:

- **Declarar elementos da UI em XML** . O Android fornece um vocabulário XML direto que corresponde às classes e subclasses View, como aquelas para widgets e layouts.

  Você também pode usar o Layout Editor do Android Studio para criar seu layout XML usando uma interface de arrastar e soltar.

- **Instancie elementos de layout em tempo de execução** . Seu aplicativo pode criar objetos View e ViewGroup (e manipular suas propriedades) programaticamente.

Declarar sua UI em XML permite separar a apresentação do seu aplicativo do código que controla seu comportamento. O uso de arquivos XML também facilita o fornecimento de diferentes layouts para diferentes tamanhos e orientações de tela (discutido mais detalhadamente em Suporte a diferentes tamanhos de tela).

A estrutura do Android oferece flexibilidade para usar um ou ambos os métodos para criar a IU do seu aplicativo. Por exemplo, você pode declarar os layouts padrão do seu aplicativo em XML e depois modificar o layout em tempo de execução.

Dica: Para depurar seu layout em tempo de execução, use a ferramenta Layout Inspector .

Escreva o XML

Usando o vocabulário XML do Android, você pode projetar rapidamente layouts de UI e os elementos de tela que eles contêm, da mesma forma que você cria páginas da web em HTML – com uma série de elementos aninhados.

Cada arquivo de layout deve conter exatamente um elemento raiz, que deve ser um objeto View ou ViewGroup. Depois de definir o elemento raiz, você pode adicionar objetos de layout ou widgets adicionais como elementos filhos para construir gradualmente uma hierarquia de visualização que defina seu layout. Por exemplo, aqui está um layout XML que usa uma vertical LinearLayout para conter a TextViewe a Button:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical" >
  <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
  <Button android:id="@+id/button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
```

```
        android:text="Hello, I am a Button" />
</LinearLayout>
```

Após declarar seu layout em XML, salve o arquivo com a extensão, no diretório .xmldo seu projeto Android , para que ele seja compilado corretamente.res/layout/

Mais informações sobre a sintaxe de um arquivo XML de layout estão disponíveis no documento Recursos de layout .

Carregar o recurso XML

Ao compilar seu aplicativo, cada arquivo de layout XML é compilado em um Viewrecurso. Você deve carregar o recurso de layout do código do seu aplicativo, na Activity.onCreate()implementação do retorno de chamada. Faça isso chamando setContentView(), passando a referência ao seu recurso de layout na forma de: . Por exemplo, se o seu layout XML for salvo como , você o carregaria para sua atividade da seguinte forma:R.layout.*layout_file_name*main_layout.xml

KotlinJava

```
fun onCreate(savedInstanceState: Bundle) {
  super.onCreate(savedInstanceState)
  setContentView(R.layout.main_layout)
}
```

O onCreate()método de retorno de chamada em sua Activity é chamado pela estrutura do Android quando sua Activity é iniciada (veja a discussão sobre ciclos de vida, no documento Atividades).

Atributos

Cada objeto View e ViewGroup suporta sua própria variedade de atributos XML. Alguns atributos são específicos de um objeto View (por exemplo, TextView suporta o textSize atributo), mas esses atributos também são herdados por quaisquer objetos View que possam estender esta classe. Alguns são comuns a todos os objetos View, porque são herdados da classe View raiz (como o idatributo). E outros atributos são considerados "parâmetros de layout", que são atributos que descrevem determinadas orientações de layout do objeto View, conforme definido pelo objeto ViewGroup pai desse objeto.

ID

Qualquer objeto View pode ter um ID inteiro associado a ele, para identificar exclusivamente a View dentro da árvore. Quando o aplicativo é compilado, esse ID é referenciado como um número inteiro, mas normalmente é atribuído no arquivo XML de layout como uma string, no idatributo. Este é um atributo XML comum a todos os objetos View (definidos pela Viewclasse) e você o utilizará com muita frequência. A sintaxe para um ID, dentro de uma tag XML é:

```
android:id="@+id/my_button"
```

O símbolo arroba (@) no início da sequência indica que o analisador XML deve analisar e expandir o restante da sequência de ID e identificá-la como um recurso de ID. O símbolo de mais (+) significa que este é um novo nome de recurso que deve ser criado e adicionado aos nossos recursos (no R.javaarquivo). Existem vários outros recursos de ID oferecidos pela estrutura Android. Ao fazer referência a um ID de recurso do Android, você não precisa do símbolo de adição, mas deve adicionar o androidnamespace do pacote, da seguinte forma:

```
android:id="@android:id/empty"
```

Com o androidnamespace do pacote instalado, agora estamos referenciando um ID da android.R classe de recursos, em vez da classe de recursos locais.

Para criar visualizações e referenciá-las no aplicativo, um padrão comum é:

1. Defina uma visualização/widget no arquivo de layout e atribua a ele um ID exclusivo:

   ```
   <Button android:id="@+id/my_button"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="@string/my_button_text"/>
   ```

2. Em seguida, crie uma instância do objeto de visualização e capture-a do layout (normalmente no onCreate()método):

   KotlinJava

   ```
   val myButton: Button = findViewById(R.id.my_button)
   ```

Definir IDs para objetos de visualização é importante ao criar um arquivo RelativeLayout. Em um layout relativo, as visualizações irmãs podem definir seu layout em relação a outra visualização irmã, que é referenciada pelo ID exclusivo.

Um ID não precisa ser exclusivo em toda a árvore, mas deve ser exclusivo na parte da árvore que você está pesquisando (que muitas vezes pode ser a

árvore inteira, por isso é melhor ser completamente exclusivo quando possível).

Observação: com o Android Studio 3.6 e versões posteriores, o recurso de vinculação de visualizações pode substituir `findViewById()` chamadas e fornece segurança de tipo em tempo de compilação para código que interage com visualizações. Considere usar vinculação de visualização em vez de `findViewById()`.

Parâmetros de layout

Os atributos de layout XML nomeados definem parâmetros de layout para a Visualização que são apropriados para o ViewGroup no qual ela reside.layout_*something*

Cada classe ViewGroup implementa uma classe aninhada que estende ViewGroup.LayoutParams. Esta subclasse contém tipos de propriedades que definem o tamanho e a posição de cada visualização filha, conforme apropriado para o grupo de visualizações. Como você pode ver na figura 2, o grupo de visualizações pai define parâmetros de layout para cada visualização secundária (incluindo o grupo de visualizações secundárias).

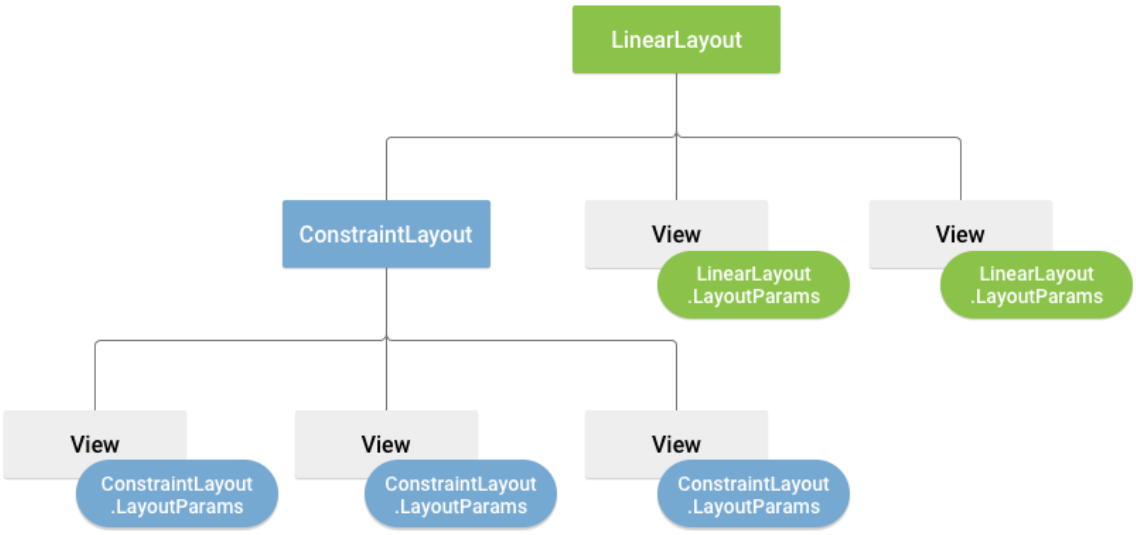

Figura 2. Visualização de uma hierarquia de visualizações com parâmetros de layout associados a cada visualização

Observe que cada subclasse LayoutParams possui sua própria sintaxe para definir valores. Cada elemento filho deve definir LayoutParams apropriados para seu pai, embora também possa definir LayoutParams diferentes para seus próprios filhos.

Todos os grupos de visualizações incluem largura e altura (layout_widthe layout_height), e cada visualização é necessária para defini-los. Muitos LayoutParams também incluem margens e bordas opcionais.

Você pode especificar largura e altura com medidas exatas, embora provavelmente não queira fazer isso com frequência. Mais frequentemente, você usará uma destas constantes para definir a largura ou a altura:

- *wrap_content* diz à sua visualização para dimensionar-se de acordo com as dimensões exigidas pelo seu conteúdo.
- *match_parent* diz à sua visualização para se tornar tão grande quanto o grupo de visualização pai permitir.

Em geral, não é recomendado especificar a largura e a altura do layout usando unidades absolutas, como pixels. Em vez disso, usando medições relativas, como unidades de pixel independentes de densidade (*dp*),*wrap_content*, ou *match_parent*, é uma abordagem melhor, pois ajuda a garantir que seu aplicativo seja exibido corretamente em vários tamanhos de tela de dispositivos. Os tipos de medição aceitos são definidos no documento Recursos Disponíveis .

Posição do layout

A geometria de uma vista é a de um retângulo. Uma vista tem uma localização, expressa como um par de coordenadas *esquerda* e *superior* , e duas dimensões, expressas como largura e altura. A unidade de localização e dimensões é o pixel.

É possível recuperar a localização de uma visualização invocando os métodos getLeft()e getTop(). O primeiro retorna a coordenada esquerda, ou X, do retângulo que representa a visualização. O último retorna a coordenada superior, ou Y, do retângulo que representa a visualização. Ambos os métodos retornam a localização da visualização em relação ao seu pai. Por exemplo, quando getLeft()retorna 20, isso significa que a visualização está localizada 20 pixels à direita da borda esquerda de seu pai direto.

Além disso, vários métodos de conveniência são oferecidos para evitar cálculos desnecessários, nomeadamente getRight()e getBottom(). Esses métodos retornam as coordenadas das bordas direita e inferior do retângulo que representa a visualização. Por exemplo, a chamada getRight() é semelhante ao seguinte cálculo: getLeft() + getWidth().

Tamanho , preenchimento e margens

O tamanho de uma visualização é expresso com largura e altura. Na verdade, uma visualização possui dois pares de valores de largura e altura.

O primeiro par é conhecido como *largura medida* e *altura medida*. Essas dimensões definem o tamanho que uma visualização deseja ter dentro de seu pai. As dimensões medidas podem ser obtidas chamando getMeasuredWidth() e getMeasuredHeight().

O segundo par é conhecido simplesmente como *largura* e *altura* , ou às vezes *largura* e *altura do desenho* . Estas dimensões definem o tamanho real da vista na tela, no momento do desenho e após o layout. Esses valores podem, mas não necessariamente, ser diferentes da largura e altura medidas. A largura e a altura podem ser obtidas chamando getWidth()e getHeight().

Para medir suas dimensões, uma visualização leva em consideração seu preenchimento. O preenchimento é expresso em pixels para as partes esquerda, superior, direita e inferior da visualização. O preenchimento pode ser usado para compensar o conteúdo da visualização em um número específico de pixels. Por exemplo, um preenchimento esquerdo de 2 empurrará o conteúdo da visualização em 2 pixels para a direita da borda esquerda. O preenchimento pode ser definido usando o setPadding(int, int, int, int)método e consultado chamando getPaddingLeft(), e . getPaddingTop()getPaddingRight()getPaddingBottom()

Embora uma visualização possa definir um preenchimento, ela não fornece suporte para margens. No entanto, os grupos de visualização fornecem esse suporte. Consulte ViewGroupe ViewGroup.MarginLayoutParamspara mais informações.

Para obter mais informações sobre dimensões, consulte Valores de dimensão .

Layouts comuns

Cada subclasse da ViewGroupclasse fornece uma maneira exclusiva de exibir as visualizações aninhadas nela. Abaixo estão alguns dos tipos de layout mais comuns integrados à plataforma Android.

Observação: embora você possa aninhar um ou mais layouts em outro layout para atingir seu design de IU, você deve se esforçar para manter sua hierarquia de layout o mais superficial possível. Seu layout será desenhado mais rápido se tiver menos layouts aninhados (uma hierarquia de visualização ampla é melhor que uma hierarquia de visualização profunda).

Layout Linear

Um layout que organiza seus filhos em uma única linha horizontal ou vertical. Ele cria uma barra de rolagem se o comprimento da janela exceder o comprimento da tela.

esquema relativo

Permite especificar a localização dos objetos filhos em relação uns aos outros (filho A à esquerda do filho B) ou ao pai (alinhado à parte superior do pai).