

Computação

Movel

Apresentação do Mercado de Aplicações Móveis, Sistemas Operacionais de Dispositivos Móveis e a Handset Alliance

Introdução

Seja bem vindo, prezado aluno! Desejamos a você um ótimo aprendizado e que este o leve ao sucesso.

Sendo este um ramo recente da área de computação, é de se esperar que esteja se perguntando, “Afinal, o que é computação móvel?”, e este é um ótimo ponto de partida para inicializarmos o estudo desta matéria.

A disciplina de computação móvel será responsável por guia-lo pelo incrível mundo do desenvolvimento de aplicações para dispositivos móveis, sendo estes, celulares, smartphones e tablets, onde, sistema operacional android será utilizado como base para nossos desenvolvimentos.

Mercado de Computação Móvel

Substituir caneta e papel, as famigeradas maletas com pastas contendo prospectos de produtos, tabelas de preços e fichas com nome dos clientes por apenas um dispositivo que cabe na palma da mão e que possa armazenar essas e muitas outras informações é uma ideia tentadora e que está em plena expansão.

A demanda por recursos tecnológicos que proporcionem maior praticidade e agilidade na realização de tarefas cotidianas seja em casa, no trabalho, nos momentos de lazer ou até mesmo no trânsito, tem exigido dos fabricantes de eletrônicos portáteis como celulares, tablets, smartphones, Personal Digital Assistants (PDAs) cada vez maior superação na criatividade e inovação a fim de atender a este requisito.

Além de outras tecnologias como: 3G, GPS, Wifi, etc., que foram agregadas e permitiram aos usuários inúmeras possibilidades de utilização, como jamais se

pensou. Por este motivo eles estão ocupando cada vez mais espaço na vida das pessoas.

De acordo com ANATEL (Agência Nacional de Telecomunicações), foram registrados até maio de 2011 mais de 215 milhões de acessos na telefonia celular e só nos primeiros meses deste ano mais de 12 milhões de novas habilitações. Este número corresponde a 5,95% de crescimento em relação aos últimos onze anos.

No mercado corporativo a utilização do conceito de mobilidade também está em plena expansão, é a chamada “força de trabalho móvel”. Por isso diversas empresas estão buscando incorporar aplicações móveis a seu dia-a-dia para agilizar seus negócios e integrar as aplicações móveis com seus sistemas de back-end.

Dispositivos Móveis

Smartphone é um telefone móvel com funcionalidades avançadas que podem ser estendidas por meio de programas executados por seu sistema operacional. Os sistemas operacionais dos smartphones permitem que desenvolvedores criem milhares de programas adicionais, com diversas utilidades, agregados em sites como o Google Play. Geralmente um smartphone possui características mínimas de hardware e software, sendo as principais a capacidade de conexão com redes de dados para acesso à internet, a capacidade de sincronização dos dados do organizador com um computador pessoal, e uma agenda de contatos que pode utilizar toda a memória disponível do celular – não é limitada a um número fixo de contatos. Um smartphone pode ser considerado um telefone celular com as funcionalidades de um PDA.

Personal digital assistants - assistente pessoal digital, (PDAs , handhelds), ou palmtop, é um computador de dimensões reduzidas (cerca de A6), dotado de grande capacidade computacional, cumprindo as funções de agenda e sistema informático de escritório elementar, com possibilidade de

interconexão com um computador pessoal e uma rede informática sem fios — Wi-Fi — para acesso a e-mail e internet.

Os PDAs de hoje possuem grande quantidade de memória e diversos softwares para várias áreas de interesse.

Os modelos mais sofisticados possuem modem (para acesso à internet), câmera digital acoplada (para fotos e filmagens), tela colorida, rede sem fio embutida.

Telefone celular é um aparelho de comunicação por ondas electromagnéticas que permite a transmissão bidirecional de voz e dados utilizáveis em uma área geográfica que se encontra dividida em células (de onde provém a nomenclatura celular), cada uma delas servida por um transmissor/receptor. A invenção do telefone celular ocorreu em 1947 pelo laboratório Bell, nos Estados Unidos.

Tablet, também conhecido como tablet PC, é um dispositivo pessoal em formato de prancheta que pode ser usado para acesso à Internet, organização pessoal, visualização de fotos, vídeos, leitura de livros, jornais e revistas e para entretenimento com jogos. Apresenta uma tela sensível ao toque (touchscreen) que é o dispositivo de entrada principal. A ponta dos dedos ou uma caneta aciona suas funcionalidades. É um novo conceito: não deve ser igualado a um computador completo ou um smartphone, embora possua funcionalidades de ambos.

Sistemas Operacionais Móveis

Windows Mobile

É o sistema operacional desenvolvido pela Microsoft especialmente para dispositivos móveis. O Windows Mobile permite uma adaptação mais rápida de aplicativos desenvolvidos, a princípio, para o Windows do seu computador (ex.: Word, Outlook, etc).

Além disso, promove facilmente a conexão de seu aparelho celular a uma rede sem fio. Recentemente, sua versão passou a apresentar suporte nativo às tecnologias Wi-Fi e Bluetooth, sem que seja preciso arquivos extras para acessá-los.

O Windows Mobile contém integração com o Microsoft Exchange Server 2003, o que implica em possibilidade de sincronia de emails e arquivos pessoais entre o seu computador de mesa ou notebook e o seu celular. Também disponibiliza o acesso ao Windows Media 9.

iOS

O sistema operacional do Iphone é tido como um dos mais avançados. Com uma interface prática, ótimas ferramentas e estabilidade, o iOS é a base do iPhone.

Ele apresenta uma interface multi-toch extremamente funcional. Independente do que esteja utilizando no celular, você tem o controle absoluto apenas pelo toque do dedo. E o enfoque deste sistema operacional é na multitarefa, possibilitando que o usuário consiga rodar seus aplicativos favoritos e alternar entre eles rapidamente, sem impacto no desempenho.

Um dos recursos mais atraentes é o FaceTime: a chamada de telefone por vídeo. Basta um toque para ver seus amigos e familiares enquanto conversa com eles - isso entre aparelhos iPhone 4 via WiFi.

Symbian OS

Desenvolvido por um consórcio de várias empresas de telefonia, é um sistema operacional para aparelhos móveis com sistema modular. Exatamente por ser um sistema modular, o Symbian OS permite que cada uma destas empresas constantes no consórcio que o criou desenvolva a sua própria interface.

O Symbian OS permite o desenvolvimento de sistemas diferenciados, que vão desde textos em telas simples e monocromáticas até sofisticados sistemas disponibilizados em smartphones de marcas como Nokia e Ericsson.

O Symbian OS foi construído principalmente para gerar economia de recursos em aparelhos móveis, por isso, traz consigo diversos mecanismos para assegurar que os aplicativos do aparelho vão utilizar a menor quantidade de memória possível.

Android

Em 2007, foi criado um consórcio chamado Open Handset Alliance formado por 47 empresas e liderado pelo Google. A função desta reunião de empresas era criar um novo sistema operacional para celulares que fosse um sistema aberto, ou seja, sem que fabricantes tivessem que pagar licença de uso. O nome deste sistema adivinha qual é? Isso mesmo: Android.

Seu grande diferencial é que, como o código é liberado, programadores de todo o mundo criam aplicativos, ou seja, pequenos programas para rodar nos celulares que tenham o sistema. Esses aplicativos ficam todos catalogados e à disposição dos usuários na Android Market, uma loja virtual acessível via celular, muitos deles gratuitos.

A grande vantagem é que você melhora as funções do telefone móvel através de programas que são elaborados justamente para isso: melhorar o funcionamento do aparelho. São aplicativos tão variados, como leitores de ebooks, tocadores de mp3, bússolas digitais e, claro, joguinhos.

Handset Alliance

O que é a Open Handset Alliance?

A Open Handset Alliance é um grupo de 84 empresas de tecnologia e móveis que se uniram para acelerar a inovação em consumidores móveis e oferecer uma experiência mais rica, menos caro, e melhor móvel. Juntos desenvolvemos Android [™], a primeira completa, aberta e plataforma móvel livre. Estamos empenhados em implantar comercialmente aparelhos e serviços usando a plataforma Android.

Que tipos de empresas estão na Open Handset Alliance?

Todas as partes do ecossistema móvel estão representados na Aliança. Os membros incluem as operadoras móveis, fabricantes de aparelhos, empresas de semicondutores, empresas de software e empresas de comercialização.

Plataformas de Desenvolvimento para Aplicações Móveis

J2ME

Java 2 Micro Edition (J2ME) é a versão da linguagem Java idealizada para programação em dispositivos móveis, que possuem as seguintes características: mobilidade, baixa capacidade de processamento e pouca memória, alimentação por baterias, áreas de vídeo pequenas e limitadas e vários métodos de entrada e saída, ou seja dispositivos como os vistos na aula passada telefones celulares, PDAs(Assistentes Digitais Pessoais), smartphones, entre outros.

Devemos entender também que o J2ME não é novo ou outro tipo de Java, mas sim, de uma adaptação da plataforma existente para que se possa desenvolver de maneira mais eficiente programas para dispositivos móveis com as características acima, nesta linguagem. Deste modo, todos os programas desenvolvidos utilizando-se a plataforma J2ME também poderão ser executados sem nenhum problema nas edições Standard (J2SE) e Enterprise (J2EE), isso logicamente se APIs usadas no desenvolvimento estejam presentes na plataforma utilizada.

O que podemos fazer com J2ME?

Com a plataforma J2ME procurou-se estender os conceitos utilizados na padronização de desenvolvimento de aplicações para pequenos dispositivos, idealizando pacotes que fazem transparecer ao desenvolvedor todos os detalhes proprietários do fabricante, como por exemplo, arquitetura de hardware utilizado e sistema operacional do dispositivo.

Para que se possa entender a evolução que isso representa, anteriormente todo dispositivo possuía uma estrutura constante, oferecido com um conjunto fixo de funcionalidades onde a programação era realizada exclusivamente pelo fabricante, utilizando uma tecnologia produzida

exclusivamente pelos proprietários. Com o J2ME, isso sofreu uma grande alteração tornando possível desenvolver, atualizar e instalar outros aplicativos que variariam com as necessidades de cada usuário. Sendo que é possível, depois de feito o download do programa para o dispositivo, usa-lo on-line ou off-line, sendo que uma das possibilidades é que caso se esteja trabalhando off-line, fazer a sincronização dos dados e das informações utilizadas quando a rede estiver presente, se este for o caso. Na plataforma J2ME pode-se desenvolver desde jogos, aplicações utilitárias domésticas, aplicações que acessam banco de dados, etc.

Tendo o J2ME como ferramenta de programação para aparelhos móveis se ganha todas as vantagens desta tecnologia, como listado abaixo:

- Dinamismo: pode-se fazer download de novas aplicações da rede e instaladas em um dispositivo móvel a qualquer instante;
- Segurança: garante a proteção das informações presentes no dispositivo móvel. Os dados utilizados em uma aplicação não são acessíveis por outras. Havendo uma verificação de classes, forte tipagem, garbage collection etc,;
- Portabilidade: As aplicações podem ser executadas em uma grande variedade de dispositivos, independente do fabricante e de seus tipos;
- Orientação a Objetos: alto nível de abstração do código, modularização e reusabilidade.

Android

Android é um conjunto de programas para dispositivos móveis que estão inseridos em um sistema operacional, um middleware e um conjunto de aplicações chaves. Sendo que desenvolvedores podem construir programas para esta plataforma fazendo uso do Android SDK. Os programas feitos para serem executados no android são comumente escritos em linguagem de programação Java e executam sobre o Dalvik, uma máquina virtual idealizada especialmente para dispositivos que possuam principalmente poucos recursos,

como por exemplo, baixa capacidade computacional, pouca área para armazenamento de dados.

Arquitetura

Kernel do Linux

A arquitetura do Android foi feita com base no kernel do GNU/Linux, versão 2.6. O kernel do sistema funciona como uma camada de abstração entre o hardware e o restante da pilha de softwares da plataforma.

O kernel GNU/Linux possui vários recursos necessários para a execução de aplicações, como gerenciamento de memória, gerenciamento de processos, pilha de protocolos de rede, módulo de segurança e vários módulos do núcleo de infra-estrutura.

Desta maneira, quando existe a necessidade de desenvolver-se, por exemplo, um novo Driver ou fazer-se melhorias num existente, isso será muito facilitado uma vez que o sistema operacional é conhecido.

Bibliotecas

O Android possui um novo grupo de bibliotecas C/C++ que são utilizados pelos diversos componentes do sistema. Sendo que as funcionalidades são utilizadas no framework do Android. Algumas destas bibliotecas que fazem parte núcleo da arquitetura estão listadas abaixo:

- Surface Manager: Gerencia o acesso ao subsistema de display. Gera as camadas gráficas 2D e 3D de múltiplas aplicações.
- 3D libraries: Representa uma implementação baseada na estrutura utilizada no OpenGL 1.0.
- SGL: Biblioteca utilizada na criação de gráficos 2D.
- Media libraries: Bibliotecas que suportam playback e gravação de formatos variados de áudio e de vídeo, bem como imagens

estáticas, os formatos disponíveis são MPEG4, H.264, MP3, AAC, AMR, JPG e PNG.

- FreeType: Biblioteca utilizada para desenhar fontes.
- SSL: Fornece encriptação de dados enviados pela Internet.
- SQLite: Biblioteca na linguagem C que dá suporte a estrutura do banco de dados SQL presente no android.

Precisamos ter em mente que quando programas que utilizam a SQLite possuem as interações com o banco de dados SQL sem executar um processo RDBMS em separado. Pois o SQLite não é apenas uma biblioteca de cliente utilizada para prover a conexão com um servidor de banco de dados. SQLite é um servidor de banco de dados. A SQLite pode ler e escrever diretamente no arquivo do banco de dados do android.

Android Runtime

Quando executamos uma aplicação Android, ela cria seu próprio processo, ou seja, cria sua própria instância da máquina virtual Dalvik. A máquina virtual Dalvik de modo que um dispositivo possa executar múltiplas máquinas virtuais simultaneamente da maneira mais eficiente possível.

A máquina virtual Dalvik executa uma série de classes compiladas da linguagem Java. Sendo que os arquivos .class gerados são transformados no formato .dex pela ferramenta dx, incluída no SDK (Software Development Kit) do Android, e são esses arquivos .dex são executados pela máquina virtual Dalvik.

Lembrando que a máquina virtual Dalvik, se utiliza do kernel do GNU/Linux, o que prove múltiplas funcionalidade, como por exemplo, threads e gerenciamento de memória de baixo nível.

No entanto existem diferenças entre a máquina virtual Dalvik e a máquina virtual Java (JVM), uma delas é que a Dalvik é baseada em registradores, enquanto que JVM é baseada em pilhas. Isso foi feito devido ao

fato de que uma estrutura baseada em registradores traz um benefício em ambientes restritos, como telefones celulares, além de que numa análise mais detalhada percebemos que máquinas virtuais baseadas em registradores executam mais rapidamente os programas do que máquinas baseadas em pilhas.

Outra diferença importante que está presente na Dalvik é a estrutura que permite que múltiplas instâncias da máquina virtual possam ser executadas ao mesmo tempo com pequenas quantidades de memória. Sendo que cada aplicação é executada como um processo Linux independente.

XCode

XCode, trata-se de um ambiente integrado de desenvolvimento de softwares que gerencia projetos baseados na estrutura do sistema operacional de Mac OS X, criado com sendo um software livre da Apple Inc. O XCode prove um grupo de ferramentas para que o usuário criar e aprimorar seus aplicativos. Sendo também um software poderoso, porém simples de utilizar, principalmente no desenvolvimento de aplicativos grandes. O Xcode pode ser obtido no DVD de instalação do Mac OS X nas versões "Leopard" e "Snow Leopard" e no website de desenvolvedores da Apple (<http://developer.apple.com/>), o Xcode NÃO será distribuído com o Mac OS X "Lion", pois o mesmo não possui DVD de instalação.

Desenvolvimento para Mac OS X

O Xcode possui todas as ferramentas necessárias ao desenvolvimento de aplicações para o Mac OS X, e suporta, por padrão, Objective-C e AppleScript, que são suas linguagens de programação.

Desenvolvimento para iOS

O Xcode também possui um conjunto de ferramentas "extras" de desenvolvimento, esses "extras" são o SDK, que são fornecidos pela Apple Inc. no website de desenvolvimento de iOS.

Apresentação das Principais APis Presentes na Estrutura do Android

Introdução

Antes de inicializar, precisamos saber o que é uma API.

API, é a abreviatura de Application Programming Interface (ou Interface de Programação de Aplicativos), uma API é constituída de uma coleção de funções e determinações de padrões para serem utilizados por um software sem que haja a necessidade do conhecimento dos detalhes da implementação da mesma, mas apenas como usar suas rotinas.

De um modo geral, uma API é composta por um conjunto de funções que só poderemos ter acesso quando utilizamos as mesmas durante a montagem de um programa.

Um exemplo interessante disto são as APIs de um sistema operacional, ela possui uma grande quantidade de funções, que possibilitam ao programador diversas facilidades, como por exemplo, criar janelas, acessar arquivos, criptografar dados etc. No entanto de desejarmos operações de nível mais baixo, como a manipulação de blocos de memória e acesso a dispositivos, não será possível, pois as essas funções costumam ser dissociadas das tarefas mais essenciais, e são tarefas realizadas pelo núcleo do sistema operacional.

APis Android

As APIs Nativas do android são:

android.util	Contém várias classes utilitárias (classes de containers, utilitários XML)
--------------	--

android.os	Contém serviços referentes ao sistema operacional, passagem de parâmetros e comunicação entre processos.
android.graphics	Pacote principal dos recursos gráficos.
android.text	
android.text.method	
android.text.style	
android.text.util	Suporte para um conjunto de ferramentas de processamento de texto, suporte ao formato de texto rico (RTF), métodos de entradas, etc.
android.database	Contém APIs para comunicação com o banco de dados SQLite
android.content	APIs de acesso a dados no dispositivo, como as aplicações instaladas e seus recursos.
android.view	O pacote principal que contém os principais componentes de interface gráfica.
android.widget	Contém widgets prontos (botões, listas, gerenciadores de layout, etc) para serem utilizados nas aplicações
android.app	APIs de alto-nível referentes ao modelo da aplicação. É implementada por meio de Activities (Atividades)
android.provider	Contém várias APIs para padrões de provedores de conteúdos (content providers)
android.telephony	APIs para interagir com funcionalidades de telefonia

`android.webkit` Inclui várias APIs para conteúdos de context web, bem como um navegador embutido que pode ser utilizado por qualquer aplicação.

Além das APIs nativas existem outras APIs desenvolvidas especialmente para android, elas tem por objetivo facilitar o desenvolvimento de aplicações diversas nesta plataforma, um bom exemplo disto são as APIs fornecidas pelo google com o exemplo a seguir.

A API Google para Android é uma extensão para o ambiente de desenvolvimento SDK do Android que fornece aos seus aplicativos Android acesso fácil aos serviços e dados do Google. O recurso central do complemento é a biblioteca externa Maps, que permite a adição de recursos avançados de mapeamento ao seu aplicativo Android.

As classes que pertencem a hierarquia do pacote `com.google.android.maps` que contempla a API Google para Android são:

`com.google.android.maps.MapActivity`

Activity necessárias para visualizar um mapa.

`com.google.android.maps.GeoPoint`

Valores de uma coordenadas geográficas.

`com.google.android.maps.MapController`

Controle de navegação e zoom.

`com.google.android.maps.Overlay`

Possibilidade de sobrepor elementos ao mapa.

`com.google.android.maps.ItemizedOverlay<Item>`

Consiste numa lista de `OverlayItems`.

`com.google.android.maps.MyLocationOverlay`

Desenha a atual localização do usuário no mapa.

`com.google.android.maps.OverlayItem`

Elemento de sobreposição.

`com.google.android.maps.TrackballGestureDetector`

Analisa eventos de movimento e detecta gestos.

`com.google.android.maps.MapView`

Objeto gráfico obtido a partir do serviço Google Maps.

`com.google.android.maps.MapView.LayoutParams`

Informações de layout associados ao MapView.

API de Localização

Outra API interessante é a de localização, que fornece classes que provêm serviços de localização. Seus métodos estão no pacote `android.location`.

A API location contém oito classes e três interfaces, sendo que as quatro classes principais são:

`LocationManager` que contém o acesso aos serviços de localização, monitora os eventos e as atividades associadas ao processo de localização. Por meio dessa classe é possível saber a posição geográfica do dispositivo móvel e enviar eventos caso o dispositivo se encontre nas proximidades de um local específico.

`LocationProvider` é uma superclasse abstrata para os prestadores de localização, tem por objetivo prover serviços de localização obedecendo a critérios estabelecidos, como informações de velocidade, altitude, bússola e outros.

Location que representa uma localização geográfica num determinado momento, reconhecendo os pontos geográficos por meio de sua latitude e longitude.

Geocoder manipula a geocodificação e geocodificação reversa, fazendo com que valores de latitude e longitude sejam convertidos em um endereço, podendo obter o nome e o código postal.

Protocolos de Redes Sem Fio, Dispositivos, Componentes e Assessórios de Equipamentos Móveis

Introdução

Quando se desenvolve aplicações para dispositivos móveis, novos obstáculos surgem devido aos recursos limitados disponíveis neste tipo de dispositivo, como tamanho da tela reduzida, pouca memória disponível, e baixo poder de processamento. Além de outros problemas que surgem em função do ambiente utilizado por estes dispositivos; a mobilidade e as redes sem fio que oferecem uma banda com menos recursos e menor confiabilidade no transporte de informações se comparada com os outros tipo de aplicação.

Restrições em Aplicações Sem Fio

Devido, principalmente, ao aumento do numero de dispositivos sem fio, o mercado esta realizando um grande esforço na tentativa de adaptar quase totalmente qualquer outra tecnologia desenvolvida para computadores de mesa de modo que estejam disponíveis em dispositivos móveis.

Porem aplicações sem fio deve seguir dentro de restrições impostas pelos dispositivos, tais como:

- Pouca memória: gerenciamento de memória um fator de extrema importância, uma vez que os dispositivos como telefones celulares e tablets possuem pouca memória.
- Baixo poder de processamento: os dispositivos de sem fio possuem uma baixa capacidade de processamento (variando de 32Kbytes a 64Mbytes).
- Entrada de dados: neste item existe uma grande variação nos tipos de entrada disponíveis no mercado, no entanto, a maior parte dos telefones celulares dispõe de uma entrada com poucos botões que variam de acordo com o aparelho e a tecnologia envolvida.

- Tela: Neste item também encontramos uma grande variação, porém como no caso anterior existe uma limitação, principalmente no que se refere ao tamanho da tela que é bastante limitada.

Desafios para o desenvolvimento sem fio

Transmissão de Erros - mensagens enviadas ou recebidas numa conexão sem fio estão sujeitas a interferência e à demora, o que pode alterar seu conteúdo.

Latência - tempo que uma mensagem leva até chegar ao seu destino.

Segurança - Quaisquer informações transmitidas numa rede sem fio podem ser interceptadas. Para solucionar este problema deve-se introduzir métodos de segurança nas duas partes, no cliente e no servidor.

Diretrizes para o Desenvolvimento de Aplicações Móveis

Algumas diretrizes úteis no processo de desenvolvimento de aplicações para dispositivos móveis.

Ambiente: pesquisar o ambiente em que a aplicação será utilizada, antes de começar o desenvolvimento da aplicação. É importante saber quais são as necessidades dos usuários, os requisitos impostos pelas redes e em que plataformas o aplicativo será utilizado;

Dividir as Tarefas da Aplicação: deve-se pensar com cuidado quais funções oferecidas pelo sistema serão realizadas pelo servidor e quais podem ou deverão ser realizadas pelo dispositivo móvel. Este é um fator importante, pois se refere a duas limitações que ocorrem em dispositivos móveis, a baixa capacidade de processamento, o que poderia ser resolvido realizando a maior parte do processamento no servidor e a latência que poderia ser resolvida mantendo os dados no dispositivo móvel;

Representação dos dados: existem muitas maneiras de se representar dados, umas ocupam mais memória do que outras. Deve-se escolher aquelas que exigirem menor quantidade de memória para seu armazenamento. Por exemplo, números normalmente são muito mais compactos na forma binária do que na forma de string;

Latência da Mensagem: Em alguns casos, pode ser possível realizar outras tarefas enquanto uma mensagem está sendo processada. Se o tempo utilizado para realização da transmissão for considerável, é importante informar ao usuário a situação do progresso da mesma.

Simplicidade da Interface: fazer a interface do aplicativo simples e intuitiva, de forma que o usuário raramente precise consultar quaisquer manuais. Para tanto é necessário reduzir a quantidade de informações exibidas no dispositivo; apresentando-as na sequência de entrada; oferecendo sempre que possível listas de seleção.

Execução : pois quanto menor o tempo de execução, menor o tempo de utilização da bateria, obtendo maior satisfação do usuário.

Agora serão apresentadas algumas técnicas de programação para aumentar a performance de aplicações J2ME.

- Sempre que possível utilizar variáveis locais ao invés de variáveis de classes, assim o acesso aos atributos de objeto é mais rápido com o uso das variáveis locais;
- Tentar diminuir as chamadas de métodos, pois quando um método é chamado, a máquina virtual java aloca um novo nodo da pilha de execução, isso faz com que a memória seja sobrecarregada;

- Minimizar a criação de objetos, pois quando um objeto é criado, na maioria das vezes ele será destruído, isso leva à uma diminuição da performance da aplicação. Para evitar a criação de muitos objetos deve se utilizar objetos que possam ser “reciclados” ou mantidos na memória o máximo de tempo possível;
- Tentar não concatenar strings, a concatenação com o operador + leva a criação de um novo objeto e por consequência utiliza mais a memória e o processamento;
- Evitar a sincronização, se um método demora algumas frações de segundos para executar, deve-se colocar a sua chamada em uma thread em separado.

Componentes de Redes Sem fio

Aqui estão alguns exemplos de equipamentos comumente utilizados para a comunicação de uma rede utilizando o meio de transporte de dados e voz através das ondas de rádio, conhecido como redes Wireless.

A figura abaixo exemplifica um Ponto de Acesso, onde equipamentos como computadores, Palm Tops, notebooks e outros, podem se conectar a ele para ter acesso a uma rede local, sendo ela cabeada ou não.



Figura 1 - Ponto de Acesso

A figura seguinte mostra uma placa PCI (Peripheral Component Interconnect - Interconector de Componentes Periféricos) dotada de um chip

rádio que normalmente é conectada internamente em um computador pessoal para comunicação com outros computadores em uma rede Ad Hoc ou a um Ponto de Acesso, para acesso a uma rede sem fio, ou até mesmo cabeada.



Figura 2 - Placa PCI WLAN.

Na figura seguinte é mostrada um exemplo de uma rede WPAN, onde vários dispositivos de rede se comunicam através da tecnologia Bluetooth.



Figura 3 - Dispositivos Bluetooth.

Nestas próximas figuras são apresentados dois exemplos de antenas externas que são utilizadas para comunicação a maiores distâncias. A figura da esquerda é um exemplo de antena omni-direcional que propaga seu sinal por todas as direções. A figura da direita é uma antena direcional, que como o

próprio nome já diz, propaga seu sinal em apenas uma direção, utilizada comumente para fechar a área de captação do sinal, diminuindo assim as chances de interceptações do mesmo.



Figura 4 - Antena omni direcional



Figura 5 - Antena direcional

Arquitetura do Sistema Operacional Android, Frameworks e Bibliotecas

Objetivo:

Apresentação da Arquitetura do sistema operacional android, demonstrando seus principais componentes, seus frameworks, suas bibliotecas e concluindo com uma breve visualização do android market.

Arquitetura Android

A estrutura do sistema operacional Android esta subdivida em camadas, onde cada camadas se responsabiliza pelo gerenciamento dos seus processos a ela ligados. Como apresentado abaixo:

Camada de Aplicações, parte responsável pelo armazenamento das aplicações que são executados pelo sistema operacional, como por exemplo, mapas, cliente de SMS e MMS, navegador, cliente de e-mail, calculadora, e outros, que incluem as aplicações desenvolvidas por todos.

Camada de Bibliotecas, camada onde se encontram as bibliotecas C/C++ que são utilizadas pelo sistema, bibliotecas de multimídia, funções de visualização 2D e 3D, funções para navegadores web, funções de aceleradores de hardware, preenchimento 3D, funções para gráficos 2D, fontes bitmap e vetorizadas e funções de acesso ao banco de dados SQLite.

Camada de Runtime, nessa camada esta localizada a instancia a máquina virtual Dalvik, sendo que, cada aplicação em execução no Android possui uma copia da mesma. A maquina virtual Dalvik constitui a melhor referente para desempenho, pois tem a maior integração com as novas gerações de hardware, tendo sido projetada para executar vários processos paralelamente.

Camada de Kernel Linux constitui o núcleo do sistema operacional Android, e por ser derivada do kernel 2.6 do Linux, herdou diversas características dessa plataforma. O que segundo seus elaboradores é parte mais importante do sistema operacional Android, pois ela é responsável pelo controle de processos, gerenciarem memória, threads, protocolos de rede, modelo de drives e a segurança dos arquivos.

Framework Android

O Android SDK é composto de um grupo de módulos que podem ser baixados conforme a necessidade do desenvolvedor utilizando-se o Android SDK Gestor, por isso quando partes do SDK são atualizadas ou uma nova versão do Android é lançado, podemos usar o Gerenciador de SDK para baixá-los.

Existem vários pacotes de instalação do Android SDK. Abaixo estão descritos alguns dos pacotes disponíveis:

Pacote	Descrição
SDK Tools	Contém ferramentas para depuração e testes.
SDK Platform-tools	Contém ferramentas para desenvolvimento e depuração de sua aplicação.
Documentation	Uma cópia offline da documentação mais Recente do SDK.
SDK Platform	Especifica qual a versão do SDK utilizada.
System Images	Uma ou mais imagens diferentes do sistema.
Sources for Android SDK	Cópia do código-fonte da plataforma Android.

Google APIs

Fornece uma plataforma pode ser usada para desenvolver um aplicativo usando as APIs do Google.

Dispositivos virtuais Android

Dispositivos virtuais Android (AVDs) utilizam o emulador QEMU como base e são capazes de emular o hardware de um dispositivo Android, além de imagens do sistema Android, que são softwares construídos para serem executados no hardware emulado. AVDs são configurados pelo gerenciador do AVD e do SDK, que define parâmetros como o tamanho dos dispositivos de armazenamento emulados e as dimensões de tela, e que permite que você especifique qual imagem do sistema Android será utilizada em cada dispositivo emulado.

AVDs permitem que você teste seu software em um escopo mais amplo de características de sistema do que o escopo que você provavelmente conseguiria adquirir e testar em dispositivos físicos. Uma vez que tanto os emuladores de hardware com base em QEMU quanto as imagens do sistema e os parâmetros dos AVDs são componentes intercambiáveis, você pode testar dispositivos e imagens do sistema mesmo antes que haja hardware disponível para executá-los.

Outras ferramentas do SDK

Além das principais ferramentas que você utilizará, normalmente, na maioria dos desenvolvimentos, há diversas outras ferramentas no SDK. Aquelas utilizadas ou invocadas diretamente pelos desenvolvedores. Algumas delas estão listadas abaixo:

Ferramenta

Descrição

Dmtracedump

Gera gráficos de rastreamento dos arquivos de log.

Hierarchyviewer	Permite depurar e otimizar a interface da Aplicação Android do usuário.
hprof-conv	Converte o arquivo hprof que é gerado pelas ferramentas de Android SDK para um formato padrão
layoutopt	Permite analisar rapidamente layouts de sua aplicação, a fim de otimizá-los para a eficiência.
Mksdcard	Simular a presença de um cartão de memória externo (como um cartão SD).
Systrace	Permite analisar a execução de sua aplicação no contexto dos processos do sistema.
Sqlite3	Permite acessar os arquivos de dados SQLite criados e utilizados por aplicações Android.
Zipalign	Otimiza arquivos .apk.

Para maiores informações consulte o site <http://developer.android.com/guide/developing/tools/index.html>.

Bibliotecas Nativas

Na camada diretamente acima do kernel estão as bibliotecas nativas do Android. Essas bibliotecas compartilhadas são todas escritas em C ou C ++, compiladas para a arquitetura de hardware específica utilizada pelo telefone, e pré-instalado pelo fabricante do telefone.

Algumas das bibliotecas nativas mais importantes incluem o seguinte:

- Surface Manager: O Android usa um gerenciador de janelas similar ao Vista e ao Compiz, só que bem mais simples.

- Gráficos 2D e 3D: Elementos bi e tridimensionais podem ser combinado em uma única interface. A biblioteca utilizará hardware 3D se o dispositivo o tiver ou utilizará software para visualizar os elementos tridimensionais.

- Codecs de mídia: Pode reproduzir vídeo, gravar e reproduzir áudio em uma variedade de formatos incluindo AAC, AVC (H.264), H.263, MP3 e MPEG-4.

- Banco de dados SQL: Inclui o banco de dados SQLite com estrutura mais leve, o mesmo banco de dados usado no Firefox e no Apple iPhone. Que pode ser usado para armazenamento persistente numa aplicação.

- Browser engine: Para uma rápida exibição de conteúdo HTML, o Android usa a biblioteca WebKit . Esta é a mesma estrutura usada no navegador Google Chrome, no navegador Safari da Apple, no iPhone e na plataforma S60 da Nokia.

Essas bibliotecas não podem ser consideradas aplicações individuais. Elas existem apenas para serem chamadas por programas de nível superior. Sendo que desde o Android 1.5, você pode escrever e implementar suas próprias bibliotecas nativas usando o Native Development Toolkit (NDK).

Android Market

O Android Market é o mercado de aplicativos do sistema operacional do Google, onde estão disponíveis jogos, papéis de parede e todas as demais aplicações - gratuitas e pagas - para a plataforma.

Para acessar o Market, primeiro você precisa logar com uma conta do Google. Desta forma, sempre que você acessar sua conta, em qualquer dispositivo, poderá ter acesso a todos os seus aplicativos, até os que foram comprados com outro aparelho.

Para saber mais acesse o endereço do android market :
<https://play.google.com/store>.

Ambiente de Programação IDE Eclipse, Ferramentas e Testes do Aplicativo Móvel

Componentes do Eclipse

O Eclipse prove um ambiente que permite a criação de tipos específicos de projetos, o que inclui projetos do tipo java. Um plug-in do tipo ADT adiciona ao eclipse a habilidade de criar e utilizar projetos do tipo Android. Deste modo, ao criar um novo projeto Android, o ADT adiciona a hierarquia de arquivos no projeto e todos os arquivos necessários para um projeto Android.

O ADT tem diversos componentes que podem ser utilizados separadamente, apesar de um “plug-in” dar a ideia de ser somente uma pequena modificação, o ADT é composto de uma quantidade considerável gama de funções. Para entendermos melhor a estrutura e o funcionamento do ADT, abaixo serão apresentadas as funcionalidades de cada uma de suas partes e como elas serão utilizadas no eclipse no desenvolvimento de aplicações para Android.

Android Layout Editor

A criação de layouts para interfaces de usuário em aplicativos Android podem ser feitos utilizando-se o XML. Com o ADT será adiciona um editor que proporcionara uma visualização do layout proposto o que auxiliara na composição e da aplicação para o Android. Ao abrir um arquivo de layout, o ADT iniciara automaticamente o editor que comporá a visualização do arquivo.

Nas versões anteriores do SDK do Android, o editor de layout possuía limitações quando se utilizava esta funcionalidade. Atualmente, entretanto, deve-se considerar a ferramenta de visualização de layouts para o Android como sendo a melhor maneira de agilizar o desenvolvimento de uma aplicação. Sendo que com a automação da especificação dos layouts aumenta significativamente a possibilidade de que os layouts gerados funcionem na maior parte dos dispositivos Android.

Android Manifest Editor

Todos os projetos Android devem possuir um arquivo de manifesto, que acompanha a aplicação e os recursos que estarão contidos no projeto quando ele é desenvolvido e que informa ao Android como instalar e utilizar o software. Este arquivo de manifesto está em XML, e um editor XML especializado é fornecido pelo ADT com a finalidade de auxiliar desenvolvedores nesta tarefa.

Editores XML para outros arquivos XML do Android

Existem outros arquivos XML em um projeto android que contem informações, como montagem de menus, recursos de strings, ou que armazenam recursos gráficos para o aplicativo, existem editores específicos que inicializam sempre que estes tipos de arquivos forem abertos.

Compilação de aplicativos Android

Sempre que solicitamos a execução de um Projeto no Eclipse ele é compilado automaticamente. Em razão disso não será possível encontrar um local individual que possua a função de transformar o código-fonte e seus respectivos recursos em um programa executável ou até mesmo um instalador. Ao invés disso o Android necessita de passos específicos para gerar a compilação de um arquivo que possa ser instalado em um emulador ou dispositivo móvel, sendo que o ADT fornece o software responsável pela execução desses passos. Como resultado de uma compilação de um projeto android será gerado um arquivo do tipo .apk, que se encontrará na subpasta bin, na hierarquia de arquivos do projeto na sua área de trabalho do Eclipse.

Compiladores específicos do Android, são fornecidos com o ADT, permitindo que se utilize Java como a linguagem para geração de aplicativos para o Android, ao mesmo tempo executa esses aplicativos numa máquina virtual Dalvik que em contrapartida executa seus próprios bytecodes.

Execução e depuração de aplicativos Android

Quando se executa ou se depura uma aplicação Android no Eclipse, o arquivo .apk dessa aplicação será invocado e inicializado em um AVD ou dispositivo Android. Durante esse processo, será utilizado o ADB e o DDMS para comunicação com o AVD ou o dispositivo, e também com o ambiente de execução da máquina virtual Dalvik que executa o código da aplicação. O ADT adiciona um conjunto de funções que permitirão ao Eclipse realizar essa execução.

Gerenciador de AVD e SDK

Um QEMU é um emulador de propósito geral, enquanto que o SDK do Android permite que se configure o QEMU, o que constitui uma funcionalidade interessante quando se cria emuladores que tem por objetivo executar aplicações para Android. O gerenciador do AVD e do SDK fornecem a interface do usuário permitindo obter o controle do dispositivo virtual com base no QEMU.

Hierarchy Viewer

O visualizador de hierarquias mostra, permitindo que se analise a hierarquia da atividade atual, ou de um dispositivo selecionado. Isso permite que se encontre e diagnostique problemas na hierarquia de suas visualizações, mesmo quando o aplicativo está sendo executado. Também permite que se visualize uma apresentação ampliada da tela da aplicação, com orientações de alinhamento que permitem identificar problemas em layouts. Para informações mais detalhadas sobre o Hierarchy Viewer acesse o endereço:

<http://developer.android.com/guide/developing/tools/hierarchy-viewer.html>.

Layoutopt

Layoutopt é uma ferramenta que analisa os arquivos de layout XML e que pode diagnosticar problemas nos layouts do Android.

Para informações detalhadas sobre o Layoutopt acesse o endereço: <http://developer.android.com/guide/developing/tools/layoutopt.html>.

Monkey

Monkey é um programa que pode ser executado tanto em um emulador, como em um dispositivo e gerando um fluxo pseudo-aleatório de eventos do usuário, como cliques, toques, ou gestos, assim como uma série de eventos a nível de sistema. Você pode usar o monkey para realizar testes de estresse em aplicativos que estão em desenvolvimento, de uma maneira aleatória ainda repetível.

Visão global

O monkey inclui uma série de opções que se dividem em quatro categorias principais:

Opções de configuração básica, como a definição do número de eventos que devem ser testados, e a quantidade de tentativas para cada evento.

Constrangimentos operacionais, tais como a restrição do teste a um único pacote.

Tipos de eventos e frequências.

Opções de depuração.

Quando o monkey é executado, serão gerados eventos que serão enviados ao sistema. Feito isso, ele observará o comportamento de três condições do sistema que estiver em teste:

Se o monkey restrito estiver o monkey para ser executado em uma ou mais aplicações específicas, ele verificará especialmente as tentativas de navegar entre as aplicações, e seus respectivos bloqueios.

Caso a aplicação produza algum tipo de falha ou receba qualquer tipo de exceção não tratada, o monkey parará e reportará a falha ou o erro encontrado.

Dependendo do nível de detalhamento selecionado, será mostrado relatórios sobre o progresso do monkey e os eventos que foram gerados e testados por ele.

Uso Básico do Monkey

Você pode iniciar o monkey usando uma linha de comando na máquina de desenvolvimento ou um script. Porque o monkey pode ser executado no emulador ou no dispositivo móvel, deve-se inicializá-lo a partir de um shell no local onde se deseja executá-lo.

A sintaxe básica de inicialização é:

```
$ Adb shell monkey [options] <event-count>[ opções ] < evento - contagem >
```

Sem nenhuma opção específica, o monkey vai iniciar em modo não detalhado, e irá enviar eventos para todas as aplicações instaladas no dispositivo onde estiver instalado. Aqui está uma linha de comando mais típica, que iniciará o pedido, e enviar 500 eventos pseudo-aleatórios a uma aplicação:

```
$ Adb shell monkey-p your.package.name-v 500- p seu . pacote . nome - v 500
```

Informações detalhadas sobre o monkey podem ser encontradas no endereço:
<http://developer.android.com/guide/developing/tools/monkey.html>.

Componentes do Android: Atividades, Serviços, Repositórios de Conteúdo, Mensagens e Arquivo de Manifesto

Activity

Uma atividade é um componente de aplicação que fornece uma tela com a qual os usuários podem interagir, a fim de fazer alguma coisa, como discar o telefone, tirar uma foto, enviar um e-mail, ou visualizar um mapa. Cada atividade é dada uma janela na qual foi desenhada sua interface de usuário. A janela tipicamente preenche a tela, mas pode ser menor do que a tela e flutuam no topo de outras janelas.

Uma aplicação geralmente consiste de múltiplas atividades, que estão frouxamente ligadas entre si. Tipicamente, uma atividade de uma aplicação é especificada como a atividade "principal", que é apresentada ao utilizador, quando ocorre o lançamento do pedido de execução da aplicação pela primeira vez. Cada atividade pode, então, iniciar outra atividade, a fim de executar ações diferentes. Cada vez que uma nova atividade começa, a atividade anterior será parada, mas o sistema preserva a atividade em uma pilha (a "pilha de volta"). Quando uma nova atividade é iniciada, ela é empurrada para a pilha de volta e leva o foco do usuário. Esta é uma pilha que usa o mecanismo de retorno básico "último a entrar, primeiro a sair da pilha", assim, quando o usuário está na atividade atual e pressiona o botão Voltar, é exibida a atividade que está registrada como a última anterior a atual, e a atual é destruída.

Quando uma atividade é interrompida porque uma nova atividade é iniciada, o sistema é notificado dessa mudança de estado através de métodos da atividade especificados no chamado do ciclo de vida da atividade. Existem vários métodos de retorno de que uma atividade pode receber, devido a uma mudança que ocorre no estado do sistema ao criar, parar, retornar ou destruir, e cada callback fornece-lhe a oportunidade de realizar um trabalho específico que é apropriado a cada mudança de estado. Por exemplo, quando parada,

sua atividade deve liberar todos os objetos grandes, como conexões de rede ou banco de dados. Quando a atividade recomeça, pode-se readquirir os recursos necessários e retomar ações que foram interrompidos. Estas transições de estado fazem parte do ciclo de vida de atividade.

Criando uma Atividade

Para criar uma atividade, deve-se criar uma subclasse da atividade (ou uma subclasse existente da mesma). Nesta subclasse, precisamos implementar métodos de retorno de chamada, que chama o sistema quando as transições de atividade entre vários estados de seu ciclo de vida, como quando a atividade está sendo criada, parada, recomeçada, ou destruída. Os dois métodos mais importantes são:

onCreate()

Deve se implementar este método. O sistema chama este ao criar a sua atividade. Dentro de sua implementação, você deve inicializar os componentes essenciais de sua atividade. Mais importante, este é o lugar onde você deve chamar `setContentView()` para definir o layout para a interface da atividade do usuário.

onPause()

O sistema chama este método como a primeira indicação de que o usuário está deixando sua atividade (embora nem sempre signifique que a atividade está sendo destruída). Este é geralmente o lugar onde deve-se comprometer todas as mudanças que devem ser mantidos para além da sessão atual do usuário (porque o usuário não pode voltar).

Existem vários métodos no ciclo de vida, alguns deles fornecem o retorno da chamada, que devemos usar a fim de proporcionar uma experiência ao usuário de movimento entre as atividades e lidar com interrupções

inesperadas que forçam sua atividade a ser interrompido e até mesmo destruído.

Implementação de uma interface de usuário

A interface de usuário para uma atividade é prestada por uma hierarquia de objetos View derivados da classe View. Cada view controla um determinado espaço retangular dentro da janela da atividade e pode responder à interação do usuário. Por exemplo, uma view pode ser um botão que inicia uma ação quando o usuário toca.

Android fornece um número de views prontas que pode-se usar para criar e organizar um layout. "Widgets" são elementos para as views que proporcionam estruturas visuais (e interativas) para a tela, como um botão, campo de texto, caixa de seleção, ou apenas uma imagem. "Layouts" são vistas derivadas de ViewGroup que fornecem um modelo de layout único para as opiniões derivas, como um layout linear, um layout de grade, ou a disposição relativa. Você também pode criar uma subclasse view e classes ViewGroup (ou subclasses existentes) para seus próprios widgets e layouts e aplicá-las no layout da atividade.

A maneira mais comum para definir um layout usando pontos de vista é com um arquivo de layout XML salvo em recursos de sua aplicação. Dessa forma, você pode manter o design de sua interface de usuário separadamente do código fonte que define o comportamento da atividade. Você pode definir o layout como a interface do usuário para a sua actividade com setContentView(), passando o ID do recurso para o esquema. No entanto, você também pode criar novas visualizações em seu código de atividade e construir uma hierarquia de vista através da inserção de novas Views em um ViewGroup, em seguida, usar esse layout, passando a ViewGroup raiz para setContentView ().

Declarando a atividade no manifesto

Você deve declarar a sua atividade no arquivo de manifesto para que ele seja acessível para o sistema. Para declarar a sua atividade, abra o seu arquivo e adicionar um elemento `<activity>` como um filho do elemento `<application>`. Por exemplo:

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

Há vários outros atributos que você pode incluir neste elemento, para definir propriedades como o rótulo para a atividade, um ícone para a atividade, ou um tema ao estilo da UI da atividade. O atributo `android:name` é o único atributo necessário ele especifica o nome da classe da atividade. Uma vez que você publicar seu aplicativo, você não deve alterar este nome, porque se você fizer isso, você pode quebrar algumas funcionalidades, como atalhos de aplicativos.

O uso de filtros para intent

Um elemento `<activity>` também pode especificar vários filtros para intent utilizando o elemento `<intent-filter>`, a fim de declarar como outros componentes de aplicação pode ativá-la.

Quando você cria um novo aplicativo usando as ferramentas do Android SDK, a parte da atividade que você cria automaticamente inclui um filtro de intents que declara a atividade responde para a ação "principal" e deve ser colocado na categoria "launcher". O filtro de intent é parecido com o exemplo a baixo:


```
<activity android:name=".ExampleActivity"
          android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

O elemento `<action>` especifica que este é o ponto de entrada "principal" para a aplicação. O elemento `<category>` especifica que esta atividade deve ser listada no launcher do sistema (para permitir aos usuários a visualização desta atividade).

Se você pretende que o seu aplicativo seja auto-suficiente e não permitir que outras aplicações utilizem suas atividades, então você não precisa de quaisquer outros filtros de intent. Apenas uma actividade deve ter a ação "principal" e categoria "launcher", tal como no exemplo anterior. Atividades que você não deseja disponibilizar para outras aplicações não devem ter filtros e você pode iniciá-los você mesmo usando intents explícitas que serão explicadas posteriormente.

No entanto, se você quiser que sua atividade possa responder às intents implícitas que são entregues a partir de outras aplicações, então você deve definir filtros intenção adicionais para a sua atividade. Para cada tipo de intent para a qual você deseja responder, você deve incluir um `<intent-filter>` que inclui um elemento `<action>` e, opcionalmente, um elemento `<category>` e/ou um elemento `<data>`. Estes elementos especificam o tipo de intent para o qual a sua atividade pode responder.

Iniciar uma Atividade

Você pode começar outra atividade chamando `startActivity()`, passando uma Intent que descreve a atividade que você deseja iniciar. A intent especifica tanto a exata atividade que você deseja iniciar ou descreve o tipo de ação que

you want to perform (the system selects the appropriate activity for you, which can be even from a different application). An intent can also transport small amounts of data to be used by the activity that is initiated.

When you work within your own application, many times you will simply need to launch a known activity. You can do this through the creation of an intent that explicitly defines the activity that you want to start, using the class name. For example, here is how one activity starts another activity by calling `SignInActivity`:

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

However, the application can also want to execute some action, such as sending an e-mail, text message, or status update, using data from its activity. In this case, the application cannot perform its own activities to perform these actions, so you can take advantage of the activities provided by other applications on the device, which can execute the actions for you. This is the place where intents are really useful; you can create an intent that describes an action that you want to execute for the system to launch the appropriate activity from another application. If there are multiple activities that can handle the intent, then the user can select which one to use. For example, if you want to allow the user to send an e-mail message, you can create the following intent:

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(Intent.EXTRA_EMAIL, recipientArray);
startActivity(intent);
```

The extra `EXTRA_EMAIL` added to the intent is a string of e-mail addresses for which the e-mail should be sent. When an e-mail application responds to this intent, it reads the string of characters provided in the extra and puts them in the "to" field of the composition form

de e-mail. Nesta situação, a atividade do aplicativo de e-mail começa e quando o usuário termina sua atividade recomeça.

Iniciando uma atividade para um resultado

Às vezes, você pode querer receber um resultado da atividade que você começou. Nesse caso, inicie a atividade chamando `startActivityForResult()` (em vez de `startActivity()`). Para, então, receber o resultado da atividade subsequente, implementando o método de retorno `onActivityResult()`. Quando a atividade subsequente é feita, ele retorna um resultado de uma intent de seu método `onActivityResult()`.

Por exemplo, talvez você queira que o usuário escolha um de seus contatos, para que a sua atividade possa fazer alguma coisa com a informação do contato. Veja como você pode criar uma intent e manipular o resultado:

```
private void pickContact() {  
    // Create an intent to "pick" a contact, as defined by the content provider URI  
    Intent intent = new Intent(Intent.ACTION_PICK, Contacts.CONTENT_URI);  
    startActivityForResult(intent, PICK_CONTACT_REQUEST);  
}  
  
@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    // If the request went well (OK) and the request was  
    PICK_CONTACT_REQUEST  
    if (resultCode == Activity.RESULT_OK && requestCode ==  
        PICK_CONTACT_REQUEST) {  
        // Perform a query to the contact's content provider for the contact's name  
        Cursor cursor = getContentResolver().query(data.getData(),  
            new String[] {Contacts.DISPLAY_NAME}, null, null, null);  
        if (cursor.moveToFirst()) { // True if the cursor is not empty  
            int columnIndex = cursor.getColumnIndex(Contacts.DISPLAY_NAME);  
            String name = cursor.getString(columnIndex);
```

```
        // Do something with the selected contact's name...
    }
}
}
```

Este exemplo mostra a lógica básica que você deve usar com o método `onActivityResult()` a fim de lidar com um resultado atividade. As verificações de condição, primeiro se o pedido foi bem sucedido, se foi, em seguida, o `resultCode` será `RESULT_OK` e se o resultado do pedido é uma resposta conhecida, neste caso, o `requestCode` corresponde ao segundo parâmetro enviado com `startActivityForResult()`. De lá, o código manipula o resultado da atividade, consultando os dados retornados em uma `Intent` (o parâmetro de dados).

O que acontece é, um `ContentResolver` executa uma consulta em um provedor de conteúdo, que retorna um cursor que permite que os dados consultados possam ser lidos.

Encerrar uma Atividade

Você pode desligar uma atividade chamando o método `finish()`. Você também pode desligar uma atividade separada que você já começou chamando `finishActivity()`.

Nota: Na maioria dos casos, você não deve terminar explicitamente uma atividade usando esses métodos. Como discutido na seção seguinte sobre o ciclo de vida de atividade, o sistema Android gerencia a vida de uma atividade para você, para que você não precisa para terminar as suas atividades. Chamar esses métodos podem afetar negativamente a experiência do usuário e só deve ser usado quando você absolutamente não quer que o usuário retorne a esta instância da atividade.

Serviços

Um serviço é um componente da aplicação que pode realizar operações de longa duração em modo background e não prove interface de usuário. Um componente da aplicação pode iniciar o serviço e ele vai continuar rodando em background mesmo que o usuário mude para outra aplicação. Adicionalmente, um componente pode fazer uma ligação com um serviço para interagir com ele e inclusive realizar processos de comunicação entre processos. Por exemplo, um serviço pode querer manusear transações de rede, tocar música, fazer uma troca de arquivo ou interagir com um provedor de conteúdo, tudo em modo background.

Um serviço essencialmente tem duas formas:

Started

Um serviço está started quando um componente de aplicação inicia-o chamando o método `startService()`. Uma vez iniciado, o serviço pode rodar em modo background indefinidamente, mesmo que o componente que o iniciou seja destruído. Usualmente, um serviço iniciado realiza uma tarefa única e não retornar o resultado para quem o chamou. Por exemplo, pode-se fazer o download ou upload de arquivos em uma rede. Quando a operação terminar, o serviço termina sem a necessidade de interação do usuário.

Bound

Um serviço está bound quando um componente de aplicação faz uma ligação com ele chamando o método `bindService()`. Um serviço bound oferece uma interface cliente servidor que permite ao componente interagir com o serviço, enviar requisições, obter resultados e mesmo fazer alguns processos com comunicação entre processos. Um serviço bound roda apenas enquanto o componente da aplicação estiver ligado a ele. Múltiplos componentes podem ser ligados a um serviço ao mesmo tempo, mas quando todos eles se desligarem dele, o serviço será destruído.

Embora esta documentação geralmente discuta esses dois tipos de serviços separadamente, o serviço pode funcionar em ambos os sentidos, ele pode ser iniciado (para rodar indefinidamente) e também permitir a ligação. É simplesmente uma questão de se implementar ambos os métodos de retorno: `onStartCommand()` para permitir que os componentes o inicie e `onBind()` para permitir a ligação com o componente.

Independentemente do fato de que a sua aplicação é iniciada, ligada, ou ambos, qualquer componente da aplicação pode usar um serviço (mesmo a partir de uma aplicação em separado), da mesma maneira que qualquer componente pode utilizar uma atividade pode iniciar uma intent. No entanto, você pode declarar um serviço como privado, no arquivo de manifesto, e bloquear o acesso de outros aplicativos.

Cuidado: Um serviço é executado na thread principal de sua aplicação, o serviço não cria sua própria thread e não é executado em um processo separado (a menos que você especifique o contrário). Isto significa que, se o serviço vai fazer qualquer trabalho intensivo na CPU ou operações de bloqueio (como a reprodução de MP3 ou de rede), você deve criar uma nova thread dentro do serviço para fazer esse trabalho. Ao usar uma thread separada, você vai reduzir o risco do aplicativo não responder erros e linha principal do aplicativo pode permanecer dedicado à interação do usuário com suas atividades.

O Básico

Para criar um serviço, você deve criar uma subclasse de `Serviço` (ou uma de suas subclasses existentes). Em sua implementação, é preciso substituir alguns métodos de retorno de chamada que lidam com os principais aspectos do ciclo de vida do serviço e fornecer um mecanismo para componentes para ligar o serviço, se for o caso. Os métodos de retorno mais importantes que você deve substituir são:

onStartCommand()

O sistema chama este método quando outro componente, como uma atividade, inicia a solicitação do serviço, chamando `StartService()`. Uma vez que este método é executado, o serviço é iniciado e pode ser executado em segundo plano por tempo indeterminado. Se você implementar dessa forma, é sua a responsabilidade parar o serviço quando o seu trabalho terminar, chamando `stopSelf()` ou `StopService()`.

onBind()

O sistema chama este método quando se quer vincular um componente com o serviço (como para executar RPC), chamando `bindService()`. Em sua implementação deste método, deve-se fornecer uma interface que os clientes usam para se comunicar com o serviço, retornando um `IBinder`. Você sempre deve implementar esse método, mas se você não quer permitir a ligação, então você deve retornar nulo.

onCreate()

O sistema chama este método quando o serviço é criado pela primeira vez, para realizar os procedimentos de configuração (antes de chamar `onStartCommand()` ou `onBind()`). Se o serviço já está em execução, este método não é chamado.

onDestroy()

O sistema chama este método quando o serviço não é mais utilizado e está sendo destruído. Seu serviço deve implementar este método para limpar quaisquer recursos como threads, listeners registrados, receptores, etc Esta é a última chamada o serviço recebe.

Se um componente inicia o serviço ligando `StartService()` (o que resulta em uma chamada para `onStartCommand()`), em seguida, o serviço continua

executando até que o componente pare com `stopSelf()` ou outro componente para-lo chamando `StopService()`.

Se um componente chama `bindService()` para criar o serviço (e `onStartCommand()` não é chamado), em seguida, o serviço é executado apenas enquanto o componente está ligado a ele. Uma vez que o serviço é desacoplado de todos os clientes, o sistema o destrói.

O sistema Android irá forçar a parada de um serviço apenas quando a memória estiver baixa e precisar recuperar os recursos do sistema para a atividade que tem o foco do usuário. Se o serviço está vinculado a uma atividade que tem o foco de usuário, então é menos provável de ser parada, e se o serviço é declarado para ser executado em primeiro plano, então ele quase nunca vai ser parado. Caso contrário, se o serviço foi iniciado e é de longa duração, então o sistema irá reduzir a sua posição na lista de tarefas de fundo ao longo do tempo e o serviço irá tornar-se altamente suscetível à parada, se o serviço é iniciado, então você deve projetá-lo para elegantemente reiniciado pelo sistema. Se o sistema de para seu serviço, ele pode reinicia-lo assim que os recursos estiverem disponíveis novamente (embora isso também dependa do valor que você retornar de `onStartCommand()`),

Declarando um serviço no manifesto

Como atividades (e outros componentes), você deve declarar todos os serviços no arquivo de manifesto do aplicativo.

Para declarar seu serviço, adicionar um elemento `<service>` como um subordinado do elemento `<application>`. Por exemplo:

```
<manifest ... >
...
<application ... >
    <service android:name=".ExampleService" />
...
```



```
</application>  
</manifest>
```

Existem outros atributos que você pode incluir no elemento `<service>` para definir propriedades, tais como permissões necessárias para iniciar o serviço e o processo em que o serviço deve ser executado. O atributo `android:name` é o único necessário, especifica o nome da classe do serviço. Uma vez que você publicar seu aplicativo, você não deve alterar este nome, porque se você fizer isso, você pode quebrar alguma funcionalidade onde as intents explícitas são usadas para referenciar o seu serviço.

Assim como uma atividade, um serviço pode definir filtros que permitem a intent de outros componentes para chamar o serviço usando as intents implícitas. Ao declarar filtros de intents, componentes de qualquer aplicativo instalado no dispositivo do usuário pode potencialmente iniciar o seu serviço se o serviço declara um filtro de intent que coincide com a intent outro aplicativo pode passar `StartService()`.

Se você planeja usar o seu serviço apenas localmente (as outras aplicações não usá-lo), então você não precisa (e não deve) fornecer todos os filtros intents. Sem filtros de intents, você deve iniciar o serviço usando uma intent que, explicitamente, nomeia a classe do serviço.

Além disso, você pode garantir que seu serviço é privado para a sua aplicação somente se você incluir o atributo `android:exported` e defini-lo como "falso". Isto é eficaz mesmo se o seu serviço fornece filtros de intents.

Criação a inicialização de um serviço

A inicialização de um serviço é aquilo que outro componente começa ligando `StartService()`, o que resulta em uma chamada de método do serviço `onStartCommand()`.

Quando um serviço é iniciado, tem um ciclo de vida que é independente do componente que começou o serviço, pode ser executado em segundo plano indefinidamente, mesmo que o componente que começou seja destruído. Como tal, o serviço deve parar quando o seu trabalho é concluído chamando `stopSelf()`, ou outro componente pode pará-lo chamando `StopService()`.

Um componente da aplicação, como uma atividade pode iniciar o serviço chamando `StartService()` e passando uma intent que especifica o serviço e inclui todos os dados que o serviço deve usar. O serviço recebe esta Intent no método `onStartCommand()`.

Por exemplo, suponha que uma atividade precisa salvar alguns dados em um banco de dados online. A atividade pode iniciar um serviço de acompanhamento e usa-lo para salvar os dados, passando a intent para um `StartService()`. O serviço recebe a intent em `onStartCommand()`, conecta-se à Internet e executa a operação de banco de dados. Quando a transação é feita, o serviço para e é destruído.

Atenção: um serviço é executado no mesmo processo que o aplicativo no qual ele é declarado e na thread principal da aplicação, por padrão. Assim, se seu serviço executa operações intensivas ou bloqueio enquanto o usuário interage com uma atividade a partir do mesmo aplicativo, o serviço vai diminuir o desempenho de atividade. Para não afetar o desempenho do aplicativo, você deve começar uma nova thread dentro do serviço.

Tradicionalmente, há duas classes que você pode estender para criar um serviço iniciado:

Service

Esta é a classe base para todos os serviços. Quando você estender essa classe, é importante que você crie uma nova thread que deve fazer todo o trabalho do serviço, porque o serviço utiliza thread principal do aplicativo, por

padrão, o que pode reduzir o desempenho de qualquer atividade de sua aplicação está rodando.

IntentService

Esta é uma subclasse de serviço que utiliza um segmento de trabalho para lidar com todos os pedidos de início, um de cada vez. Esta é a melhor opção se você não exigir que o seu serviço lide com múltiplas solicitações simultaneamente. Tudo que você precisa fazer é implementar `onHandleIntent()`, que recebe a intent de cada solicitação de início para que você possa fazer o trabalho em segundo plano

Estendendo a classe IntentService

Porque a maioria dos serviços inicializados não precisam lidar com vários pedidos ao mesmo tempo (o que pode realmente ser um cenário multi-threading perigoso), provavelmente é melhor se você implementar o seu serviço usando a classe `IntentService`.

O IntentService faz o seguinte:

Criar uma thread de trabalho padrão que executa todas as intents entregues com `onStartCommand()` separados da thread principal do seu aplicativo.

Cria uma fila de trabalho que passa uma intent em um momento de sua implementação `onHandleIntent()`, para que você nunca mais tenha de se preocupar com multi-threading.

Interrompe o serviço depois que todos os pedidos iniciais foram manipulados, assim você nunca tem que chamar `stopSelf()`.

Fornece implementação padrão `onBind()` que retorna nulo.

Fornecer uma implementação padrão de `onStartCommand()` que envia a intent de uma fila de trabalho e, em seguida, para o seu `onHandleIntent()` implementado.

A isso acrescenta-se ao fato de que tudo que você precisa fazer é implementar `onHandleIntent()` para fazer o trabalho fornecido pelo cliente. (Embora, você também precisa fornecer um construtor pequeno para o serviço.)

Aqui está um exemplo de implementação `IntentService`:

```
public class HelloIntentService extends IntentService {

    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
     * The IntentService calls this method from the default worker thread with
     * the intent that started the service. When this method returns, IntentService
     * stops the service, as appropriate.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
```

```

        wait(endTime - System.currentTimeMillis());
    } catch (Exception e) {
    }
}
}
}
}
}
}

```

Isso é tudo que você precisa: um construtor e uma implementação de `onHandleIntent()`.

Se você decidir também substituir outros métodos de retorno, como `onCreate()`, `onStartCommand()`, ou `OnDestroy()`, certifique-se de chamar a implementação `super`, de modo que o `IntentService` pode tratar adequadamente a vida da thread de trabalho.

Por exemplo, `onStartCommand()` deve retornar a implementação padrão (que é como a intent é entregue para `onHandleIntent()`):

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
    return super.onStartCommand(intent, flags, startId);
}

```

Além de `onHandleIntent()`, o único método do qual você não precisa chamar da classe `super` é `onBind()` (mas você precisa implementar se o seu serviço permite a ligação).

Estendendo a classe de serviço

Como vimos anteriormente, usando `IntentService` fazemos a implementação da inicialização de um serviço muito simples. Se, entretanto, você solicitar ao seu serviço para executar um multi-threading (em vez de

processar os pedidos através de uma fila de trabalho), então você pode estender a classe de serviço para lidar com cada intent.

Para efeito de comparação, o código de exemplo a seguir é uma implementação da classe de serviço que realiza o trabalho exatamente como o exemplo acima usando IntentService. Isto é, para cada pedido, ele usa uma thread para executar o trabalho e processa apenas um pedido de cada vez.

```
public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {
            // Normally we would do some work here, like download a file.
            // For our sample, we just sleep for 5 seconds.
            long endTime = System.currentTimeMillis() + 5*1000;
            while (System.currentTimeMillis() < endTime) {
                synchronized (this) {
                    try {
                        wait(endTime - System.currentTimeMillis());
                    } catch (Exception e) {
                    }
                }
            }
            // Stop the service using the startId, so that we don't stop
            // the service in the middle of handling another job
            stopSelf(msg.arg1);
        }
    }
}
```

```
}
```

```
@Override
```

```
public void onCreate() {
```

```
    // Start up the thread running the service. Note that we create a
```

```
    // separate thread because the service normally runs in the process's
```

```
    // main thread, which we don't want to block. We also make it
```

```
    // background priority so CPU-intensive work will not disrupt our UI.
```

```
    HandlerThread thread = new HandlerThread("ServiceStartArguments",
```

```
        Process.THREAD_PRIORITY_BACKGROUND);
```

```
    thread.start();
```

```
    // Get the HandlerThread's Looper and use it for our Handler
```

```
    mServiceLooper = thread.getLooper();
```

```
    mServiceHandler = new ServiceHandler(mServiceLooper);
```

```
}
```

```
@Override
```

```
public int onStartCommand(Intent intent, int flags, int startId) {
```

```
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show();
```

```
    // For each start request, send a message to start a job and deliver the
```

```
    // start ID so we know which request we're stopping when we finish the job
```

```
    Message msg = mServiceHandler.obtainMessage();
```

```
    msg.arg1 = startId;
```

```
    mServiceHandler.sendMessage(msg);
```

```
    // If we get killed, after returning from here, restart
```

```
    return START_STICKY;
```

```
}
```

```
@Override
```

```
public IBinder onBind(Intent intent) {
```

```
    // We don't provide binding, so return null
```

```
        return null;
    }

    @Override
    public void onDestroy() {
        Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show();
    }
}
```

Como você pode ver, é muito mais trabalho do que usar `IntentService`.

No entanto, porque você lida com cada chamada para `onStartCommand()`, você pode executar várias solicitações simultaneamente. Isso não é o que este exemplo faz, mas se é isso que você quer, então você pode criar um novo tópico para cada pedido e executá-los imediatamente (em vez de esperar que o pedido anterior conclua).

Observe que o método `onStartCommand()` deve retornar um inteiro. O inteiro é um valor que descreve como o sistema deve continuar o serviço no caso em que o sistema precisar encerrar o serviço. O valor de retorno de `onStartCommand()` deve ser uma das seguintes constantes:

START_NOT_STICKY

Se o sistema encerra o serviço após o retorno de `onStartCommand()`, não recria o serviço, a não ser que existam intents pendentes para entregar. Esta é a opção mais segura para evitar a execução de seu serviço quando não for necessário e quando o aplicativo pode simplesmente reiniciar os trabalhos inacabados.

START_STICKY

Se o sistema encerra o serviço após o retorno de `onStartCommand()`, recria o serviço e chama `onStartCommand()`, mas não entrega novamente a

intent passada. Em vez disso, o sistema chama `onStartCommand()` com uma intent nula, a menos que houvesse intents pendentes para iniciar o serviço, caso em que, essas intents são entregues. Isso é adequado para players de mídia (ou serviços semelhantes) que não estão executando comandos, mas rodando indefinidamente e à espera de um trabalho.

START_REDELIVER_INTENT

Se o sistema mata o serviço após o retorno `onStartCommand()`, recria o serviço e chamada `onStartCommand()` com a última intent que foi entregue para o serviço. Quaisquer intents pendentes são entregues posteriormente. Isso é adequado para serviços que estão ativamente execução de um trabalho que devem ser imediatamente retomadas, como o download de um arquivo.

Iniciando um Serviço

Você pode iniciar um serviço a partir de uma atividade ou componente de uma aplicação diferente pela passagem de uma Intent (especificando o serviço que quer iniciar) para `StartService()`. O sistema Android chama o método do serviço `onStartCommand()` e passa as Intents. (Você nunca deve chamar `onStartCommand()` diretamente.)

Por exemplo, uma atividade pode iniciar o serviço do exemplo na seção anterior (HelloService) usando uma intent explícita com `StartService()`:

```
Intent intent = new Intent(this, HelloService.class);  
startService(intent);
```

`StartService()` retorna imediatamente e o sistema Android chama o método do serviço `onStartCommand()`. Se o serviço não estiver em execução, o sistema chama primeiro `onCreate()`, depois chama `onStartCommand ()`.

Se o serviço não fornecer o vínculo, a intent entregue com `StartService()` é o único modo de comunicação entre o componente da aplicação e o serviço.

No entanto, se você quiser que o serviço envie um resultado de volta, o cliente que inicia o serviço pode criar um `PendingIntent` (Uma descrição de uma intent e ação de destino para realizar com ela) para uma transmissão (com `getBroadcast()`) e entregá-lo para o serviço na Intent que inicia o serviço. O serviço pode então usar a transmissão para entregar o resultado.

Muitos pedidos iniciam um serviço em resposta a várias chamadas de correspondentes de `onStartCommand()`. No entanto, é necessário que apenas um pedido encerre o serviço (com `stopSelf()` ou `StopService()`).

Parando um serviço

Um serviço deve gerenciar seu próprio ciclo de vida. Ou seja, o sistema não interrompe ou destrói o serviço a menos que ele precise recuperar a memória do sistema e o serviço continua a funcionar depois que `onStartCommand()` retorna. Assim, o serviço deve encerrar chamando `stopSelf()` ou outro componente pode pará-lo chamando `StopService()`.

Uma vez solicitado para encerrar com `stopSelf()` ou `StopService()`, o sistema destrói o serviço logo que possível.

No entanto, se o serviço lida com várias solicitações `onStartCommand()` ao mesmo tempo, então você não deve interromper o serviço quando tiver terminado o processamento de um pedido, porque você pode ter recebido um novo pedido para iniciar (parando no final do primeiro pedido iria encerrar o segundo). Para evitar esse problema, você pode usar `stopSelf(int)` para garantir que o seu pedido sempre encerrara baseado no pedido de início mais recente. Isto é, quando você chamar `stopSelf(int)`, você passa o ID do pedido inicial (a `startID` entregue em `onStartCommand()`) para que o seu pedido de parada encerre o pedido corresponde. Então, se o serviço recebeu um novo pedido antes de você fosse capaz de chamar `stopSelf(int)`, o ID não irá corresponder e o serviço não vai parar.

Criando um serviço vinculado

Um serviço vinculado permite que se ligue um componente da aplicação a ele com uma chamada `bindService()`, criando uma ligação de longa duração (e, geralmente, não permite que os componentes o iniciem chamando `StartService()`).

Você deve criar este tipo de serviço quando você quiser interagir com o serviço de atividades e outros componentes em seu aplicativo ou para expor algumas das funcionalidades do aplicativo para outras aplicações, por meio de comunicação entre processos (IPC).

Para criar um serviço vinculado, você deve implementar o `onBind ()` método de retorno para retornar um `IBinder` que define a interface para a comunicação com o serviço. Outros componentes de aplicação pode então chamar `bindService ()` para recuperar a interface e começar a chamar métodos no serviço. O serviço só vive para servir o componente de aplicação que é ligado a ele, então quando não há componentes vinculados ao serviço, o sistema destrói (você não precisa interromper um serviço vinculado na forma como você deve quando o serviço é iniciado através `onStartCommand ()`).

Para criar um serviço vinculado, a primeira coisa que você deve fazer é definir a interface que especifica como um cliente pode se comunicar com o serviço. Esta interface entre o serviço e o cliente deve ser uma implementação de `IBinder` e é o que o seu serviço deve retornar a partir do método `onBind()`. Uma vez que o cliente recebe o `IBinder`, ele pode começar a interagir com o serviço por meio dessa interface.

Vários clientes podem ligar-se ao serviço de uma só vez. Quando um cliente termina a interação com o serviço, ele chama `unbindService()` para desvincular. Uma vez que não há clientes ligados ao serviço, o sistema destrói o serviço.

Existem várias maneiras de implementar um serviço vinculado e a implementação é mais complicado do que um outro tipo de serviço, então a discussão completa sobre serviços vinculados não aparece neste documento.

Execução de um serviço em primeiro plano

Um serviço de primeiro plano é um serviço que é considerado como algo que o usuário está ativamente consciente e, portanto, não um candidato para o sistema de encerrar quando esta com pouca memória. Um serviço de primeiro plano deve fornecer uma notificação para a barra de status, que é colocado sob o título "em curso", o que significa que a notificação não pode ser finalizado a menos que o serviço esteja parado ou removido do primeiro plano.

Por exemplo, um player de música que toca música de um serviço deve ser definido para ser executado em primeiro plano, porque o usuário é explicitamente consciente de sua operação. A notificação na barra de status pode indicar a música atual e permitir que o usuário inicie uma atividade para interagir com o player de música.

Para solicitar que seu serviço executado em primeiro plano, ligue `startForeground()`. Este método tem dois parâmetros: um inteiro que identifica a notificação e a notificação para a barra de status. Por exemplo:

```
Notificação de notificação = new Notificação (R.drawable.icon, getText
                                     (R.string.ticker_text),
                                     System.currentTimeMillis ());
NotificationIntent intenção Intenção = new (este, ExampleActivity.class);
PendingIntent PendingIntent PendingIntent.getActivity = (isto, 0,
                                     notificationIntent, 0);
notification.setLatestEventInfo (isto, getText (R.string.notification_title),
                                getText (R.string.notification_message), PendingIntent);
startForeground (ONGOING_NOTIFICATION, notificação);
```

Para remover o serviço do primeiro plano, ligue `stopForeground()`. Este método usa um booleano, indicando se remover a notificação de barra de status também. Este método não para o serviço. No entanto, se você parar o serviço enquanto ele ainda está executando em primeiro plano, em seguida, a notificação também é removido.

Nota: Os métodos `startForeground()` e `stopForeground()` foram introduzidas no Android 2.0 (API Level 5). A fim de executar o serviço em primeiro plano em versões mais antigas da plataforma, você deve usar o método `SetForeground()` ao invés de `startForeground()`.

Gerenciando o Ciclo de Vida de um Serviço

O ciclo de vida de um serviço é muito mais simples do que a de uma atividade. No entanto, é ainda mais importante que você preste atenção à forma como o serviço é criado e destruído, porque um serviço pode ser executado em segundo plano, sem o utilizador se aperceba.

O ciclo de vida de um serviço, desde quando ele é criado ate quando ele é destruído, pode seguir dois caminhos diferentes:

Um serviço iniciado

O serviço é criado quando outro componente chama `StartService()`. O serviço é executado em seguida, por tempo indeterminado e deve ser finalizado chamando `stopSelf()`. Outro componente também pode parar o serviço chamando `StopService()`. Quando o serviço for interrompido, o sistema o destrói.

Um serviço vinculado

O serviço é criado quando outro componente (um cliente) chama `bindService()`. O cliente se comunica com o serviço através de uma interface `IBinder`. O cliente pode fechar a conexão chamando `unbindService()`. Vários

clientes podem ligar-se ao mesmo serviço e quando todos eles se desligarem, o sistema destrói o serviço.

Estes dois caminhos não são totalmente separados. Ou seja, você pode ligar um serviço que já foi iniciado com `StartService()`. Por exemplo, um serviço de música de fundo pode ser iniciado pelo telefone com `StartService()` uma intent que identifica que música vai tocar. Mais tarde, possivelmente quando o usuário desejar exercer algum controle sobre o player ou obter informações sobre a música atual, uma atividade pode ligar para o serviço chamando `bindService()`. Em casos como este, `StopService()` ou `stopSelf()` na verdade não interrompe o serviço até que todos os clientes se desvinculem.

Implementar os retornos do ciclo de vida

Como uma atividade, um serviço tem métodos de retorno do ciclo de vida que você pode implementar para monitorar mudanças no estado do serviço e executar o trabalho nos momentos apropriados. O esqueleto do serviço esta demonstrado a seguir e cada um dos métodos de ciclo de vida:

```
public class ExampleService extends Service {
    int mStartMode;    // indicates how to behave if the service is killed
    IBinder mBinder;   // interface for clients that bind
    boolean mAllowRebind; // indicates whether onRebind should be used

    @Override
    public void onCreate() {
        // The service is being created
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // The service is starting, due to a call to startService()
        return mStartMode;
    }

    @Override
```

```

public IBinder onBind(Intent intent) {
    // A client is binding to the service with bindService()
    return mBinder;
}

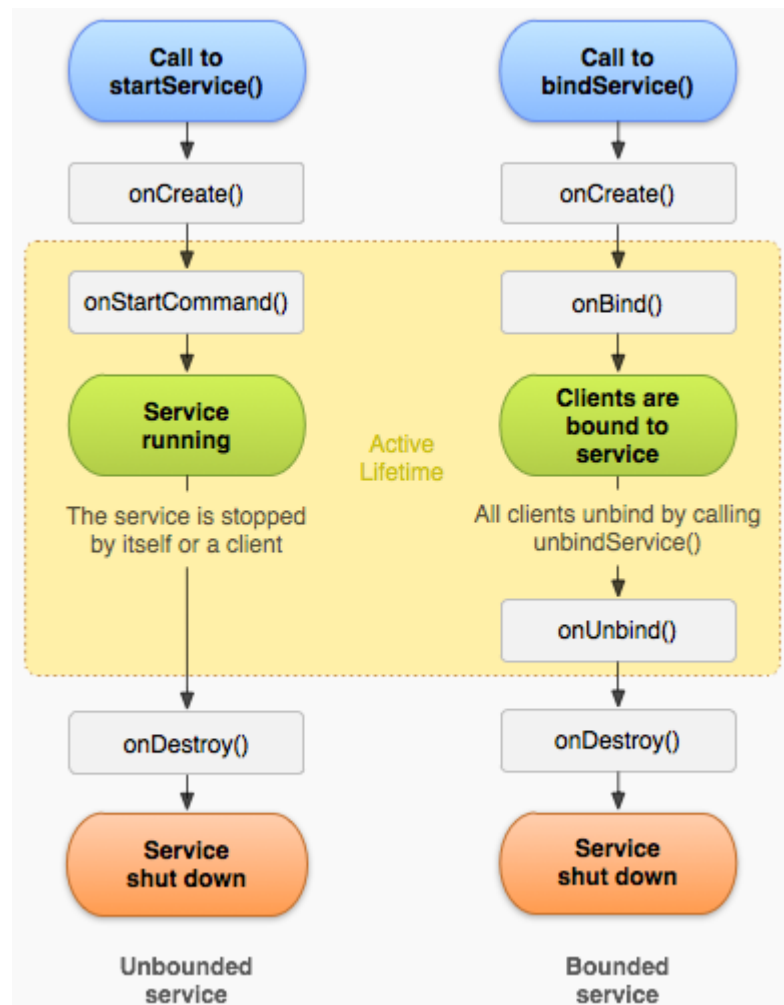
@Override
public boolean onUnbind(Intent intent) {
    // All clients have unbound with unbindService()
    return mAllowRebind;
}

@Override
public void onRebind(Intent intent) {
    // A client is binding to the service with bindService(),
    // after onUnbind() has already been called
}

@Override
public void onDestroy() {
    // The service is no longer used and is being destroyed
}
}

```

Nota: Ao contrário dos métodos de retorno do ciclo de vida da atividade, você não é obrigado a chamar a implementação de superclasse destes métodos de retorno de chamada.



O ciclo de vida do serviço. O diagrama acima mostra o ciclo de vida quando o serviço é criado com `StartService()` e o diagrama da direita mostra o ciclo de vida quando o serviço é criado com `bindService()`.

Ao implementar esses métodos, você pode monitorar dois loops aninhados do ciclo de vida do serviço:

A vida inteira de um serviço acontece entre o `onCreate()` e o `onDestroy()`. Como uma atividade, um serviço faz a sua configuração inicial no `onCreate()` e libera todos os recursos restantes no `onDestroy()`. Por exemplo, um serviço de reprodução de música poderia criar o segmento onde a música vai ser tocada no `onCreate()`, em seguida, parar no segmento `onDestroy()`.

Métodos `onCreate()` e `onDestroy()` são chamados para todos os serviços, sejam eles criados por `StartService()` ou `bindService()`.

A vida ativa de um serviço começa com uma chamada para qualquer `onStartCommand()` ou `onBind()`. Cada método é entregue a `Intent` que foi passado para `StartService()` ou `bindService()`, respectivamente.

Se o serviço é iniciado, o tempo de vida ativa termina ao mesmo tempo que as extremidades vida inteira (o serviço ainda está ativo, mesmo após `onStartCommand()` retorna). Se o serviço é limitado, o tempo de vida ativa termina quando retorna `onUnbind()`.

Nota: Apesar de um serviço iniciado é interrompido por uma chamada para qualquer `stopSelf()` ou `StopService()`, não existe um retorno para o respectivo serviço (não há retorno de chamada `onStop()`). Então, a menos que o serviço está vinculado a um cliente, o sistema destrói quando o serviço for interrompido, `onDestroy()` é o retorno.

A Figura acima ilustra os métodos de retorno típicas para um serviço. Embora a figura separe os serviços que são criados por `StartService()` daqueles criados por `bindService()`, tenha em mente que qualquer serviço, não importa como ele é iniciado, pode, potencialmente, permitir que os clientes se liguem a ele. Assim, um serviço que inicialmente começou com `onStartCommand()` (por um cliente chamado `StartService()`) pode ainda receber uma chamada para `onBind()` (quando um cliente chama `bindService()`).

Arquivo de Manifesto

Cada aplicação deve ter um arquivo `AndroidManifest.xml` em seu diretório raiz. O manifesto apresenta informações essenciais sobre o aplicativo para o sistema Android, informações que o sistema deve ter antes de executar qualquer um código do aplicativo. Entre outras coisas, o manifesto faz o seguinte:

- Ele cita o pacote Java para a aplicação. O nome do pacote serve como um identificador exclusivo para o aplicativo.
- Ele descreve os componentes da aplicação - as atividades, serviços, receptores de broadcast e fornecedores de conteúdo que a aplicação é composta. Ele cita as classes que implementam cada um dos componentes e publica as suas capacidades (por exemplo, as mensagens que eles podem lidar com Intents). Estas declarações deixam o sistema Android saber o que os componentes são e em que condições eles podem ser utilizados.
- Ele determina que processos vão sediar os componentes do aplicativo.
- Ele declara que permissões do aplicativo deve ter para acessar partes protegidas da API e interagir com outras aplicações.
- Ele também declara as permissões que os outros são obrigados a ter, a fim de interagir com os componentes do aplicativo.
- Ele lista as classes de instrumentação que fornecem perfis e outras informações, como o aplicativo é executado. Estas declarações estão presentes no manifesto apenas enquanto o aplicativo está sendo desenvolvido e testado, eles são removidos antes da aplicação ser publicado.
- Ele declara o nível mínimo da API do Android que o aplicativo requer.
- Ele lista as bibliotecas necessárias a execução do aplicativo.

Estrutura do arquivo de manifesto

O diagrama abaixo mostra a estrutura geral do arquivo e todos os elementos que ele pode conter.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
  <uses-permission />
  <permission />
  <permission-tree />
  <permission-group />
```

```
<instrumentation />
<uses-sdk />
<uses-configuration />
<uses-feature />
<supports-screens />
<compatible-screens />
<supports-gl-texture />
<application>
  <activity>
    <intent-filter>
      <action />
      <category />
      <data />
    </intent-filter>
    <meta-data />
  </activity>
  <activity-alias>
    <intent-filter> . . . </intent-filter>
    <meta-data />
  </activity-alias>
  <service>
    <intent-filter> . . . </intent-filter>
    <meta-data/>
  </service>
  <receiver>
    <intent-filter> . . . </intent-filter>
    <meta-data />
  </receiver>
  <provider>
    <grant-uri-permission />
    <meta-data />
    <path-permission />
  </provider>
```

```
<uses-library />  
</application>  
</manifest>
```

Todos os elementos que podem aparecer no arquivo de manifesto estão listados abaixo em ordem alfabética. Estes são os únicos elementos legais, você não pode adicionar seus próprios elementos ou atributos.

```
<action>  
<activity>  
<activity-alias>  
<application>  
<category>  
<data>  
<grant-uri-permission>  
<instrumentation>  
<intent-filter>  
<manifest>  
<meta-data>  
<permission>  
<permission-group>  
<permission-tree>  
<provider>  
<receiver>  
<service>  
<supports-screens>  
<uses-configuration>  
<uses-feature>  
<uses-library>  
<uses-permission>  
<uses-sdk>
```

Convenções de arquivo

Algumas convenções e regras de aplicação geral a todos os elementos e atributos do manifesto:

Elementos

Somente os elementos <manifest> e <application> são necessários, cada um deve estar presente e só pode ocorrer uma vez. A maioria dos outros pode ocorrer muitas vezes ou não ocorrer - embora pelo menos alguns deles devem estar presentes para o manifesto de realizar algo significativo.

Se um elemento não contém nada, ele contém outros elementos. Todos os valores são definidos através de atributos, e não como dados de caracteres dentro de um elemento.

Elementos do mesmo nível geralmente não são solicitados. Por exemplo os elementos <activity>, <provider> e <service> podem ser misturados em qualquer sequência. (Um elemento <activity-alias> é a exceção a esta regra: Ele deve seguir as <activity> é um alias)

Atributos

Em um sentido formal, todos os atributos são opcionais. No entanto, há alguns que devem ser especificados para um elemento para cumprir a sua finalidade. Para atributos verdadeiramente opcionais, menciona um valor padrão ou estados que acontecem na ausência de uma especificação.

Com exceção de alguns atributos do elemento raiz <manifest>, todos os nomes de atributos começam com um andróide:prefixo - por exemplo, android: alwaysRetainTaskState. Porque o prefixo é universal, a documentação em geral omite quando se refere a atributos por nome.

Declarando nomes de classe

Muitos elementos correspondem a objetos Java, incluindo elementos para a aplicação em si (o elemento <application>) e seus componentes principais - atividades (<activity>), serviços (<service>), receptores de broadcast (<receiver>) e provedores de conteúdo (<provider>).

Se você definir uma subclasse, como quase sempre faria para as classes de componentes (Activity, Service, BroadcastReceiver, e ContentProvider), a subclasse é declarado por meio de um atributo name. O name deve constar a designação do pacote completo. Por exemplo, uma subclasse de serviço pode ser declarado como segue:

```
<manifest . . . >
  <application . . . >
    <service android:name="com.example.project.SecretService" . . . >
      . . .
    </service>
    . . .
  </application>
</manifest>
```

No entanto, como um atalho, se o primeiro caractere da cadeia é um período, a seqüência é anexado ao nome do aplicativo do pacote (conforme especificado pelo atributo do elemento <manifest> do pacote). A atribuição seguinte é o mesmo que o descrito acima:

```
<manifest package="com.example.project" . . . >
  <application . . . >
    <service android:name=".SecretService" . . . >
      . . .
    </service>
    . . .
  </application>
```

</manifest>

Ao iniciar um componente, o Android cria uma instância da subclasse chamada. Se uma subclasse não é especificada, ele cria uma instância da classe base.

Vários valores

Se mais do que um valor pode ser especificado, os elementos são quase sempre repetidos, em vez de listar os valores múltiplos dentro de um único elemento. Por exemplo, um filtro de intent pode listar várias ações:

```
<intent-filter . . . >
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.INSERT" />
    <action android:name="android.intent.action.DELETE" />
    . . .
</intent-filter>
```

Valores de recursos

Alguns atributos têm valores que podem ser exibidas para os usuários - por exemplo, um rótulo e um ícone para uma atividade. Os valores desses atributos deve ser localizados e, portanto, definir um recurso ou tema. Valores de recursos são expressos no seguinte formato:

@[package:]type:name

onde o nome do pacote pode ser omitido se o recurso está no mesmo pacote do aplicativo, tipo é um tipo de recurso - como "string" ou "drawable" – e o nome é o nome que identifica o recurso específico. Por exemplo:

```
<activity android:icon="@drawable/smallPic" . . . >
```

Valores a partir de um tema são expressas de uma forma semelhante, mas com uma primeira '?' em vez de "@":

?[package:]type:name

Valores de strings

Onde um valor de atributo é uma string, duas barras invertidas ('\') deve ser usado para especificar os caracteres - por exemplo, '\n' para uma nova linha ou '\uxxxx' para um caractere Unicode.

Características de arquivo

As seções seguintes descrevem como algumas características do Android são refletidas no arquivo de manifesto.

Filtros de Intent

Os principais componentes de uma aplicação (suas atividades, serviços e receptores de broadcast) são ativados por intents. Uma intent é um conjunto de informações (um objeto Intent) descrevendo uma ação desejada - incluindo os dados a serem postas em prática, a categoria do componente que deve executar a ação, e outras instruções pertinentes. Android localiza um elemento adequado para responder à intent, inicia uma nova instância do componente se for necessário, e passa o objeto Intent.

Componentes anunciam suas capacidades - os tipos de intents que pode responder - através de filtros de intents. Desde que o sistema Android tem que entender quais componentes uma intent pode suportar antes de executar o componente, filtros intent são especificados no manifesto como elementos <intent-filter>. Um componente pode ter qualquer número de filtros, cada um descrevendo uma capacidade diferente.

Uma intent que, explicitamente, nomeia um componente de destino irá ativar esse componente, o filtro não desempenha um papel. Mas uma intent que não especifica um destino pelo nome pode ativar um componente só se este passar por um dos filtros do componente.

Ícones e Labels

Uma série de elementos tem o atributo ícone e labels para um pequeno ícone e um rótulo de texto que pode ser exibido para os usuários. Alguns também têm um atributo de descrição de um texto mais explicativo que também pode ser mostrado na tela. Por exemplo, o elemento `<permission>` tem todos esses três atributos, para que quando o usuário é perguntado se deve conceder a permissão para um aplicativo que tenha solicitado, um ícone que representa a autorização, o nome da permissão, e uma descrição e o que ela implica, podem ser apresentados ao utilizador.

Em todos os casos, o ícone e o conjunto de labels em um elemento contendo se torna o ícone padrão e as configurações de etiquetas são para todos os subelementos do contêiner. Assim, o ícone e o rótulo definido no elemento `<application>` são o ícone padrão e uma etiqueta para cada um dos componentes do aplicativo. Da mesma forma, o ícone e o conjunto de rótulos de um componente - por exemplo, um elemento `<activity>` - são as configurações padrão para cada um dos elementos do componente `<intent-filter>`. Se um elemento `<application>` define um rótulo, mas uma atividade e filtro de sua intent de não fazer, o rótulo de aplicação é tratado como o rótulo para tanto a atividade e o filtro de intent.

O ícone de etiqueta e ajustar para um filtro de intent são usados para representar um componente quando o componente é apresentado ao utilizador como cumprindo a função anunciados pelo filtro. Por exemplo, um filtro com `"android.intent.action.MAIN"` e `"android.intent.category.LAUNCHER"` configurações anuncia uma atividade como aquele que inicia uma aplicação - isto é, como um que deve ser exibido na tela do menu. O ícone e o rótulo definido no filtro, portanto, são os mais exibidos no aplicativo.

Permissões

A permissão de uma restrição limitando o acesso a uma parte do código ou a dados sobre o dispositivo. A limitação é imposta para proteger dados críticos e códigos que poderiam ser utilizadas para distorcer ou danificar a experiência do usuário.

Cada permissão é identificada por um único rótulo. Muitas vezes, o rótulo indica a ação que está restrita. Por exemplo, aqui estão algumas permissões definidas pelo Android:

```
android.permission.CALL_EMERGENCY_NUMBERS  
android.permission.READ_OWNER_DATA  
android.permission.SET_WALLPAPER  
android.permission.DEVICE_POWER
```

A característica pode ser protegida por no máximo uma permissão.

Se um aplicativo precisa ter acesso a um recurso protegido por uma permissão, ele deve declarar que ele exige que a permissão com um elemento `<uses-permission>` no manifesto. Então, quando o aplicativo é instalado no dispositivo, o instalador determina se deve ou não conceder a permissão solicitada, verificando as autoridades que assinaram certificados do aplicativo e, em alguns casos, perguntar ao usuário. Se a permissão for concedida, o aplicativo é capaz de usar os recursos protegidos. Se não, suas tentativas de acesso a esses recursos simplesmente falhará sem qualquer notificação ao usuário.

Um aplicativo também pode proteger seus próprios componentes (atividades, serviços, receptores de broadcast e provedores de conteúdo) com permissões. Ele pode empregar qualquer das permissões definidas pelo Android (listados na `android.Manifest.permission`) ou declarados por outras aplicações. Ou pode definir o seu próprio. Uma nova permissão é declarada

com o elemento <permission>. Por exemplo, uma actividade pode ser protegida como se segue:

```
<manifest . . . >
  <permission android:name="com.example.project.DEBIT_ACCT" . . . />
  <uses-permission android:name="com.example.project.DEBIT_ACCT" />
  . . .
  <application . . .>
    <activity android:name="com.example.project.FreneticActivity"
      android:permission="com.example.project.DEBIT_ACCT"
      . . . >
      . . .
    </activity>
  </application>
</manifest>
```

Nota-se que, neste exemplo, a permissão DEBIT_ACCT é não só com o elemento <permission> declarado, a sua utilização é também solicitada com o elemento <uses-permission>. Seu uso deve ser solicitado para que outros componentes do aplicativo possam iniciar a atividade protegida, mesmo que a proteção seja imposta pelo próprio aplicativo.

Se, no mesmo exemplo, o atributo de permissão foi definida como uma permissão para outros países (como android.permission.CALL_EMERGENCY_NUMBERS, não teria sido necessário declará-la novamente com um elemento <permission>. Entretanto, ele ainda teria sido necessário solicitar a sua utilização com <uses-permission>.

O elemento <permission-tree> declara um namespace para um grupo de permissões que serão definidos no código. E <permission-group> define um rótulo para um conjunto de permissões (tanto os declarados no manifesto com elementos <permission> e os declarados em outros lugares). Ela afeta apenas como as permissões são agrupadas ao serem apresentadas ao usuário. O elemento <permission-group> não especifica quais permissões

pertencem ao grupo, ele só dá um nome ao grupo. A permissão é colocada no grupo, atribuindo o nome do grupo para o atributo do elemento <permission> de permissionGroup.

Bibliotecas

Cada aplicativo está vinculado a biblioteca padrão Android, que inclui os pacotes básicos para a criação de aplicativos (com as classes comuns, tais como o Serviço, Intent, View, Button, Aplicação, provedores de conteúdo, e assim por diante).

No entanto, alguns pacotes residem em suas próprias bibliotecas. Se o seu aplicativo utiliza o código de qualquer um desses pacotes, ele deve explicitamente pedir para ser ligado a eles. O manifesto deve conter um elemento separado <uses-library> com o nome de cada uma das bibliotecas. (O nome de biblioteca pode ser encontrado na documentação para o pacote.)

Tipos de Elementos que podem ser colocados no AndroidManifest

A seguir alguns dos principais elementos que podem ser informados no AndroidManifest.

<action>

sintaxe:

```
<action android:name="string" />
```

Colocar em : <intent-filter>

Descrição:

Adiciona uma ação para um filtro de intent. Um elemento <intent-filter> deve conter um ou mais elementos <action>. Se ele não contém nenhum, não existem objectos Intent que passem pelo filtro.

atributos: android:name

Nome da ação. Algumas ações padrão são definidas na Intent de classe como constantes ACTION_string. Para atribuir uma dessas ações a esse atributo, utilize "android.intent.action." para a string que segue ACTION_ . Por exemplo, para ACTION_MAIN, use "android.intent.action.MAIN" e para ACTION_WEB_SEARCH use "android.intent.action.WEB_SEARCH".

Para as ações definidas, o melhor é usar o nome do pacote como um prefixo para garantir a exclusividade. Por exemplo, uma ação Transmogrify pode ser especificada como se segue:

```
< action android:name="com.example.project.TRANSMOGRIFY" />
```

introduzida em:

API Nível 1

<activity>

Sintaxe :

```
<activity android:allowTaskReparenting=["true" | "false"]
    android:alwaysRetainTaskState=["true" | "false"]
    android:clearTaskOnLaunch=["true" | "false"]
    android:configChanges=["mcc", "mnc", "locale",
        "touchscreen", "keyboard", "keyboardHidden",
        "navigation", "screenLayout", "fontScale", "uiMode",
        "orientation", "screenSize", "smallestScreenSize"]
    android:enabled=["true" | "false"]
    android:excludeFromRecents=["true" | "false"]
    android:exported=["true" | "false"]
    android:finishOnTaskLaunch=["true" | "false"]
    android:hardwareAccelerated=["true" | "false"]
    android:icon="drawable resource"
```

```

        android:label="@string/resource"
        android:launchMode=["multiple" | "singleTop" |
            "singleTask" | "singleInstance"]
        android:multiprocess=["true" | "false"]
        android:name="@string"
        android:noHistory=["true" | "false"]
        android:parentActivityName="@string"
        android:permission="@string"
        android:process="@string"
        android:screenOrientation=["unspecified" | "user" | "behind" |
            "landscape" | "portrait" |
            "reverseLandscape" | "reversePortrait" |
            "sensorLandscape" | "sensorPortrait" |
            "sensor" | "fullSensor" | "nosensor"]
        android:stateNotNeeded=["true" | "false"]
        android:taskAffinity="@string"
        android:theme="@resource or theme"
        android:uiOptions=["none" | "splitActionBarWhenNarrow"]
        android:windowSoftInputMode=["stateUnspecified",
            "stateUnchanged", "stateHidden",
            "stateAlwaysHidden", "stateVisible",
            "stateAlwaysVisible", "adjustUnspecified",
            "adjustResize", "adjustPan"] >
        . . .
    </activity>

```

Colocar em : <application>

pode conter: <intent-filter> e <meta-data>

Descrição:

Declara uma atividade (uma subclasse de Activity) que implementa a parte visual da interface de usuário do aplicativo. Todas as atividades devem

ser representadas por elementos <activity> no arquivo de manifesto. As que não são declaradas não serão visto pelo sistema e nunca serão executado.

atributos:

android: allowTaskReparenting

A atividade pode ou não ser movida para frente a partir da tarefa que começou com a tarefa e tem uma afinidade para quando essa tarefa ocorre - "true" se pode mover, e "false" se deve permanecer com a tarefa onde começou.

Se este atributo não for definido, o valor fixado pelo atributo correspondente allowTaskReparenting do elemento <application> se aplica à atividade. O valor padrão é "falso".

Normalmente, quando uma atividade é iniciada, ela é associada com a tarefa da atividade que começou e fica lá por toda sua vida útil. Você pode usar esse atributo para forçá-la a ser realocada para a tarefa que tem uma afinidade quando a sua tarefa atual não é mais exibida. Normalmente, ela é usada para fazer com que as atividades de um aplicativo mudarem para a tarefa principal associada com esse aplicativo.

Por exemplo, se uma mensagem de e-mail contém um link para uma página da Web, clicando no link traz uma atividade que pode exibir a página. Essa atividade é definida pela aplicação do browser, mas é lançado como parte da tarefa de e-mail. Se ele é associado a tarefa de navegador, ele será mostrado quando o navegador em seguida vem para a frente, e estará ausente quando a tarefa de e-mail novamente vem a frente.

A afinidade de uma activity é definida pelo atributo taskAffinity. A afinidade de uma tarefa é determinada através da leitura da afinidade da sua activity raiz. Portanto, por definição, uma activity raiz é sempre uma tarefa com a mesma afinidade. Como as atividades com modo de visualização

"singleTask" ou "SingleInstance" podem estar na raiz de uma tarefa, a usabilidade é limitada aos modos "standart" e "singleTop".

android: configChanges

Listas de alterações de configuração que a atividade irá lidar. Quando ocorre uma mudança de configuração em tempo de execução, a atividade é desligada e reiniciada por padrão, mas declarar uma configuração com este atributo irá impedir a atividade de ser reiniciada. Em vez disso, a actividade continuara a funcionar e o seu método `onConfigurationChanged()` sera chamado.

Qualquer um ou todos os seguintes textos são valores válidos para este atributo. Vários valores são separados por ' | '- por exemplo, "locale|navegation|orientation".

Valor	Descrição
"mcc "	O IMSI código de país móvel (MCC) mudou - um SIM foi detectado e atualizou o MCC.
"MNC "	O IMSI código de rede móvel (MNC) mudou - um SIM foi Detectado e atualizou o MNC.
"locale "	A localidade foi alterada - o usuário selecionou uma nova Linguagem que o texto deve ser apresentado.
"touchscreen"	O touchscreen mudou. (Isso nunca deveria acontecer normalmente.)
"keyboard "	O tipo de teclado mudou - por exemplo, o usuário tiver conectado um teclado externo.
"keyboardHidden"	A acessibilidade do teclado mudou - por exemplo, o utilizador tem utilizado o teclado de hardware.

"navegation"	O tipo de navegação (trackball / dpad) mudou. (Isso nunca deveria acontecer normalmente.)
"screenLayout "	O layout da tela mudou - isso pode ser causado por uma exibição diferente sendo ativado.
"fontScale "	O fator de escala de fontes mudou - o usuário tenha selecionado um novo tamanho de fonte global.
"uiMode "	O modo de interface de usuário mudou - o que pode ser causado quando o usuário coloca o dispositivo em uma doca de mesa / carro ou quando as mudanças do modo de noite.
"orientation"	A orientação da tela mudou - o usuário tem rodado o dispositivo.
"ScreenSize"	O tamanho da tela atual disponível mudou. Isso representa uma mudança no tamanho disponíveis atualmente, em relação à proporção atual, então vai mudar quando o usuário alterna entre paisagem e retrato.
"smallestScreenSize"	O tamanho da tela física mudou. Isso representa uma mudança no tamanho, independentemente da orientação, isso só vai mudar quando o tamanho da tela físico real mudou, como a mudança para um monitor externo. A alteração desta configuração corresponde a uma mudança na configuração smallestWidth .
"layoutDirection"	A direção do layout mudou. Por exemplo, mudar da esquerda para a direita (LTR) para a direita para a esquerda (RTL).

Todas essas alterações de configuração podem afetar os valores dos recursos vistos pela aplicação. Portanto, quando

onConfigurationChanged() é chamado, ele será geralmente necessário para voltar a recuperar todos os recursos (incluindo layouts view, Drawables, e assim por diante) para lidar corretamente com a mudança.

android:enabled

A atividade pode ou não ser instanciada pelo sistema - "true" se pode ser, e "false" se não. O valor padrão é "true".

O elemento <application> tem seu próprio habilitado atributo que se aplica a todos os componentes do aplicativo, incluindo atividades. Os atributos <application> e <activity> devem ambos ser "true" (já que ambos são por padrão) para que o sistema seja capaz de instanciar a atividade. Se um é "false", não pode ser instanciado.

android:label

Um label legível pelo usuário para a atividade. O label é exibido na tela quando a atividade é apresentada ao usuário. É muitas vezes apresentada juntamente com o ícone da atividade.

Se este atributo não for definido, o rótulo definido é o nome do aplicativo.

Label da atividade - se definir pelo elemento <application> - é também o label padrão para todos os filtros de intent da atividade.

O label deve ser definido como uma referência a um recurso de string, de modo que ela pode ser localizada como outras strings na interface de utilizador.

android: launchMode

Uma instrução sobre como a atividade deve ser iniciada. Existem quatro modos de que o trabalho em conjunto com as configurações da atividade (constantes FLAG_ACTIVITY_*) em objetos Intent para determinar o que deve acontecer quando a atividade é chamada para lidar com uma intent. Eles são:

"standard"

"singleTop"

"singleTask"

"singleInstance"

O modo padrão é "standard".

Como mostrado na tabela abaixo, os modos dividem-se em dois grupos principais, com as atividades "standard" e "singleTop" de um lado, e "singleTask" e "SingleInstance" do outro. Uma atividade com o "standard" ou "singleTop" podem ser instanciadas várias vezes. Os exemplos podem pertencer a todas as tarefas e pode estar localizado em qualquer lugar na pilha de atividade. Normalmente, eles são utilizados para a tarefa que chama startActivity() (a menos que o objecto contém uma Intent com a instrução FLAG_ACTIVITY_NEW_TASK, caso em que uma tarefa diferente é escolhida.

Em contraste, atividades "singleTask" e "SingleInstance" só podem começar uma tarefa. Eles são sempre na raiz da pilha de actividade. Além disso, o dispositivo pode conter apenas uma instância da atividade de cada vez.

Os modos "standad" e "singleTop" diferem entre si em apenas em um aspecto: Toda vez que há uma nova intent de uma atividade "standard", uma nova instância da classe é criada para responder a essa intent. Cada instância lida com um único intent. Da mesma forma, uma nova instância de um "singleTop". A atividade também pode ser criada para lidar com uma nova intent. No entanto, se a tarefa alvo já tem uma instância existente da atividade

no topo da sua pilha, que o exemplo receberá a intent (numa chamada `onNewIntent()`); uma nova instância não é criada. Em outras circunstâncias - por exemplo, se uma instância existente da atividade "singleTop" actividade é a tarefa alvo, mas não no topo da pilha, ou se é no topo de uma pilha, mas não na tarefa alvo - uma nova instância será criada e colocada na pilha.

Os métodos "singleTask" e "SingleInstance" também diferem uns dos outros em um aspecto: A atividade "singleTask" permite outras atividades como parte da sua missão. É sempre na raiz de sua tarefa, mas outras atividades (necessariamente "standart" e "singleTop") podem ser utilizadas na tarefa. A atividade "SingleInstance", por outro lado, não permite que outras atividades fassam ser parte da sua missão. É a única atividade na tarefa.

`android:multiprocess`

Verifica se a atividade pode ser utilizada no processo do componente que começou - "true" se pode ser, e "false" se não. O valor padrão é "false".

Normalmente, uma nova instância de uma atividade é utilizada no processo da aplicação que definiu, para todas as instâncias da atividade executarem no mesmo processo. No entanto, se esta opção é definido como "true", as instâncias da atividade podem ser executadas em vários processos, permitindo que o sistema crie instâncias onde eles são utilizados (permissões fornecidas permitir isso), algo que quase nunca é necessário ou desejável.

`android:name`

O nome da classe que implementa a atividade, uma subclasse da atividade. O valor do atributo deve ser um nome de classe totalmente qualificado (como, "com.example.project.ExtracurricularActivity"). No entanto, como um atalho, se o primeiro caractere do nome é um período (por exemplo, ".ExtracurricularActivity"), que é anexado ao nome do pacote especificado no elemento `<manifest>`.

Depois de publicar sua aplicação, você não deve alterar este nome (a menos que você tenha definido `android:exported = "false"`).

Não há padrão. O nome deve ser especificado.

`android:permission`

O nome de uma permissão que os clientes devem ter para iniciar a atividade ou fazê-la responder a uma intent. Se não foi concedida uma permissão específica a um chamador de `startActivity()` ou `startActivityForResult()`, sua intent não será entregue para a atividade.

Se este atributo não for definido, a permissão definida pelo elemento `<application>` a permissão do atributo se aplica à atividade. Se nem o atributo é definido, a atividade não é protegida por uma permissão.

`Android:process`

O nome do processo em que a atividade deve ser executado. Normalmente, todos os componentes de um aplicativo executado no processo padrão criado para a aplicação. Ela tem o mesmo nome que o pacote da aplicação. O elemento `<application>` processa o atributo podendo definir um padrão diferente de todos os componentes. Mas cada componente pode substituir o padrão, permitindo que você espalhe a sua aplicação em vários processos.

Se o nome atribuído a esse atributo começa com dois pontos (':'), um novo processo, privada para a aplicação, é criado quando é necessário e a atividade é executada nesse processo. Se o nome do processo começa com um caractere minúsculo, a atividade será executada em um processo global de mesmo nome, desde que tenha permissão para fazê-lo. Isto permite que os componentes em diferentes aplicações para compartilhar um processo, reduzindo o uso de recursos.

introduzida em:

API Nível 1 para todos os atributos, exceto para `noHistory` e `windowSoftInputMode`, que foram adicionados em API Nível 3.

<application>

sintaxe:

```
<application android:allowTaskReparenting=["true" | "false"]
    android:backupAgent="string"
    android:debuggable=["true" | "false"]
    android:description="string resource"
    android:enabled=["true" | "false"]
    android:hasCode=["true" | "false"]
    android:hardwareAccelerated=["true" | "false"]
    android:icon="drawable resource"
    android:killAfterRestore=["true" | "false"]
    android:largeHeap=["true" | "false"]
    android:label="string resource"
    android:logo="drawable resource"
    android:manageSpaceActivity="string"
    android:name="string"
    android:permission="string"
    android:persistent=["true" | "false"]
    android:process="string"
    android:restoreAnyVersion=["true" | "false"]
    android:supportsRtl=["true" | "false"]
    android:taskAffinity="string"
    android:theme="resource or theme"
    android:uiOptions=["none" | "splitActionBarWhenNarrow"] >
    . . .
</application>
```

Informar em : `<manifest>`

pode conter: <activity>, <activity-alias>, <service>, <receiver>, <provider> e <uses-library>

Descrição:

Declaração da aplicação. Este elemento contém subelementos que declaram cada um dos componentes do aplicativo e tem atributos que podem afetar todos os componentes. Muitos desses atributos (como ícone, label, permissão, processo, taskAffinity e allowTaskReparenting) possuem valores padrão definidos para os atributos correspondentes dos elementos do componentes. Outros (como debug, habilitado, descrição e allowClearUserData) não possuem valores definidos para a aplicação como um todo e não pode ser substituído pelos componentes.

atributos

android: allowTaskReparenting

Define se o aplicativo pode ou não passar as tarefas iniciadas por uma atividade para outra que têm uma afinidade para quando essa tarefa possui os seguintes valores - "true" se elas podem se mover, e "false" se eles devem permanecer com a tarefa onde começou. O valor padrão é "false" .

O elemento <activity> tem seu próprio atributo allowTaskReparenting que pode substituir o valor definido aqui.

android:BackupAgent

Nome da classe que implementada é BackupAgent do aplicativo, uma subclasse de BackupAgent. O valor do atributo deve ser um nome de classe totalmente qualificado (como, "com.example.project.MyBackupAgent"). No entanto, como um atalho, se o primeiro caractere do nome é um ponto (por exemplo, ".MyBackupAgent"), que é anexado ao nome do pacote especificado no elemento <manifest>.

Não há padrão. O nome deve ser especificado.

android:debuggable

O aplicativo pode ou não ser depurado, mesmo quando rodando em um dispositivo em modo de usuário - "true" se ele pode ser, e "false" se não. O valor padrão é "false" .

android:description

Texto legível pelo usuário sobre a aplicação, mais longo e mais descritivo do que o rótulo de aplicação. O valor deve ser definido como uma referência a um recurso de string. Não há valor padrão.

android:enabled

O sistema Android pode ou não instanciar componentes da aplicação - "true" se puder, e "false" se não. Se o valor for "true" , de cada atributo de componente habilitado determina se o componente está habilitado ou não. Se o valor for "false", ela substitui os valores específicos do componente; todos os componentes estão desativados.

O valor padrão é "true" .

Android:ícon

Um ícone para a aplicação como um todo, e o ícone padrão para cada um dos componentes do aplicativo.

Este atributo deve ser definido como uma referência a um recurso drawable que contém a imagem (por exemplo, "@ drawable/icon"). Não há ícone padrão.

android:killAfterRestore

Se a aplicação em questão deve ser cancelada após suas configurações serem restauradas durante uma operação de restauração de todo o sistema. Operações de pacote único de restauração nunca faram com que o aplicativo seja encerrado. A restauração completa do sistema normalmente ocorrem apenas uma vez, quando o telefone é criado. Para aplicativos de terceiros, normalmente não sera preciso usar este atributo.

O padrão é true, o que significa que após a aplicação terminou de processar seus dados durante uma restauração total do sistema, ele será encerrado.

android:label

Um label legível pelo usuário para a aplicação como um todo, e um rótulo padrão para cada um dos componentes do aplicativo.

O label deve ser definido como uma referência a um recurso de string, de modo que ela pode ser localizada como outras strings na interface do usuário. No entanto, como uma conveniência quando você estiver desenvolvendo o aplicativo, ele também pode ser definida como uma sequência livre.

Android:logo

Um logotipo para a aplicação como um todo, e o logotipo padrão para as atividades.

Este atributo deve ser definido como uma referência a um recurso drawable que contém a imagem (por exemplo, "@drawable/logo"). Não há logotipo padrão.

android:manageSpaceActivity

O nome completo de uma subclasse de atividade que o sistema pode executar o gerenciamento da memória ocupada pelo aplicativo no dispositivo. A atividade deve também ser declarada com um elemento <activity>.

android:name

O nome completo de uma aplicação implementada subclasse para a aplicação. Quando o processo de aplicação é iniciada, essa classe é instanciada antes de qualquer dos componentes do aplicativo.

A subclasse é opcional, a maioria dos aplicativos não precisa de uma. Na ausência de uma subclasse, Android utiliza uma instância da classe de aplicações de base.

android:permissão

O nome de uma permissão que os clientes devem ter, a fim de interagir com o aplicativo. Este atributo é uma forma conveniente para definir uma permissão que se aplica a todos os componentes do aplicativo. Ele pode ser substituído por definir as permissões de componentes individuais.

introduzida em:

API Nível 1

<category>

sintaxe:

<category android:name="string" />

Informar em : <intent-filter>

Descrição:

Adiciona um nome de categoria para um filtro de intent.

atributos:

android:name

Nome da categoria. Categorias padrão são definidas na classe Intent como CATEGORY_ seguido de nomes constantes. O nome atribuído aqui pode ser obtidos a partir dos constantes prefixando "android.intent.category." para o nome que se segue CATEGORY_ . Por exemplo, o valor da string para CATEGORY_LAUNCHER é "android.intent.category.LAUNCHER".

Categorias personalizadas deve usar o nome do pacote como um prefixo, para garantir que eles são únicos.

introduzida em:

API Nível 1

<data>

sintaxe:

```
<data android:host="string"
    android:mimeType="string"
    android:path="string"
    android:pathPattern="string"
    android:pathPrefix="string"
    android:port="string"
    android:scheme="string" />
```

Informar em : <intent-filter>

Descrição:

Adiciona uma especificação de dados de um filtro de intent. A especificação pode ser apenas um tipo de dados (atributo mimeType), um URI, ou o tipo de dados e um URI. Um URI é especificado por atributos separados para cada uma das suas partes:

scheme://host:port/path or pathPrefix or pathPattern

Estes atributos são opcionais, mas também mutuamente dependentes: Se um esquema não é especificado para o filtro de intent, todos os atributos URI outros são ignorados. Se um host não é especificado para o filtro, o atributo porta e todos os atributos de caminho são ignorados.

Todos os elementos <data> contidos dentro do mesmo elemento <intent-filter> contribuem para o mesmo filtro. Assim, por exemplo, a seguinte especificação de filtro,

```
<intent-filter . . . >  
  <data android:scheme="something" android:host="project.example.com" />  
  . . .  
</intent-filter>
```

é equivalente a essa:

```
<intent-filter . . . >  
  <data android:scheme="something" />  
  <data android:host="project.example.com" />  
  . . .  
</intent-filter>
```

Você pode colocar qualquer número de elementos dentro de um <data> <intent-filter> para dar-lhe várias opções de dados. Nenhum de seus atributos têm valores padrão.

atributos:

Android:host

O host parte de uma autoridade URI. Este atributo é sem sentido a menos que um esquema de atributo também seja especificado para o filtro.

Nota: nome do host correspondente no Android é sensível a maiúsculos e minúsculos, ao contrário do RFC formal. Como resultado, você sempre deve especificar nomes de host usando letras minúsculas.

android:mimeType

Um tipo de mídia MIME, como image/jpeg ou audio/mpeg4-generic . O subtipo pode ser o coringa asterisco (*) para indicar que todas as correspondências de subtipos.

Nota: MIME tipo de correspondência no âmbito Android é sensível a maiúsculas, ao contrário de tipos formais RFC MIME. Como resultado, você sempre deve especificar os tipos de MIME usando letras minúsculas.

- android:path
- android: pathPrefix
- android: pathPattern

A parte do path de um URI. O atributo caminho especifica um caminho completo que é comparado com o caminho completo em um objeto Intent. O atributo PathPrefix especifica um caminho parcial, que é comparado com apenas a parte inicial do caminho no objeto de Intent. O atributo pathPattern especifica um caminho completo que é comparado com o caminho completo no objeto de Intent, mas pode conter os seguintes coringas:

- Um asterisco ('*') corresponde a uma seqüência de 0 a muitas ocorrências de caráter imediatamente anterior.
- Um asterisco seguido por ponto ("* ") corresponde a qualquer seqüência de 0 a muitos personagens.

Esses atributos são significativos somente se os atributos esquemas e host também são especificados para o filtro.

android:port

A parte da porta de um endereço URI. Este atributo só é significativo se os atributos esquemas e host também são especificados para o filtro.

android:scheme

A parte de um esquema de URI. Este é o atributo essencial mínimo para especificar um URI; pelo menos um esquema de atributo deve ser definido para o filtro, ou nenhum dos atributos URI outros são significativos.

Um esquema é especificado sem a vírgula à direita (por exemplo, HTTP, em vez de http:).

Se o filtro tem um conjunto de tipo de dados (o atributos mimeTypeX), mas nenhum esquema, o conteúdo e arquivo os esquemas são assumidos.

Nota: esquema de correspondência no âmbito Android é sensível a maiúsculas, ao contrário do RFC. Como resultado, você sempre deve especificar esquemas com letras minúsculas.

introduzida em: API Nível 1

<manifest>

syntax:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="string"
    android:sharedUserId="string"
    android:sharedUserLabel="string resource"
    android:versionCode="integer"
    android:versionName="string"
    android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
    . . .
</manifest>
```

Informer em : nenhum

deve conter: <application>

pode conter: <instrumentation>, <permission>, <permission-group>, <permission-tree>, <uses-configuration>, <uses-permission> e <uses-sdk>

Descrição:

Elemento raiz do arquivo AndroidManifest.xml. Ele deve conter um elemento <application> e especificar xmlns: android e pacotes atributos.

atributos:

xmlns: android

Define o espaço para o Android. Este atributo deve ser sempre definido como "http://schemas.android.com/apk/res/android".

package

Nome completo do pacote Java-language para a aplicação. O nome deve ser único. O nome pode conter letras maiúsculas ou minúsculas ('A' a 'Z'), números e sublinhados ("_"). No entanto, partes do nome dos pacotes individuais só podem começar com letras.

Para evitar conflitos com outros desenvolvedores, você deve usar a propriedade de domínio da Internet como base para os seus nomes de pacotes (em sentido inverso). Por exemplo, aplicativos publicados pelo Google começam com `com.google` . Você também nunca deve usar o namespace `com.example` ao publicar suas aplicações.

O nome do pacote serve como um identificador exclusivo para o aplicativo. É também o nome padrão para o processo de aplicação.

Atenção: Depois de publicar sua aplicação, você não pode mudar o nome do pacote . O nome do pacote define a identidade do seu aplicativo, por isso, se você alterá-lo, então ele é considerado uma aplicação diferente e os usuários da versão anterior não pode atualizar para a nova versão.

`android:sharedUserId`

O nome de um ID de usuário do Linux que será compartilhado com outras aplicações. Por padrão, o Android atribui a cada aplicação seu próprio ID de usuário único. No entanto, se este atributo está definido para o mesmo valor para dois ou mais pedidos, todos eles vão compartilhar o mesmo ID - desde que eles também possuam o mesmo certificado. Aplicação com o mesmo ID de usuário podem acessar dados uns dos outros e, se desejar, executado no mesmo processo.

`android: sharedUserLabel`

Um label legível pelo usuário para o ID de usuário compartilhado. O label deve ser definido como uma referência a um recurso string.

android: versionCode

Número interno de versão. Este número é usado apenas para determinar se uma versão é mais recente do que o outro, com números mais altos indicando versões mais recentes. Este não é o número da versão apresentado aos utilizadores; esse número é definido pelo atributo `versionName`.

O valor deve ser definido como um número inteiro, como "100". Você pode defini-lo como quiser, desde que cada versão sucessiva tem um número maior. Ou você pode traduzir um número de versão no formato "XY" para um inteiro através da codificação do "x" e "y" em separado nos bits inferiores e superiores. Ou ainda você pode simplesmente aumentar o número em um cada vez que uma nova versão é lançada.

android:versionName

O número da versão mostrada para os usuários. Este atributo pode ser definido como uma sequência ou como uma referência a um recurso de string. A string tem outra finalidade que não seja exibir para os usuários. O atributo `versionCode` contém o número da versão significativa usado internamente.

android:installLocation

O local de instalação padrão para o aplicativo.

As strings de palavras-chave a seguir são aceitas:

Valor	Descrição
"internalOnly"	O pedido deve ser instalado no dispositivo de armazenamento interno. Se isso for definido, o aplicativo nunca vai ser instalado no armazenamento externo. Se a

	memória interna está cheia, então o sistema não irá instalar o aplicativo. Este também é o comportamento padrão se você não definir android:installLocation .
"auto"	O aplicativo pode ser instalado no armazenamento externo, mas o sistema irá instalar o aplicativo na memória interna por padrão. Se a memória interna está cheia, então o sistema irá instalá-lo no armazenamento externo. Uma vez instalado, o usuário pode mover o aplicativo para qualquer armazenamento interno ou externo através das configurações do sistema.
"preferExternal"	A aplicação prefere ser instalado no armazenamento externo (cartão SD). Não há garantia de que o sistema irá honrar esse pedido. O aplicativo pode ser instalado no armazenamento interno se a mídia externa não está disponível ou completo. Uma vez instalado, o usuário pode mover o aplicativo para qualquer armazenamento interno ou externo através das configurações do sistema.

Nota: Por padrão, o aplicativo será instalado no armazenamento interno e não pode ser instalado no armazenamento externo, a menos que você defina esse atributo para ser "auto" ou "preferExternal".

Quando um aplicativo é instalado no armazenamento externo:

- O arquivo .apk é salvo para o armazenamento externo, mas os dados do aplicativo (como bancos de dados) ainda está guardado na memória interna do aparelho.
- O recipiente em que o arquivo .apk salvo é criptografado com uma chave que permite que o aplicativo funcione apenas no dispositivo que o instalou. Embora, vários cartões SD podem ser utilizados com o mesmo dispositivo.
- A pedido do usuário, o aplicativo pode ser movido para o armazenamento interno.

O usuário também pode pedir para mover um aplicativo da memória interna para o armazenamento externo. No entanto, o sistema não permite que o usuário mova o aplicativo para armazenamento externo se este atributo está definido para `internalOnly`, que é a configuração padrão.

introduzida em:

API Nível 1, para todos os atributos, salvo indicação em contrário na descrição do atributo.

<permission>

syntax:

```
<permission android:description="string resource"
    android:icon="drawable resource"
    android:label="string resource"
    android:name="string"
    android:permissionGroup="string"
    android:protectionLevel=["normal" | "dangerous" |
        "signature" | "signatureOrSystem"] />
```

Informar em : <manifest>

Descrição:

Declara uma permissão de segurança que pode ser usada para limitar o acesso a componentes específicos ou características de aplicações.

atributos:

`android:Description`

Descrição da permissão legível ao usuário, mais longo e mais informativo do que o label. Ela pode ser exibida para explicar a permissão para

o usuário - por exemplo, quando o usuário é perguntado se deseja conceder a permissão para outro aplicativo.

Este atributo deve ser definido como uma referência a um recurso de string.

`android:icon`

Uma referência a um recurso drawable para um ícone que representa a permissão.

`android:label`

Um nome para a autorização, que pode ser exibido para os usuários.

Como uma conveniência, o label pode ser diretamente definido como uma string, enquanto você está desenvolvendo o aplicativo. No entanto, quando a aplicação está pronto a ser publicado, ele deve ser definido como uma referência a um recurso de string, de modo que ela pode ser localizada como outras strings na interface de utilizador.

`android:name`

O nome da permissão. Este é o nome que será usado no código para se referir à permissão - por exemplo, em um elemento `<uses-permission>` e as permissões de componentes de aplicação.

O nome deve ser único, por isso deve usar o estilo Java de escopo - por exemplo, " com.example.project.PERMITTED_ACTION ".

`android: permissionGroup`

Atribui essa permissão a um grupo. O valor deste atributo é o nome do grupo, a qual deve ser declarado com o elemento `<permission-group>` da

aplicação. Se este atributo não for definido, a permissão não pertence a um grupo.

android:ProtectionLevel

Caracteriza o risco potencial implícito na permissão e indica o procedimento caso algo ocorra, o sistema deve seguir para determinar se deve ou não conceder a permissão solicitada por um aplicativo. O valor pode ser definido como um dos seguintes textos:

Valor	Significado
"standard"	O valor padrão. A permissão de baixo risco que dá acesso ao solicitar aplicativos isolados em nível de aplicativo , com o mínimo de risco para outras aplicações, o sistema ou o usuário. O sistema automaticamente concede esse tipo de permissão para um requerimento solicitando a instalação, sem pedir a aprovação explícita do usuário (embora o usuário sempre tem a opção de rever essas permissões antes de instalar.
"dangerous"	A permissão de maior risco dada a um requerimento solicitando acesso aos dados particulares do usuário ou controle sobre o dispositivo que pode impactar negativamente o usuário. Porque este tipo de permissão apresenta risco potencial, o sistema pode não concede automaticamente para o aplicativo solicitante. Por exemplo, as permissões perigosas solicitados por um aplicativo pode ser exibida para o usuário e requerem confirmação antes de prosseguir, ou alguma outra abordagem pode ser tomada para evitar que o usuário acesse automaticamente o sistema, permitindo o uso de tais instalações.
"signature"	A permissão que o sistema concede somente se o aplicativo solicitante possui assinatura com o mesmo certificado que o aplicativo que declarou a permissão. Se os certificados correspondem, o sistema automaticamente

concede a permissão sem notificar o usuário ou pedir aprovação explícita do usuário.

"signatureOrSystem " A permissão que o sistema concede somente para aplicativos que estão na imagem do sistema Android ou que são assinados com o mesmo certificado que o aplicativo que declarou a permissão. Por favor, evite usar esta opção. A permissão " signatureOrSystem " é usada para certas situações especiais em vários fornecedores que têm aplicações construídas em uma imagem do sistema e a necessidade de partilhar recursos específicos explicitamente, porque eles estão sendo construídas em conjunto.

introduzida em:

API Nível 1

<provider>

syntax:

```
<provider android:authorities="list"
    android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:grantUriPermissions=["true" | "false"]
    android:icon="drawable resource"
    android:initOrder="integer"
    android:label="string resource"
    android:multiprocess=["true" | "false"]
    android:name="string"
    android:permission="string"
    android:process="string"
    android:readPermission="string"
    android:syncable=["true" | "false"]
    android:writePermission="string" >
```

. . .

</provider>

Informar em : <application>

pode conter: <meta-data>, <grant-uri-permission> e <path-permission>

Descrição:

Declara um componente provedor de conteúdo. Um provedor de conteúdo é uma subclasse de `ContentProvider` que fornece acesso estruturado aos dados geridos pela aplicação. Todos os provedores de conteúdo em sua aplicação devem ser definidos em um elemento <provider> no arquivo de manifesto, caso contrário, o sistema não tem conhecimento delas e não as executa.

Você só declara provedores de conteúdo que fazem parte de sua aplicação. Os provedores de conteúdo em outras aplicações que você usa em seu aplicativo não devem ser declarado.

O sistema Android armazena referências para os provedores de conteúdo de acordo com uma string de autorização, parte do provedor de conteúdo URI . Por exemplo, suponha que você queira acessar um provedor de conteúdo que armazena informações sobre os profissionais de saúde. Para fazer isso, você chama o método `ContentResolver.query()`, que, entre outros argumentos leva um URI que identifica o provedor:

```
content://com.example.project.healthcareprovider/nurses/rn
```

O conteúdo:scheme identifica o URI como um URI conteúdo apontando para um provedor de conteúdo do Android. A autorização `com.example.project.healthcareprovider` identifica o próprio prestador, o sistema Android procura a autorização em sua lista de fornecedores conhecidos. A substring `nurses/RN` é um caminho, que o provedor de conteúdo pode utilizar para identificar subconjuntos de dados do provedor.

Observe que quando você define o seu provedor no elemento `<provider>`, você não inclui o regime ou o caminho no argumento `android:name`, somente a autorização.

atributos:

`android:authorities`

Uma lista de uma ou mais autoridades de URI que identificam os dados oferecidos pelo provedor de conteúdo. Múltiplas autoridades estão listados separando seus nomes com um ponto e vírgula. Para evitar conflitos, os nomes de autoridades devem usar uma convenção de nomenclatura Java (como `com.example.provider.cartoonprovider`). Normalmente, é o nome da subclasse `ContentProvider` que implementa o provedor. Não há padrão. Pelo menos uma autoridade deve ser especificada.

`android:enabled`

O provedor de conteúdo pode ou não ser instanciado pelo sistema - "true" se pode ser, e "false" se não. O valor padrão é "true".

O elemento `<application>` tem seu próprio atributo `enabled` que se aplica a todos os componentes do aplicativo, incluindo provedores de conteúdo. Os atributos `<application>` e `<provider>` devem ambos ser "true" (já que ambos são por padrão) para o provedor de conteúdo para ser habilitado.

Se um é "false", o provedor está desativado, ele não pode ser instanciado.

`android:exported`

Se o provedor de conteúdo está disponível para outros aplicativos :

- true : O fornecedor está disponível para outros aplicativos. Qualquer aplicativo pode usar URI do provedor de conteúdo, sem prejuízo das permissões especificadas para o provedor.
- falsa : O fornecedor não está disponível para outros aplicativos. Set `android:exported = "false"` para limitar o acesso ao fornecedor para suas aplicações. Somente os aplicativos que têm o mesmo ID do usuário (UID) como o provedor terá acesso a ele.

`android:icon`

Um ícone representando o provedor de conteúdo. Este atributo deve ser definido como uma referência a um recurso drawable contendo a definição da imagem. Se não estiver definido, o ícone especificado para a aplicação como um todo é usado.

`android:label`

Um label legível pelo usuário para o conteúdo fornecido. Se este atributo não for definido, o rótulo definido para o aplicativo como um todo é usado.

O label deve ser definido como uma referência a um recurso de string, de modo que ela pode ser localizada como outras strings na interface do usuário.

`android:multiprocess`

Uma instância do provedor de conteúdo pode ou não ser criado em cada processo cliente - "true" se as instâncias pode ser executado em vários processos, e "false" se não. O valor padrão é "false".

Normalmente, um provedor de conteúdo é instanciado no processo da aplicação que o definiu. No entanto, se este parametro é definido como "true", o sistema pode criar uma instância em todo o processo, onde há um

cliente que quer interagir com ele, evitando assim a sobrecarga de comunicação entre processos.

android:name

Nome da classe que implementa o provedor de conteúdo, uma subclasse de `ContentProvider`. Este deve ser um nome de classe totalmente qualificado (como, "com.example.project.TransportationProvider"). No entanto, como um atalho, se o primeiro caractere do nome é um ponto, ela será anexada ao nome do pacote especificado no elemento `<manifest>`.

Não há padrão. O nome deve ser especificado.

android:permission

O nome de uma permissão que os clientes devem ter para ler ou gravar dados do provedor de conteúdo. Este atributo é uma maneira conveniente de definir uma única permissão para leitura e escrita. No entanto, os atributos `readPermission` e `writePermission` têm precedência sobre esta. Se o atributo `readPermission` também é definido, ele controla o acesso para consultar o provedor de conteúdo. E se o atributo `writePermission` é definido, ele controla o acesso para modificar os dados do provedor.

Android:process

Nome do processo em que o provedor de conteúdo deve ser executado. Normalmente, todos os componentes de um aplicativo executado no processo padrão criado para a aplicação. Ela tem o mesmo nome que o pacote da aplicação. O elemento `<application>` pode definir um padrão diferente de todos os componentes. Mas cada componente pode substituir o padrão com o seu próprio processo, permitindo que você espalhe sua aplicação em vários processos.

Se o nome atribuído a esse atributo começa com dois pontos (':'), um novo processo, privado para a aplicação, é criado quando é necessário e a atividade é executada nesse processo. Se o nome do processo começa com um caractere minúsculo, a atividade será executado em um processo global de mesmo nome, desde que tenha permissão para fazê-lo. Isto permite que os componentes em diferentes aplicações para compartilhar um processo, reduzindo o uso de recursos.

android: readPermission

A permissão que os clientes devem ter para consultar o provedor de conteúdo.

android: writePermission

A permissão que os clientes devem ter para fazer alterações aos dados controlados pelo provedor de conteúdo.

introduzida em:

API Nível 1

<service>

syntax:

```
<service android:enabled=["true" | "false"]
    android:exported=["true" | "false"]
    android:icon="drawable resource"
    android:isolatedProcess=["true" | "false"]
    android:label="string resource"
    android:name="string"
    android:permission="string"
    android:process="string" >
    . . .
</service>
```

Informar em : <application>

pode conter : <intent-filter> e <meta-data>

Descrição:

Declara um serviço como um dos componentes do aplicativo. Ao contrário de atividades, serviços carecem de uma interface visual. Eles são usados para implementar operações de longa duração de fundo ou uma rica API de comunicação que pode ser chamado por outros aplicativos.

Todos os serviços devem ser representados por elementos <service> no arquivo de manifesto. Os que não são declarados não serão visto pelo sistema e nunca serão executados.

atributos:

android:enabled

O serviço pode ou não ser instanciado pelo sistema - "true" se pode ser, e "false" se não. O valor padrão é "true".

O elemento <application> tem seu próprio atributo que se aplica a todos os componentes de aplicativos, incluindo serviços. Os atributos <application> e <service> devem ambos ser "true" (já que ambos são por padrão) para o serviço a ser habilitado. Se um é "false", o serviço está desativado, ele não pode ser instanciado.

android:exported

Os componentes de outras aplicações podem ou não invocar o serviço ou interagir com ela - "true" se eles podem, e "false" se não. Quando o valor é "false", somente os componentes da mesma aplicação ou aplicações com o mesmo ID de usuário pode iniciar o serviço ou se ligam a ele.

O valor padrão depende se o serviço contém filtros de intents. A ausência de filtros significa que ele pode ser invocado apenas especificando o nome da classe exata. Isso implica que o serviço é destinado apenas para uso interno da aplicação (desde que os outros não sabem o nome da classe). Portanto, neste caso, o valor padrão é "false". Por outro lado, a presença de pelo menos um filtro implica que o serviço é destinado para uso externo, de modo que o valor padrão é "true".

Este atributo não é o único meio para limitar a exposição de um serviço para outras aplicações. Você também pode usar uma permissão para limitar as entidades externas que podem interagir com o serviço.

`android:icon`

Um ícone que representa o serviço. Este atributo deve ser definido como uma referência a um recurso drawable contendo a definição da imagem. Se não estiver definido, o ícone especificado para a aplicação como um todo é usada.

Ícone do serviço - se definido ou pelo elemento `<application>` - é também o ícone padrão para todos os filtros do serviço intent.

`android:isolatedProcess`

Se verdadeiro, este serviço será executado sob um processo especial que é isolado do resto do sistema. A única comunicação com ele possui é através da API Service.

`android:label`

Um nome para o serviço que pode ser exibido para os usuários. Se este atributo não for definido, o label definido para o aplicativo como um todo é usada.

Label do serviço - se definido ou pelo elemento <application> - é também o rótulo padrão para todos os filtros de intent do serviço.

O label deve ser definido como uma referência a um recurso de string, de modo que ele pode ser localizada como outras strings na interface de usuário.

android:name

Nome do serviço da subclasse que implementa o serviço. Este deve ser um nome de classe totalmente qualificado (como, "com.example.project.RoomService"). No entanto, como um atalho, se o primeiro caractere do nome é um ponto (por exemplo, ".Roomservice"), que é anexado ao nome do pacote especificado no elemento <manifest>.

Depois de publicar sua aplicação, você não deve alterar este nome (a menos que você tenha definido android:exportado = "false").

Não há padrão. O nome deve ser especificado.

android:permission

O nome de uma permissão que uma entidade deve ter para executar o serviço ou se ligam a ela. Se um chamador de StartService(), bindService(), ou StopService(), não foi concedido essa permissão, o método não irá funcionar e o objeto Intent não será entregue para o serviço.

Se este atributo não for definido, a permissão definida pelo elemento <application> no atributo permissão se aplica ao serviço. Se nem o atributo é definido, o serviço não está protegido por uma permissão.

O nome do processo em que o serviço é executado. Normalmente, todos os componentes de um aplicativo é executado no processo padrão criado para a aplicação. Ela tem o mesmo nome que o pacote da aplicação. O

elemento <application> no processo pode definir um padrão diferente de todos os componentes. Mas componente pode substituir o padrão com o seu próprio processo, permitindo que você possa espalhar a sua aplicação em vários processos.

Se o nome atribuído a esse atributo começa com dois pontos (':'), um novo processo, privado para a aplicação, é criado quando é necessário e o serviço é executado nesse processo. Se o nome do processo começa com um caractere minúsculo, o serviço será executado em um processo global de mesmo nome, desde que tenha permissão para fazê-lo. Isto permite que os componentes em diferentes aplicações para compartilhar um processo, reduzindo o uso de recursos.

introduzida em:

API Nível 1

Recebendo Mensagens

Um mensagem fornece feedback simples sobre uma operação em uma pequena janela. Ele só enche a quantidade de espaço necessário para a mensagem e a atividade atual permanece visível e interativa. Por exemplo, de sair de um e-mail antes de enviá-lo aciona uma mensagem "Rascunho salvo" para que você saiba que pode continuar editando depois. mensagens desaparecem automaticamente depois de um tempo limite.



Se a resposta do usuário para uma mensagem de status é necessária, considere a utilização de uma Notificação .

O Básico

Primeiro, instanciar um objeto mensagem com um método `makeText()`. Este método utiliza três parâmetros: a aplicação de contexto, a mensagem de texto, e a duração da mensagem. Ele retorna um objeto mensagem inicializado corretamente. É possível apresentar a mensagem com `show()`, como mostrado no exemplo a seguir:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

Este exemplo demonstra tudo o que precisa para a maioria das mensagens. Raramente você deve precisar de mais nada. Você pode, no entanto, quer posicionar a mensagem diferente ou até mesmo usar o seu próprio layout em vez de uma simples mensagem de texto. As seguintes seções descrevem como você pode fazer essas coisas.

Você também pode nesclar seus métodos com as mensagens, como esta:

```
Toast.makeText(context, text, duration).show();
```

Posicionar o sua mensagem

Uma mensagem padrão aparece na parte inferior da tela, centralizado horizontalmente. Você pode alterar esta posição com o método

setGravity(int, int, int). Este aceita três parâmetros: a gravidade constante, um offset da posição x, e um offset da posição y.

Por exemplo, se você decidir que a mensagem deve aparecer no canto superior esquerdo, você pode definir a gravidade como esta:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

Se quiser deslocar a posição para a direita, aumente o valor do segundo parâmetro. Para empurrá-lo para baixo, aumente o valor do último parâmetro.

Criando uma tela de mensagem personalizado

Se uma mensagem de texto simples não é suficiente, você pode criar um layout personalizado para a sua mensagem. Para criar um layout personalizado, definir um layout View, em XML ou no código do aplicativo, e passar a raiz do objeto view para o setContentView.

Por exemplo, você pode criar o layout para a mensagem visível na imagem à direita, com o seguinte XML :

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/toast_layout_root"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="8dp"
    android:background="#DAAA"
    >
    <ImageView android:src="@drawable/droid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```

        android:layout_marginRight="8dp"
    />
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="#FFF"
    />
</LinearLayout>

```

Note-se que a identificação do elemento LinearLayout é "toast_layout". Você deve usar essa identificação para ahustar o layout do XML, como mostrado aqui:

```

LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
    (ViewGroup) findViewById(R.id.toast_layout_root));

TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("This is a custom toast");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();

```

Primeiro, recuperar o LayoutInflater com getLayoutInflater() (ou getSystemService()), e depois inflar o layout do XML usando inflate(int, ViewGroup) . O primeiro parâmetro é o ID do recurso layout e a segunda é a exibição raiz. Você pode usar esse layout inflado para encontrar mais objetos de Visualização no layout, agora capturar e definir o conteúdo para os elementos ImageView e TextView. Finalmente, crie uma nova mensagem com Toast(Context) e definir algumas propriedades da mensagem, como a gravidade e duração. Em seguida, chamar setView(View) e passá-lo a

disposição inflada. Agora você pode exibir a mensagem com o layout personalizado chamando `show()`.

Provedores de conteúdo

Os provedores de conteúdo gerenciam o acesso a um conjunto estruturado de dados. Eles encapsulam os dados e proporcionam mecanismos para a definição de segurança dos dados. Os provedores de conteúdo são a interface padrão que conecta dados em um processo com código em execução em outro processo.

Quando você quiser acessar dados em um provedor de conteúdo, você usa o objeto `ContentResolver` em seu aplicativo de contexto para se comunicar com o fornecedor como um cliente. O objeto `ContentResolver` se comunica com o objeto do provedor, uma instância de uma classe que implementa `ContentProvider`. O objeto do provedor recebe solicitações de dados de clientes, realiza a ação solicitada, e retorna os resultados.

Você não precisa desenvolver seu próprio provedor, se você não pretende compartilhar seus dados com outras aplicações. No entanto, você precisa de seu próprio provedor para fornecer sugestões de pesquisa personalizados em seu próprio aplicativo. Você também precisa de seu próprio provedor se você quiser copiar e colar dados complexos ou arquivos de seu aplicativo para outras aplicações.

Android em si inclui provedores de conteúdo que gerenciam dados, como áudio, vídeo, imagens e informações de contato pessoal. Você pode ver alguns deles listados na documentação de referência para o pacote `android.provider`. Com algumas restrições, esses provedores são acessíveis a qualquer aplicativo Android.

Os seguintes tópicos descrevem os provedores de conteúdo em mais detalhes:.

Provedor Básico de conteúdo

Um provedor de conteúdo gerencia o acesso a um repositório central de dados. Um provedor é parte de um aplicativo Android, o que muitas vezes fornece a sua própria interface do usuário para trabalhar com os dados. No entanto, os provedores de conteúdo são principalmente destinados a ser utilizados por outras aplicações, que acessam o provedor usando um objeto cliente e provedor. Juntos, provedores e clientes do provedor oferecer uma interface consistente padrão de dados que também lida com a comunicação entre processos e acesso seguro aos dados.

Este tópico descreve os conceitos básicos do seguinte:

Como provedores de conteúdo trabalhar.

A API utiliza recuperar dados de um provedor de conteúdo.

A API que você usar para inserir, atualizar ou excluir dados em um provedor de conteúdo.

Outros recursos da API que facilitam o trabalho com os fornecedores.

Visão global

Um provedor de conteúdos apresenta dados para aplicações externas, como sendo uma ou mais tabelas que são semelhantes aos das tabelas encontradas em uma base de dados relacional. A linha representa uma instância de algum tipo de dados coletado do provedor, e cada linha da coluna representa uma peça individual de dados coletados formando uma informação completa.

Por exemplo, um provedor de embutidos na plataforma Android compõe o dicionário de dados do usuário, este armazena as grafias de palavras não-padrão que o usuário deseja manter. A tabela 1 ilustra o que os dados podem aparecer na tabela deste prestador:

Tabela 1: Exemplo de tabela de dicionário do usuário.

palavra	App ID	freqüência	localidade	_ID
MapReduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
miniaplicativo	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	pt_PT	5

Na tabela 1, cada linha representa uma instância de uma palavra que não pode ser encontrado em um dicionário comum. Cada coluna representa alguns dados da palavra, como por exemplo o local onde foi encontrada pela primeira vez. Os cabeçalhos de coluna são os nomes das colunas que são armazenadas no provedor. Para se referir ao local de uma linha, você se refere a sua localidade coluna. Para este provedor, a coluna `_ID` serve como uma "chave primária" que o provedor gera automaticamente.

Nota: Um fornecedor não é obrigado a ter uma chave primária, e não é obrigado a usar `_id` como o nome da coluna de uma chave primária, se estiver presente. No entanto, se você deseja vincular dados de um provedor para um `ListView`, um dos nomes de coluna tem que ser `_id`. Este requisito é explicada em mais detalhe na secção Resultados resultados de consulta.

Acessando um provedor

Um aplicativo acessa os dados de um provedor de conteúdo com um `ContentResolver` objeto cliente. Este objeto tem métodos que chamam métodos com nomes idênticos no objeto do provedor, uma instância de uma das subclasses concretas de `ContentProvider`. Os `ContentResolver` métodos fornecem a base "CRUD" (criar, recuperar, atualizar e excluir) funções de armazenamento persistente.

O objeto `ContentResolver` no processo do aplicativo cliente e o `ContentProvider` objeto no aplicativo que possui o provedor automaticamente lidar com inter-processo de comunicação. `ContentProvider` também atua como

uma camada de abstração entre o seu repositório de dados e a aparência externa de dados como tabelas.

Nota: Para acessar um provedor, a sua aplicação geralmente tem que solicitar permissões específicas em seu arquivo de manifesto.

Por exemplo, para obter uma lista das palavras e suas localidades do Provedor de dicionário do usuário, você chama `ContentResolver.query()` . A consulta chama o método `ContentProvider.query()` definido pelo Provedor de dicionário do usuário. As seguintes linhas de código mostram uma chamada `ContentResolver.query()` :

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,    // The content URI of the
words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                    // Selection criteria
    mSortOrder);                       // The sort order for the returned rows
```

Tabela 2 mostra como os argumentos para consulta (`Uri`, `projeção`, `seleção`, `selectionArgs`, `SortOrder`) correspondem a uma instrução SQL `SELECT`:

Tabela 2: Query () comparado a consulta SQL.

query () argumento de	SELECIONE chave / parâmetro	Notas
Uri	DE <i>table_name</i>	Uri mapeado para a tabela no provedor chamado <i>table_name</i> .
projeção	<i>col, col, col</i> ...	projeção é uma matriz de colunas que devem ser incluídos para cada linha recuperada.
seleção	ONDE <i>col</i> = <i>valor</i>	seleção especifica os critérios para a seleção de linhas.
selectionArgs	(Equivalente Não exata. Argumentos Seleção substituir ? espaços reservados na cláusula de seleção.)	
SortOrder	ORDER BY <i>col</i> , <i>col</i> ...	SortOrder especifica a ordem em que as linhas aparecem na retornou <i>Cursor</i> .

O conteúdo URI é uma URI que identifica os dados em um provedor. Conteúdo URIs incluir o nome simbólico do provedor inteiro e um nome que aponta para uma tabela. Quando você chama um método de cliente para acessar uma tabela em um provedor, a URI para a tabela é um dos argumentos.

Nas linhas de código anterior, a constante CONTENT_URI contém o URI conteúdo do dicionário do usuário "palavras" da tabela. O objeto ContentResolver analisa a autoridade da URI, e usa para "resolver" o provedor, comparando a autoridade de uma tabela do sistema de provedores conhecidos. O ContentResolver pode despachar os argumentos de consulta para o provedor correto.

O ContentProvider usa a parte do caminho da URI para escolher a tabela de acesso. Um provedor geralmente tem um caminho para cada tabela que possui.

Nas linhas de código anterior, o URI completo para o "palavras" tabela é:

```
content://user_dictionary/words
```

onde o user_dictionary string é autoridade do provedor, e a string words é o caminho da tabela. A sequência de content:// está sempre presente, e identifica isso como um conteúdo de URI.

Muitos provedores permitem acessar uma única linha em uma tabela, acrescentando um valor de ID para o final da URI. Por exemplo, para recuperar uma linha cujo _ID é 4 de dicionário do usuário, você pode usar este URI:

```
Uri singleUri = ContentUris.withAppendedId(UserDictionary.Words.CONTENT_URI,4);
```

Muitas vezes usa-se valores de ids quando você pega um conjunto de linhas e depois quer atualizar ou excluir um deles.

Recuperando dados do provedor

Esta seção descreve como recuperar dados de um provedor, usando o provedor Dicionário do usuário como um exemplo.

Por uma questão de clareza, os trechos de código desta seção de chamada a ContentResolver.query() no "segmento interface do usuário". No código atual, no entanto, você deve fazer consultas de forma assíncrona em um segmento separado. Uma maneira de fazer isso é usar a classe CursorLoader, o qual é descrito em mais detalhe no Loaders. Também, as linhas de código são só fragmentos, eles não demonstram uma aplicação completa.

Para recuperar dados de um provedor, siga estas etapas básicas:

Solicitar a permissão de acesso de leitura para o provedor.

Definir o código que envia uma consulta para o provedor.

Pedir permissão de acesso de leitura

Para recuperar dados de um provedor, seu aplicativo precisa "permissão de acesso de leitura" do provedor. Você não pode pedir essa permissão em tempo de execução, em vez disso, você tem que especificar que você precisa dessa permissão em seu manifesto, usando o elemento `<uses-permission>` e o nome da permissão exata definida pelo provedor. Quando você especificar esse elemento em seu manifesto, você de fato está "pedindo" essa permissão para a sua aplicação. Quando os usuários instalar o aplicativo, eles implicitamente conceder este pedido.

Para encontrar o nome exato da permissão de acesso de leitura para o provedor que você está usando, bem como os nomes para outras permissões de acesso utilizados pelo prestador, procure na documentação do fornecedor.

O Provedor de dicionário do usuário define a permissão `android.permission.READ_USER_DICTIONARY` em seu arquivo de manifesto, portanto, uma aplicação que quer ler o prestador deve solicitar essa permissão.

Criando a consulta

O próximo passo na recuperação de dados de um fornecedor é a construção de uma consulta. Este primeiro trecho define algumas variáveis para acessar o provedor Dicionário Usuário:

```
// A "projection" defines the columns that will be returned for each row
String[] mProjection =
{
```

```

        UserDictionary.Words._ID,    // Contract class constant for the _ID
column name
        UserDictionary.Words.WORD,  // Contract class constant for the word
column name
        UserDictionary.Words.LOCALE // Contract class constant for the
locale column name
    };

    // Defines a string to contain the selection clause
    String mSelectionClause = null;

    // Initializes an array to contain selection arguments
    String[] mSelectionArgs = {""};

```

O trecho a seguir mostra como usar `ContentResolver.query()`, usando o provedor Dicionário do usuário como um exemplo. A consulta de cliente provedor é semelhante a uma consulta SQL, e que contém um conjunto de colunas para retornar, um conjunto de critérios de seleção, e uma ordem de classificação.

O conjunto de colunas que a consulta deve retornar é chamado de projeção (a variável `mProjection`).

A expressão que especifica as linhas para recuperar é dividida em uma cláusula de seleção e os argumentos de seleção. A cláusula de seleção é uma combinação de expressões lógicas e Boolean, nomes de colunas e valores (a variável `mSelectionClause`). Se você especificar o parâmetro substituível ? vez de um valor, o método de consulta recupera o valor da matriz argumentos seleção (a variável `mSelectionArgs`).

No trecho seguinte, se o usuário não digitar uma palavra, a cláusula de seleção está definido como nulo, e a consulta retorna todas as palavras do provedor. Se o usuário digitar uma palavra, a cláusula de seleção está definido

para `UserDictionary.Words.WORD + "=?"` e o primeiro elemento da matriz de argumentos de seleção é definido pelas palavras que o usuário digita.

```
/*
 * This defines a one-element String array to contain the selection
argument.
 */
String[] mSelectionArgs = {""};

// Gets a word from the UI
mSearchString = mSearchWord.getText().toString();

// Remember to insert code here to check for invalid d or malicious input.

// If the word is the empty string, gets everything
if (TextUtils.isEmpty(mSearchString)) {
    // Setting the selection clause to null will return all words
    mSelectionClause = null;
    mSelectionArgs[0] = "";

} else {
    // Constructs a selection clause that matches the word that the user
entered.
    mSelectionClause = UserDictionary.Words.WORD + " = ?";

    // Moves the user's input string to the selection arguments.
    mSelectionArgs[0] = mSearchString;

}

// Does a query against the table and returns a Cursor object
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI, // The content URI of the
words table
```

```

        mProjection,                // The columns to return for each row
        mSelectionClause            // Either null, or the word the user
entered
        mSelectionArgs,            // Either empty, or the string the user
entered
        mSortOrder);              // The sort order for the returned rows

```

// Some providers return null if an error occurs, others throw an exception

```
if (null == mCursor) {
```

```
    /*
```

```
        * Insert code here to handle the error. Be sure not to use the cursor!
```

You may want to

```
        * call android.util.Log.e() to log this error.
```

```
        *
```

```
    */
```

```
// If the Cursor is empty, the provider found no matches
```

```
} else if (mCursor.getCount() < 1) {
```

```
    /*
```

```
        * Insert code here to notify the user that the search was unsuccessful.
```

This isn't necessarily

* an error. You may want to offer the user the option to insert a new row, or re-type the

```
        * search term.
```

```
    */
```

```
} else {
```

```
    // Insert code here to do something with the results
```

```
}
```

Esta consulta é semelhante à instrução SQL:

```
SELECT _ID, word, locale FROM words WHERE word = <userinput>  
ORDER BY word ASC;
```

Nesta instrução SQL, os nomes das colunas reais são usados em vez de constantes classe de contrato.

Proteção contra entrada não premitidas

Se os dados gerenciados pelo provedor de conteúdo está em um banco de dados SQL, incluindo dados externos não adquiridos por instruções SQL que pode levar a proteção do SQL.

Considere esta cláusula de seleção:

```
// Constructs a selection clause by concatenating the user's input to the  
column name
```

```
String mSelectionClause = "var = " + mUserInput;
```

Se você fizer isso, você está permitindo que o usuário concatenar SQL malicioso em sua instrução SQL. Por exemplo, o usuário poderia entrar "nada; DROP TABLE *;" para mUserInput , o que resultaria na cláusula seleção var = nada; DROP TABLE *; . Uma vez que a cláusula de seleção é tratada como uma instrução SQL, isso pode levar o prestador a apagar todas as tabelas no banco de dados SQLite subjacente.

Para evitar este problema, utilize uma cláusula de seleção que utiliza ? como um parâmetro substituível e uma matriz separada dos argumentos de seleção. Quando você fizer isso, a entrada do usuário é vinculado diretamente à consulta, em vez de ser interpretada como parte de uma instrução SQL. Porque não é tratado como SQL, a entrada do usuário não pode utilizar um SQL malicioso. Em vez de usar concatenação de incluir a entrada do usuário, usar esta cláusula de seleção:

```
// Constructs a selection clause with a replaceable parameter
```

```
String mSelectionClause = "var = ?";
```

Defina-se a matriz de argumentos de seleção como esta:

```
// Defines an array to contain the selection arguments  
String[] selectionArgs = {""};
```

Colocar um valor na matriz argumentos seleção como esta:

```
// Sets the selection argument to the user's input  
selectionArgs[0] = mUserInput;
```

A cláusula de seleção que usa ? como um parâmetro substituível e uma série de argumentos matriz de seleção são a maneira preferida para especificar uma seleção, mesmo que o fornecedor não é baseado em um banco de dados SQL.

Exibindo resultados da consulta

O método `ContentResolver.query()` de cliente sempre retorna um `Cursor` contendo as colunas especificadas pela projeção da consulta para as linhas que correspondem aos critérios da consulta. O objeto `Cursor` fornece acesso de leitura aleatória para as linhas e colunas que contém. Usando métodos do `Cursor`, você pode interar sobre as linhas nos resultados, determinar o tipo de dados de cada coluna, obter os dados de uma coluna, e examinar outras propriedades dos resultados. Alguns `Cursors` implementam atualização automaticamente o objeto quando os dados do provedor são alterados, ou métodos de disparo em um objeto que verifica quando os `Cursor` sofrem alterações, ou ambos.

Nota: Um provedor pode restringir o acesso às colunas com base na natureza do objecto que faz a consulta. Por exemplo, o Provedor de Contatos restringe o acesso de algumas colunas para sincronizar adaptadores, por isso não irá devolvê-los a uma atividade ou serviço.

Se não houverem linhas que correspondam aos critérios de seleção, o provedor retorna um Cursor para o qual Cursor.getCount() é 0 (um cursor vazio).

Se ocorrer um erro interno, os resultados da consulta dependem do provedor em particular. Ele pode escolher retornar nulo, ou pode enviar uma exceção .

Uma vez que um cursor é uma "lista" de linhas, uma boa maneira de exibir o conteúdo de um cursor é associá-lo a um ListView através de um SimpleCursorAdapter .

O trecho seguinte continua o código do trecho anterior. Ele cria um objeto SimpleCursorAdapter que contém o cursor recuperado pela consulta, e define esse objeto para ser o mostrado em um ListView :

```
// Defines a list of columns to retrieve from the Cursor and load into an  
output row
```

```
String[] mWordListColumns =  
{  
    UserDictionary.Words.WORD, // Contract class constant containing  
the word column name  
    UserDictionary.Words.LOCALE // Contract class constant containing  
the locale column name  
};
```

```
// Defines a list of View IDs that will receive the Cursor columns for each  
row
```

```
int[] mWordListItems = { R.id.dictWord, R.id.locale};
```

```
// Creates a new SimpleCursorAdapter
```

```
mCursorAdapter = new SimpleCursorAdapter(  
    getApplicationContext(), // The application's Context object
```

```

        R.layout.wordlistrow,          // A layout in XML for one row in the
ListView
        mCursor,                      // The result from the query
        mWordListColumns,             // A string array of column names in
the cursor
        mWordListItems,              // An integer array of view IDs in the
row layout
        0);                          // Flags (usually none are needed)

// Sets the adapter for the ListView
mWordList.setAdapter(mCursorAdapter);

```

Nota: Para fazer um ListView com um cursor , o cursor deve conter uma coluna chamada `_id`. Devido a isso, a consulta mostrada anteriormente recupera a coluna `_ID` para o campo "wolds" da tabela, embora a ListView não a exiba. Esta restrição também explica por que a maioria dos provedores tem uma coluna `_ID` para cada uma de suas tabelas.

Obtenção de dados de resultados da consulta

Em vez que é simples exibir os resultados da consulta, você pode usá-los para outras tarefas. Por exemplo, você pode recuperar grafias do dicionário do usuário e então procurá-los em outros provedores. Para fazer isso, você interar sobre as linhas do Cursor :

```

// Determine the column index of the column named "word"
int index = mCursor.getColumnIndex(UserDictionary.Words.WORD);

/*
 * Only executes if the cursor is valid. The User Dictionary Provider
returns null if
 * an internal error occurs. Other providers may throw an Exception
instead of returning null.
 */

```



```

        if (mCursor != null) {
            /*
             * Moves to the next row in the cursor. Before the first movement in the
            cursor, the
             * "row pointer" is -1, and if you try to retrieve data at that position you
            will get an
             * exception.
            */
            while (mCursor.moveToNext()) {

                // Gets the value from the column.
                newWord = mCursor.getString(index);

                // Insert code here to process the retrieved word.
                ...
                // end of while loop
            }
        } else {

            // Insert code here to report an error if the cursor is null or the provider
            threw an exception.
        }

```

O cursor de implementação contém vários métodos "get" para recuperar diferentes tipos de dados do objeto. Por exemplo, o trecho anterior usa `getString()`. Eles também têm um método `getType()` que retorna um valor que indica o tipo de dados da coluna.

Provedor de permissões de conteúdo

A aplicação de um provedor pode especificar permissões que outros aplicativos devem ter para acessar os dados do provedor. Estas permissões garantem que o usuário saiba os dados que uma aplicação vai tentar acessar.

Com base nos requisitos do provedor, outras aplicações solicitam as permissões necessárias para acessar o provedor. Os usuários finais informam as permissões solicitadas ao instalar a aplicação.

Se a aplicação de um provedor não especifica as permissões, então outras aplicações não têm acesso aos dados do provedor. No entanto, os componentes na aplicação do provedor sempre terão acesso completo de leitura e de escrita, independentemente das permissões especificadas.

Como observado anteriormente, o Provedor de dicionário do usuário requer a permissão `android.permission.READ_USER_DICTIONARY` para recuperar dados a partir dele. O utilizador tem o que requer a permissão `android.permission.WRITE_USER_DICTIONARY` separadamente para poder inserir, atualizar ou excluir dados.

Para obter as permissões necessárias para acessar um provedor, um aplicativo deve-se usar o elemento `<uses-permission>` em seu arquivo de manifesto. Quando o Gerenciador de Pacotes Android instala o aplicativo, o usuário deve aprovar todas as permissões pedidas a aplicação. Se o usuário aprova todos eles, o Gerenciador de Pacotes continua a instalação, se o usuário não aprová-los, Gerenciador de Pacotes aborta a instalação.

A seguir o elemento `<uses-permission>`, para pedidos de acesso de leitura para o Provedor Dicionário Usuário:

```
<uses-permission  
android:name="android.permission.READ_USER_DICTIONARY">
```

Inserindo, atualizando e excluindo dados

Da mesma forma que você recuperar dados de um provedor, você também usa a interação entre um cliente e um provedor `ContentProvider` para modificar os dados. Você chama um método `ContentResolver` com argumentos que são passados para o método correspondente da `ContentProvider`. O

provedor automaticamente lida com a comunicação de segurança e inter-processo.

Inserção de dados

Para inserir dados em um provedor, você chama o método `ContentResolver.insert()`. Esse método insere uma nova linha para o provedor e retorna um URI contendo para essa linha. Este trecho mostra como inserir uma nova palavra no Provedor de dicionário do usuário:

```
// Defines a new Uri object that receives the result of the insertion
Uri mNewUri;

...

// Defines an object to contain the new values to insert
ContentValues mNewValues = new ContentValues();
/*
 * Sets the values of each column and inserts the word. The arguments
to the "put"
 * method are "column name" and "value"
 */
mNewValues.put(UserDictionary.Words.APP_ID, "example.user");
mNewValues.put(UserDictionary.Words.LOCALE, "en_US");
mNewValues.put(UserDictionary.Words.WORD, "insert");
mNewValues.put(UserDictionary.Words.FREQUENCY, "100");
mNewUri = getContentResolver().insert(
    UserDictionary.Word.CONTENT_URI, // the user dictionary content
    URI
    mNewValues // the values to insert
);
```

Os dados para a nova linha iram para um único objeto `ContentValues`, o que é semelhante em forma a um cursor de uma linha. As colunas neste objeto não precisam ter o mesmo tipo de dados, e se você não quiser especificar um

valor, você pode definir uma coluna para nulo usando `ContentValues.putNull ()` .

O código não precisa adicionar coluna `_ID`, porque essa coluna é mantida automaticamente. O provedor atribui um valor único de `_ID` para cada linha que é adicionado. Provedores costumam usar este valor como chave primária da tabela.

O URI retornado em `newUri` identifica a linha recém-adicionada, com o seguinte formato:

```
content://user_dictionary/words/<id_value>
```

O `<id_value>` é o conteúdo da `_ID` para a nova linha. A maioria dos provedores pode detectar este tipo de URI automaticamente e depois executar a operação solicitada nessa linha particular.

Para obter o valor de `_ID` da retornou Uri , use `ContentUris.parseId ()` .

Atualizando dados

Para atualizar uma linha, você usa um `ContentValues` objeto com os valores atualizados, assim como você faz com uma inserção, e os critérios de seleção, assim como você faz com uma consulta. O método de cliente que você usa é `ContentResolver.update()` . Você só precisa adicionar valores à `ContentValues` objeto para colunas que você está atualizando. Se você quiser limpar o conteúdo de uma coluna, defina o valor para nulo .

O trecho a seguir altera todas as linhas cuja localidade tem a língua "en" para um local nulo . O valor de retorno é o número de linhas que foram atualizadas:

```
// Defines an object to contain the updated values
ContentValues mUpdateValues = new ContentValues();
```

```

// Defines selection criteria for the rows you want to update
String mSelectionClause = UserDictionary.Words.LOCALE + "LIKE ?";
String[] mSelectionArgs = {"en_%"};

// Defines a variable to contain the number of updated rows
int mRowsUpdated = 0;

...
/*
 * Sets the updated value and updates the selected words.
 */
mUpdateValues.putNull(UserDictionary.Words.LOCALE);

mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content
    URI
    mUpdateValues // the columns to update
    mSelectionClause // the column to select on
    mSelectionArgs // the value to compare to
);

```

Você também deve limpar a entrada do usuário quando você chama `ContentResolver.update()`.

Exclusão de dados

Excluir linhas é semelhante a recuperar dados de uma linha: você especifica critérios de seleção para as linhas que você deseja excluir e o método de cliente retorna o número de linhas excluídas. O trecho abaixo exclui linhas cuja termo `AppId` é "user". O método retorna o número de linhas excluídas.

```

// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";

```

```
String[] mSelectionArgs = {"user"};

// Defines a variable to contain the number of rows deleted
int mRowsDeleted = 0;
...
// Deletes the words that match the selection criteria
mRowsDeleted = getContentResolver().delete(
    UserDictionary.Words.CONTENT_URI, // the user dictionary content
    URI
    mSelectionClause // the column to select on
    mSelectionArgs // the value to compare to
);
```

Você também deve limpar a entrada do usuário quando você chama `ContentResolver.delete()`.

Tipos de dados do provedor

Os provedores de conteúdo podem oferecer muitos tipos de dados diferentes. O Provedor de dicionário do usuário oferece apenas o texto, mas os provedores também podem oferecer os seguintes formatos:

- número inteiro (int)
- inteiro longo (long)
- ponto flutuante (float)
- ponto flutuante de comprimento (double)

Outro tipo de dados que, com frequência, usamos é o objeto binário longo (BLOB) implementado como um array de 64KB bytes.

O tipo de dados para cada coluna em um provedor normalmente é listado em sua documentação. Os tipos de dados para o provedor de dicionário do usuário são listados na documentação de referência para os seus contatos

da classe `UserDictionary.Words`. Você também pode determinar o tipo de dados, chamando `Cursor.getType()` .

Os provedores também manter um MIME de informações sobre o tipo de dados para cada URI definida. Você pode usar as informações do tipo MIME para descobrir se seu aplicativo pode manipular dados que o provedor oferece, ou escolher um tipo de manipulação com base no tipo de MIME. Você geralmente precisa o tipo MIME quando você está trabalhando com um provedores que contém com estruturas complexas de dados ou arquivos. Por exemplo, a tabela `ContactsContract.Data` no provedor de Contatos usa tipos MIME para rotular o tipo de dados armazenados do contato em cada linha. Para obter o tipo MIME correspondente a um URI, chamar `ContentResolver.getType()` .

Formas alternativas de acesso Provider

Três formas alternativas de acesso são igualmente importantes no desenvolvimento de aplicações:

- Acesso lote : Você pode criar um lote de chamadas de acesso com métodos na classe `ContentProviderOperation`, e depois aplicá-los com `ContentResolver.applyBatch()` .
- Consultas assíncronas: Você deve fazer consultas em um segmento separado. Uma maneira de fazer isso é utilizar um objeto `CursorLoader`.
- Dados acessados através de intents : Embora você não possa enviar uma intent diretamente a um provedor, você pode enviar uma intent de uma aplicação do provedor, que normalmente é o melhor equipado para modificar os dados do provedor.

Acesso e modificação em lotes através de intents são descritos nas seções seguintes.

Acesso a dados via intents

Intents podem fornecer acesso indireto a um provedor de conteúdo. Você permite que o usuário possa acessar dados em um provedor, mesmo que a sua aplicação não tenha permissões de acesso, quer pela obtenção de um resultado de uma intent que volta de uma aplicação que tem permissões, ou ativar um aplicativo que tenha permissões e deixa o usuário fazer o trabalho nele.

Realizar acesso com permissões temporárias

Você pode acessar dados em um provedor de conteúdo, mesmo se você não tem as permissões de acesso adequadas, através do envio de uma intent de um aplicativo que tem as permissões e recebe de volta uma intent de resultado contendo a "uri" de permissões. Estes são permissões para um URI específico que duram até que a atividade que os recebe seja concluída. O aplicativo que tem permissões permanentes concede permissões temporárias através de um indicador na intent resultado:

Permissão de leitura: FLAG_GRANT_READ_URI_PERMISSION

Permissão de gravação: FLAG_GRANT_WRITE_URI_PERMISSION

Nota: Geralmente não é possível ler ou escrever para o provedor de acesso, cuja autoridade está contida na URI. O acesso é apenas para a própria URI.

Um provedor define permissões de Uri para conteúdo URIs em seu manifesto, usando o atributo android:grantUriPermission do elemento <provider>, bem como a elemento <grant-uri-permission> filho do elemento <provider>.

Por exemplo, você pode recuperar dados de um contato no provedor de Contatos, mesmo se você não tem a permissão READ_CONTACTS. Você pode querer fazer isso em um aplicativo que envia e-saudações a um contato

em seu aniversário. Em vez de solicitar `READ_CONTACTS` , o que lhe dá acesso a todos os contatos do usuário e todas as suas informações, você prefere deixar o controle de usuário que contatos são usados por seu aplicativo. Para fazer isso, você usa o seguinte processo:

O aplicativo envia uma intent que contém a ação `ACTION_PICK` e os "contatos" tipo `MIME CONTENT_ITEM_TYPE`, utilizando o método `startActivityForResult()` .

Porque esta intent corresponde ao filtro de intents para a aplicação da atividade "selection", a atividade vai vir para o primeiro plano.

Na atividade de seleção, o usuário seleciona um contato para atualizar. Quando isso acontece, a atividade de seleção chama `setResult (resultCode, intent)` para configurar uma intent que retorna à sua aplicação. A intent contém o URI do contato do usuário selecionado, e as propriedades "extras" `FLAG_GRANT_READ_URI_PERMISSION`. A atividade de seleção, então, chama `finish()` para retornar o controle de sua aplicação.

Sua atividade retorna para o primeiro plano, e o sistema de chamada sua atividade `onActivityResult()`. Esse método recebe o resultado criado pela atividade de seleção na aplicação Pessoas.

Com a URI da intent resultado, você pode ler os dados do contato do Provedor de contatos, mesmo que você não tenha pedido permissão de acesso permanente de leitura para o provedor em seu manifesto. Então você pode obter as informações de contato de aniversário ou seu endereço de e-mail e, em seguida, enviar-lhe uma saudação.

Usando outro aplicativo

Uma maneira simples de permitir que o usuário modifique os dados para o qual você não tem permissões de acesso é ativar um aplicativo que tem permissões e deixar o usuário fazer o trabalho lá.

Por exemplo, o aplicativo Calendário aceita um ACTION_INSERT intent, que permite ativar a interface do usuário do aplicativo de inserção. Você pode passar "extras" dados desta intenção, que o aplicativo usa para preencher a interface do usuário. Como os eventos recorrentes têm uma sintaxe complexa, a forma preferida de inserção de eventos no Calendário Provider é ativar a aplicação Calendário com um ACTION_INSERT e então deixar o usuário inserir o evento lá.

A criação de um Provedor de Conteúdo

Um provedor de conteúdo gerencia o acesso a um repositório central de dados. Você pode implementar um provedor como uma ou mais classes em um aplicativo Android, juntamente com elementos do arquivo de manifesto. Uma de suas classes implementa uma subclasse ContentProvider, que é a interface entre o fornecedor e outras aplicações. Embora os provedores de conteúdo sejam destinados a tornar os dados disponíveis para outras aplicações, você pode, naturalmente, ter atividades em seu aplicativo que permite ao usuário consultar e modificar os dados gerenciados pelo seu provedor.

Antes de iniciar a construção

Antes de iniciar a construção de um provedor, faça o seguinte:

Decida se você precisa de um provedor de conteúdo . Você precisa construir um provedor de conteúdo se você quer fornecer uma ou mais das seguintes características:

- Você quer oferecer dados complexos ou arquivos para outros aplicativos.
- Você deseja permitir aos usuários copiar dados complexos a partir de sua aplicação em outros aplicativos.

- Você quer fornecer sugestões de pesquisa personalizados usando a estrutura de pesquisa.
- Você não precisa de um provedor para usar um banco de dados SQLite, se o uso é inteiramente dentro de sua própria aplicação.

Em seguida, siga estes passos para construir o seu provedor:

Projetar o armazenamento bruto para seus dados. Um provedor de conteúdo oferece dados de duas maneiras:

Os dados do arquivo

Dados que normalmente estão em arquivos, como fotos, áudio ou vídeos. Armazenar os arquivos em espaço privado do seu aplicativo. Em resposta a um pedido de um arquivo de outro aplicativo, o provedor pode oferecer um identificador para o arquivo.

"Estrutura" dos dados

Dados que normalmente vão para um banco de dados, uma matriz ou estrutura semelhante. Armazenam os dados de uma forma que seja compatível com as tabelas de linhas e colunas. A linha representa uma entidade, tal como uma pessoa ou um item no inventário. Uma coluna representa alguns dados para a entidade, como nome de uma pessoa ou o preço de um item. Uma maneira comum para armazenar este tipo de dados está em um banco de dados SQLite, mas você pode usar qualquer tipo de armazenamento persistente.

Definir uma implementação concreta da classe `ContentProvider` e seus métodos necessários. Esta classe é a interface entre os dados e o resto do sistema Android.

Definir a string de autoridade do provedor, o seu conteúdo URIs, e os nomes das colunas. Se você quer uma aplicação do provedor para lidar com

as intents, também definir ações da intent, dados extras, e propriedades. Também definir as permissões que você vai precisar para aplicações que desejam acessar seus dados. Você deve considerar a definição de todos esses valores como constantes em um contrato separado de classe, mais tarde, você pode expor essa classe para outros desenvolvedores.

Adicione outras peças opcionais, tais como amostras de dados ou uma implementação do `AbstractThreadedSyncAdapter` que podem sincronizar dados entre o provedor e outros baseados em nuvem de dados.

Projetando Data Storage

Um provedor de conteúdo é a interface para os dados salvos em um formato estruturado. Antes de criar a interface, você deve decidir como armazenar os dados. Você pode armazenar os dados em qualquer forma que você quiser, e depois projetar a interface para ler e gravar os dados, se necessário.

Estas são algumas das tecnologias de armazenamento de dados que estão disponíveis no Android:

O sistema Android inclui um banco de dados SQLite que os provedores próprios do Android usam para armazenar dados orientados tabela. A classe `SQLiteOpenHelper` ajuda a criar bases de dados, e a classe `SQLiteDatabase` é a classe base para acessar bancos de dados.

Lembre-se que você não tem que usar um banco de dados para implementar o seu repositório. Um provedor aparece externamente como um conjunto de tabelas, semelhante a uma base de dados relacional, mas isto não é um requisito para a aplicação interna do provedor.

Para o armazenamento de dados em arquivos, o Android tem uma variedade de arquivos orientado a APIs. Se você está projetando um provedor

que oferece meios de dados relacionados, tais como música ou vídeos, você pode ter um provedor que combina dados da tabela e arquivos.

Para trabalhar com dados baseados em rede, usar classes no `java.net` e `android.net`. Você também pode sincronizar dados baseados em rede para um armazenamento de dados local, como um banco de dados, e então oferecer os dados como tabelas ou arquivos.

Considerações sobre o design de dados

Aqui estão algumas dicas para desenhar a estrutura do seu provedor de dados:

Dados da tabela devem ter sempre uma "chave primária" coluna que o provedor mantém como um valor numérico único para cada linha. Você pode usar esse valor para ligar a linha com linhas relacionadas em outras tabelas (usando-o como uma "chave estrangeira"). Embora você possa usar qualquer nome para esta coluna, usando `BaseColumns._ID` é a melhor escolha, porque ligando os resultados de uma consulta a um provedor de `ListView` requer que uma das colunas recuperadas seja o nome `_ID`.

Se você deseja fornecer imagens bitmap ou outras peças muito grandes de arquivo de dados, armazene os dados em um arquivo e, em seguida, forneça-os indiretamente, em vez de armazená-lo diretamente em uma tabela. Se você fizer isso, você precisa dizer a usuários do seu provedor que eles precisam usar um método `ContentResolver` de arquivo para acessar os dados.

Use o objeto tipo binário grande (BLOB) de dados para armazenar dados que variam em tamanho ou tem uma estrutura variável. Por exemplo, você pode usar uma coluna BLOB para armazenar um buffer de protocolo ou estrutura JSON.

Você também pode usar um BLOB para implementar um esquema independente de dados. Neste tipo de tabela, você define uma coluna de chave primária, uma coluna de tipo MIME e uma ou mais colunas genéricas como BLOB. O significado dos dados nas colunas BLOB é indicado pelo valor na coluna de tipo MIME. Isso permite que você armazene tipos de linhas diferentes na mesma tabela. O Provedor de contatos de "dados" tabela `ContactsContract.Data` é um exemplo de uma tabela de esquema independente.

Projetando conteúdo URIs

A URI de conteúdo é uma URI que identifica os dados em um provedor. Conteúdo URIs incluem o nome simbólico do provedor inteiro (sua autoridade) e um nome que aponta para uma tabela ou arquivo (um caminho). A parte ID opcional aponta para uma linha individual em uma tabela. Cada método de acesso de dados de `ContentProvider` tem um URI de conteúdo como argumento, o que permite determinar a tabela, linha ou arquivo de acesso.

Projetando uma autoridade

Um provedor normalmente tem uma única entidade, que serve como o seu nome Android-interno. Para evitar conflitos com outros fornecedores, você deve usar de propriedade de domínio da Internet (em sentido inverso), como a base no seu procedor de autoridade. Porque esta recomendação também é verdade para nomes de pacotes Android, você pode definir seu provedor de autoridade como uma extensão do nome do pacote que contém o provedor. Por exemplo, se o seu nome de pacote Android é `com.example.<appname>`, você deve dar o seu provedor a autoridade `com.example.<appname>.provedor`.

Projetando uma estrutura de caminho

Desenvolvedores geralmente criam conteúdo URIs da autoridade, acrescentando caminhos que apontam para tabelas individuais. Por exemplo, se você tem duas tabelas `table1` e `table2`, combinamos a autoridade do exemplo anterior e produzir a URIs `com.example.<appname>.provider/table1` e `com.example.<appname>.provider/table2`. Os caminhos não são limitados a um único segmento, e não tem de ser uma tabela para cada nível do caminho.

Manipulando conteúdo URI IDs

Por convenção, os provedores oferecem acesso a uma única linha em uma tabela, aceitando um URI com um valor de ID para a linha no final da URI. Também por convenção, os provedores correspondem ao valor de ID para a tabela `_ID` coluna, e realizam o acesso solicitado para a linha que corresponde.

Esta convenção facilita um padrão de design comum para aplicativos que acessam um provedor. O aplicativo faz uma consulta em um provedor e exibe o resultado do cursor em um `ListView` usando um `CursorAdapter`. A definição de `CursorAdapter` requer uma das colunas do cursor para ser `_id`.

O usuário, então, escolhe uma das linhas exibidas a partir da interface do usuário, a fim de olhar ou modificar os dados. O aplicativo obtém a linha correspondente ao `Cursor` apoiando o `ListView`, obtém o valor de `_ID` para esta linha, acrescenta à URI, e envia a solicitação de acesso ao provedor. O provedor pode fazer a consulta ou modificar a linha exata do usuário escolheu.

URI padrões de conteúdo

Para ajudar você a escolher qual ação a ser tomada por um URI de entrada, a API do provedor inclui a classe de conveniência `UriMatcher`, que

mapeia os conteúdos "padrões" das URI para valores inteiros. Você pode usar os valores inteiros em um interruptor de declaração que escolhe a ação desejada para o URI que correspondem a um determinado padrão.

Um padrão URI de conteúdo corresponde ao conteúdo URIs usando caracteres curinga:

* : Corresponde a uma seqüência de caracteres válidos de qualquer comprimento.

: Corresponde a uma seqüência de caracteres numéricos de qualquer comprimento.

Como um exemplo de concepção e de codificação manipulação de URI de conteúdo, considere um provedor com a autoridade com.example.app.provider que reconhece o seguinte conteúdo URIs apontando para tabelas:

content://com.example.app.provider/table1: A table called table1.

content://com.example.app.provider/table2/dataset1: A table called dataset1.

content://com.example.app.provider/table2/dataset2: A table called dataset2.

content://com.example.app.provider/table3: A table called table3

O fornecedor também reconhece estes conteúdos URIs se eles têm um ID de linha anexado a eles, como por exemplo o content://com.example.app.provider/table3/1 para a linha identificada por uma de table3 .

Os padrões de URI seguintes seriam possíveis:

content://com.example.app.provider/*

Corresponde a qualquer URI conteúdo no provedor.

`content://com.example.app.provider/table2/*:`

Corresponde a um URI de conteúdo para as tabelas DataSet1 e dataset2 , mas não corresponde o conteúdo URIs para tabela1 ou table3 .

`content://com.example.app.provider/table3/#:`

Corresponde a um URI conteúdo para linhas simples em table3 , como conteúdo `:// com.example.app.provider/table3/6` para a linha identificada por 6 .

O trecho de código a seguir mostra como os métodos em UriMatcher trabalham. Este código trata URIs para toda uma tabela diferente de URIs para uma única linha, usando o conteúdo URI padrão `://content <authority>/<caminho>` para tabelas e conteúdo `://<authority>/<caminho>/<id>` para linhas individuais.

O método `addURI()` mapeia uma autoridade e caminho para um valor inteiro. O método `match()` retorna o valor inteiro para uma URI. Um seletor de declaração escolhe entre consultar a tabela inteira, e consulta para um único registro:

```
public class ExampleProvider extends ContentProvider {
    ...
    // Creates a UriMatcher object.
    private static final UriMatcher sUriMatcher;
    ...
    /*
     * The calls to addURI() go here, for all of the content URI patterns
that the provider
     * should recognize. For this snippet, only the calls for table 3 are
shown.
     */
    ...
    /*
```

* Sets the integer value for multiple rows in table 3 to 1. Notice that no wildcard is used

* in the path

*/

```
sUriMatcher.addURI("com.example.app.provider", "table3", 1);
```

/*

* Sets the code for a single row to 2. In this case, the "#" wildcard is

used. "content://com.example.app.provider/table3/3" matches, but

* "content://com.example.app.provider/table3 doesn't.

*/

```
sUriMatcher.addURI("com.example.app.provider", "table3/#", 2);
```

...

```
// Implements ContentProvider.query()
```

```
public Cursor query(
```

```
    Uri uri,
```

```
    String[] projection,
```

```
    String selection,
```

```
    String[] selectionArgs,
```

```
    String sortOrder) {
```

...

/*

* Choose the table to query and a sort order based on the code returned for the incoming

* URI. Here, too, only the statements for table 3 are shown.

*/

```
switch (sUriMatcher.match(uri)) {
```

```
    // If the incoming URI was for all of table3
```

```
    case 1:
```

```

        if (TextUtils.isEmpty(sortOrder)) sortOrder = "_ID ASC";
        break;

// If the incoming URI was for a single row
case 2:

    /*
     * Because this URI was for a single row, the _ID value
part is
     * present. Get the last path segment from the URI; this is
the _ID value.
     * Then, append the value to the WHERE clause for the
query
    */
    selection = selection + "_ID = " + uri.getLastPathSegment();
    break;

default:
    ...
    // If the URI is not recognized, you should do some error
handling here.
    }
    // call the code to actually do the query
    }

```

Outra classe, `ContentUris`, fornece métodos de conveniência para trabalhar com o ID de parte do conteúdo de URIs. As classes `Uri` e `Uri.Builder` incluem métodos de conveniência para analisar objetos `Uri` existentes e construir novos.

Implementação da Classe `ContentProvider`

A instância `ContentProvider` gerencia o acesso a um conjunto estruturado de dados por tratamento de pedidos de outras aplicações. Todas

as formas de acesso eventualmente chamam ContentResolver, que, em seguida, chama um método concreto ContentProvider para ter acesso.

Métodos necessários

A classe abstrata ContentProvider define seis métodos abstratos que devem ser implementadas como parte de sua própria subclasse concreta. Todos esses métodos, exceto onCreate() são chamados por uma aplicação cliente que está tentando acessar seu provedor de conteúdo:

query()

Recupera dados de seu provedor. Utilize os argumentos para selecionar a tabela para consulta, as linhas e colunas para voltar, e a ordem de classificação do resultado. Retorna os dados como um cursor.

insert()

Inserir uma nova linha em seu provedor. Utilize os argumentos para selecionar a tabela de destino e para obter os valores para usar nas colunas. Retorna um URI de conteúdo para a linha recém-inserida.

update()

Atualizado em linhas existentes em seu provedor. Utilize os argumentos para selecionar a tabela e as linhas para atualizar e obter os valores de coluna atualizados. Retorna o número de linhas atualizadas.

delete()

Exclui linhas de seu provedor. Utilize os argumentos para selecionar a tabela e as linhas a serem excluídos. Retorna o número de linhas excluídas.

`getType()`

Retornar o tipo de MIME correspondente a um URI.

`onCreate()`

Inicializa o seu provedor. O sistema Android chama este método imediatamente depois que ele cria seu provedor. Observe que o provedor não é criado até um `ContentResolver` tenta acessá-lo.

Observe que esses métodos têm a mesma assinatura do identicamente nomeados nos métodos `ContentResolver`.

Sua implementação destes métodos deve conter o seguinte:

Todos esses métodos, exceto `onCreate()` podem ser chamados por vários segmentos de uma vez, então elas devem ser thread-safe.

Evite fazer operações demoradas em `onCreate()`. Adiar tarefas de inicialização até que eles sejam realmente necessários.

Embora você deva implementar esses métodos, o código não tem que fazer nada além de retornar o tipo de dados esperado. Por exemplo, você pode querer impedir que outras aplicações insiram dados em algumas tabelas.

Para fazer isso, você pode ignorar a chamada para `Insert()` e retornar 0.

Implementar o método `query()`

O método `ContentProvider.query()` deve retornar um `Cursor`, ou se não, informar uma exceção. Se você estiver usando um banco de dados SQLite, como o armazenamento de dados, você pode simplesmente devolver o cursor retornado por uma das `query()` da classe `SQLiteDatabase`. Se a consulta

não corresponde a nenhuma linha, você deve retornar um Cursor instância cujo getCount() retorna 0. Você deve retornar nulo somente se ocorreu um erro interno durante o processo de consulta.

Se você não estiver usando um banco de dados SQLite como seu armazenamento de dados, use uma das subclasses concretas de Cursor . Por exemplo, o MatrixCursor implementa um cursor em que cada linha é um array de objetos . Com essa classe, use AddRow() para adicionar uma nova linha.

Lembre-se que o sistema Android deve ser capaz de comunicar a exceção limite de processo. Android pode fazer isso para as seguintes exceções que podem ser úteis no tratamento de erros de consulta:

- IllegalArgumentException (Você pode escolher informar isso se o seu provedor recebe uma URI conteúdo inválido)
- NullPointerException

Aplicação do método de insert()

O método insert() adiciona uma nova linha à tabela apropriada, usando os valores no argumento ContentValues. Se um nome de coluna não está no ContentValues, você pode querer fornecer um valor padrão para ela ou em seu código de provedor ou no seu esquema de banco de dados.

Este método deve retornar o URI de conteúdo para a nova linha. Para construir este, acrescente a nova linha de _ID (ou outra chave primária) valor a URI da tabela de conteúdo, utilizando withAppendedId() .

Implementar o método delete()

O método delete() não tem que excluir fisicamente as linhas de seu armazenamento de dados. Se você estiver usando um adaptador de sincronização com o seu provedor, você deve considerar a marcação de uma linha excluída com um "delete" uma informação em vez de remover a linha

completamente. O adaptador de sincronização pode verificar se há registros apagados e removê-los do servidor antes de excluí-los do provedor.

Implementar o método update()

O método update() leva o mesmo argumento ContentValues usado por insert() , e a mesma selection e selectionArgs usados por excluir() e ContentProvider.query() . Isso pode permitir a reutilização de código entre esses métodos.

Implementar o método onCreate()

O sistema Android chama onCreate() quando ele inicia o provedor. Você deve executar somente as tarefas de funcionamento rápido de inicialização neste método, e adiar a criação do banco de dados e o carregamento de dados até que o provedor realmente recebe um pedido para os dados. Se você realizar tarefas longas em onCreate() , você vai retardar a inicialização do seu provedor. Por sua vez, isso vai retardar a resposta do provedor para outras aplicações.

Por exemplo, se você estiver usando um banco de dados SQLite, você pode criar um novo objeto SQLiteOpenHelper em ContentProvider.onCreate(), e então criar as tabelas SQL a primeira vez que você abrir o banco de dados. Para facilitar isso, a primeira vez que você chama getWritableDatabase(), ele automaticamente chama o método SQLiteOpenHelper.onCreate().

Os dois trechos a seguir demonstram a interação entre ContentProvider.onCreate() e SQLiteOpenHelper.onCreate(). O primeiro trecho é a implementação de ContentProvider.onCreate() :

```
public class ExampleProvider extends ContentProvider
/*
```

* Defines a handle to the database helper object. The MainDatabaseHelper class is defined

* in a following snippet.

*/

```
private MainDatabaseHelper mOpenHelper;
```

```
// Defines the database name
```

```
private static final String DBNAME = "mydb";
```

```
// Holds the database object
```

```
private SQLiteDatabase db;
```

```
public boolean onCreate() {
```

```
/*
```

* Creates a new helper object. This method always returns quickly.

* Notice that the database itself isn't created or opened

* until SQLiteOpenHelper.getWritableDatabase is called

```
*/
```

```
mOpenHelper = new SQLiteOpenHelper(  
    getContext(),    // the application context  
    DBNAME,          // the name of the database)  
    null,             // uses the default SQLite cursor  
    1                 // the version number  
);
```

```
return true;
```

```
}
```

```
...
```

```
// Implements the provider's insert method
```

```
public Cursor insert(Uri uri, ContentValues values) {
```

// Insert code here to determine which table to open, handle error-checking, and so forth


```

...
/*
    * Gets a writeable database. This will trigger its creation if it
doesn't already exist.
    *
    */
    db = mOpenHelper.getWritableDatabase();
}
}

```

O trecho seguinte é a implementação de SQLiteOpenHelper.onCreate() , incluindo uma classe auxiliar:

```

...
// A string that defines the SQL statement for creating a table
private static final String SQL_CREATE_MAIN = "CREATE TABLE "
+
    "main " +           // Table's name
    "(" +               // The columns in the table
    "_ID INTEGER PRIMARY KEY, " +
    "WORD TEXT"
    " FREQUENCY INTEGER " +
    " LOCALE TEXT );";
...
/**
    * Helper class that actually creates and manages the provider's
underlying data repository.
    */
    protected static final class MainDatabaseHelper extends
SQLiteOpenHelper {

        /*
            * Instantiates an open helper for the provider's SQLite data
repository

```

```

        * Do not do database creation and upgrade here.
        */
MainDatabaseHelper(Context context) {
    super(context, DBNAME, null, 1);
}

/*
    * Creates the data repository. This is called when the provider
attempts to open the
    * repository and SQLite reports that it doesn't exist.
    */
public void onCreate(SQLiteDatabase db) {

    // Creates the main table
    db.execSQL(SQL_CREATE_MAIN);
}
}

```

Execução ContentProvider Tipos MIME

A classe ContentProvider tem dois métodos para retornar tipos MIME:

getType()

Um dos métodos que você deve implementar para qualquer provedor.

getStreamTypes ()

Um método que você deve implementar se o seu provedor oferece arquivos.

Tipos MIME para tabelas

O método `getType()` retorna uma string no formato MIME que descreve o tipo de dados retornados pelo argumento URI. O argumento Uri pode ser um padrão em vez de uma URI específica, neste caso, você deve retornar o tipo de dados associados ao conteúdo URIs que correspondem ao padrão.

Para os tipos mais comuns de dados, tais como texto, HTML, ou JPEG, `GetType()` deve retornar o tipo MIME padrão para esses dados.

Para o índice de URIs que apontam para uma ou mais linhas de dados da tabela, `getType()` deve retornar um tipo de MIME no provedor específico do Android formato MIME:

Digite parte: `vnd`

Parte subtipo:

Se o padrão de URI é para uma única linha: `android.cursor.artigo /`

Se o padrão de URI é para mais de uma linha: `. android.cursor.dir /`

Específica do provedor parte: `. vnd <name> . <type>`

Você fornece o `<name>` e `<type>`. O valor `<name>` deve ser globalmente único, e o valor `<type>` deve ser exclusivo para o padrão URI correspondente. Uma boa opção para `<name>` é o nome da sua empresa ou alguma parte do seu aplicativo como o nome do pacote Android. Uma boa opção para o `<type>` é uma string que identifica a tabela associada com a URI.

Por exemplo, se a autoridade de um provedor é `com.example.app.provider`, e ele expõe uma tabela chamada `table1`, o tipo MIME para várias linhas em `table1` é:

`vnd.android.cursor.dir/vnd.com.example.provider.table1`

Para uma única linha de table1 , o tipo MIME é:

```
vnd.android.cursor.item/vnd.com.example.provider.table1
```

Tipos MIME para arquivos

Se o seu provedor oferece arquivos, que implementar `getStreamTypes()` . O método retorna uma string de tipos MIME para os arquivos que o seu provedor pode retornar para um URI de conteúdo. Você deve filtrar os tipos MIME que oferecem pelo argumento de tipo MIME filtro, de modo que você retorne apenas os tipos MIME que o cliente quer lidar.

Por exemplo, considere um provedor que oferece imagens de fotos como arquivos em .jpg , .png , e .gif. Se um aplicativo chama `ContentResolver.getStreamTypes()` com o filtro de string `image/*` (algo que é uma "imagem"), então os `ContentProvider.getStreamTypes()` deve retornar a matriz:

```
{ "image/jpeg", "image/png", "image/gif" }
```

Se o aplicativo só está interessado em arquivos jpg, então ele pode chamar `ContentResolver.getStreamTypes()` com a string de filtro `*/jpeg`, e `ContentProvider.getStreamTypes()` deve retornar:

```
{ "image/jpeg" }
```

Se o seu provedor não oferece qualquer dos tipos MIME solicitados na sequência de filtro, `getStreamTypes()` deve retornar nulo .

Implementando uma classe Contrato

Uma classe é um contrato público final é uma classe que contém as definições constantes para a URIs, nomes de colunas, tipos de MIME e outros meta-dados que pertencem ao provedor. A classe estabelece um contrato entre

o provedor e outras aplicações, garantindo que o provedor pode ser acessado corretamente, mesmo que haja alterações aos valores reais de URIs, nomes de coluna, e assim por diante.

Uma classe de contrato também ajuda os desenvolvedores, pois geralmente tem nomes mnemônicos para suas constantes, para que os desenvolvedores fiquem menos propensos a usar valores incorretos para nomes de coluna ou URIs. Uma vez que é uma classe, ele pode conter documentação Javadoc. Ambientes de desenvolvimento integrado como o Eclipse pode auto-completar os nomes constantes da classe contrato e Javadoc de exibição para as constantes.

Os desenvolvedores não podem acessar o arquivo a classe contrato a partir de sua aplicação, mas eles podem estaticamente compilá-lo em sua aplicação a partir de um arquivo .jar que você fornece.

A classe `ContactsContract` e suas classes aninhadas são exemplos de classes de contrato.

Permissões de execução

Todos os aplicativos podem ler ou escrever para o seu provedor, mesmo se os dados subjacente forem privado, porque, por padrão o seu provedor não tem o conjunto de permissões. Para mudar isso, definir as permissões para o seu provedor em seu arquivo de manifesto, utilizando atributos ou elementos filho do elemento `<provider>`. Você pode definir permissões que se aplicam a todo o provedor, ou para certos quadros, ou até mesmo para certos registros, ou todos os três.

Você define permissões para seu provedor com um ou mais elementos `<permission>` em seu arquivo de manifesto. Para fazer a permissão exclusiva para o seu provedor, use o escopo Java-style para o atributo `android:name`. Por exemplo, o nome da permissão de leitura `com.example.app.provider.permission.READ_PROVIDER`.

Recursos de Imagens e Strings no Android, Localização da Aplicação

Recursos de Imagens

Um recurso drawable é um conceito geral para um gráfico que pode ser desenhada na tela e que você pode recuperar com APIs como `getDrawable(int)` ou aplicar em outro recurso de XML com atributos como `android:drawable` e `android:icon`. Existem vários tipos diferentes de drawables:

Arquivo de bitmap

Um arquivo gráfico de bitmap (.png, .jpg ou .gif). Cria um `BitmapDrawable`.

Nine Patch-File

Um arquivo PNG com regiões elásticas para permitir o redimensionamento da imagem com base em conteúdo. Cria um `NinePatchDrawable`.

Lista camada

Uma Drawable que gerencia uma série de outras Drawables. Estes são desenhados a uma matriz, de modo que o elemento com o maior índice é desenhado em cima. Cria um `LayerDrawable`.

Lista de Estado

Um arquivo XML que faz referência a diferentes gráficos bitmap para estados diferentes (por exemplo, para usar uma imagem diferente quando um botão é pressionado). Cria um `StateListDrawable`.

Lista de nível

Um arquivo XML que define um drawable que gerencia um número de suplentes, cada atribuído Drawable possui um valor numérico máximo. Cria um LevelListDrawable .

Transição Drawable

Um arquivo XML que define um drawable que pode alternar entre dois recursos drawable. Cria um TransitionDrawable .

Inset Drawable

Um arquivo XML que define um drawable que insere outro drawable a uma determinada distância. Isto é útil quando a View precisa de um drawable de fundo que é menor do que os limites reais da vista.

Clipe Drawable

Um arquivo XML que define um drawable com cliques tendo outro drawable com base no valor presente do Drawable do nível atual. Cria um ClipDrawable .

Drawable escala

Um arquivo XML que define um drawable que altera o tamanho de outro Drawable base no seu valor atual. Cria um ScaleDrawable

Forma Drawable

Um arquivo XML que define uma forma geométrica, incluindo cores e gradientes. Cria um ShapeDrawable .

Bitmap

Uma imagem bitmap. Android suporta arquivos de bitmap em um três formatos: .png (preferencial), .jpg (aceitável), .gif (desanimados).

Você pode fazer referência a um arquivo de bitmap diretamente, usando o nome de arquivo como o ID do recurso, ou criar uma ID de recurso alias em XML.

Arquivo de bitmap

Um arquivo de bitmap é um arquivo .png, .jpg ou .gif. Android cria um recurso Drawable para qualquer um desses arquivos, quando você salvá-los no res/drawable/directory.

localização do arquivo:

res/drawable/nome do arquivo.png (. png, .jpg ou .gif)

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma BitmapDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

exemplo:

Com uma imagem salva no res/drawable/myimage.png , este XML disposição aplica-se a imagem de uma vista:

<ImageView


```
android:layout_height = "wrap_content"  
android:layout_width = "wrap_content"  
android:src = "@drawable/myimage" />
```

O código da aplicação a seguir recupera a imagem como um Drawable :

```
Resources res = getResources() ;  
Drawable drawable = res.getDrawable (R.drawable.myimage);
```

Bitmap XML

Um bitmap XML é um recurso definido em XML que aponta para um arquivo bitmap. O efeito é um alias para um arquivo bitmap-prima. O XML pode especificar propriedades adicionais para o bitmap, tais como composição de cores.

Nota: Você pode usar um elemento <bitmap> como um herdeiro de um elemento <item>. Por exemplo, ao criar uma lista estadual ou lista de camadas, você pode excluir o atributo android:drawable de um elemento <item> e uma coleção de <bitmap> dentro dele que define o item drawable.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma BitmapDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable/file

sintaxe:

```
<? Versão xml = "1.0" encoding = "UTF-8">
< bitmap
  xmlns: android = "http://schemas.android.com/apk/res/android"
  android: src = "@ [pacote]: drawable / drawable_resource "
  android: antialias = ["verdadeiro" | "false"]
  android: dither = ["verdadeiro" | "false"]
  android: filter = ["verdadeiro" | "false"]
  android: gravidade = ["top" | "de baixo" | "esquerda" | "direito" |
"center_vertical" |
                        "Fill_vertical" | "center_horizontal" | "fill_horizontal" |
                        "Centro" | "encher" | "clip_vertical" | "clip_horizontal"]
  android: TileMode = ["desativado" | "braçadeira" | "repeat" |
"espelho"] />
```

elementos:

<bitmap>

Define a fonte bitmap e suas propriedades.

atributos:

xmlns:android

String. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android". Isso é necessário apenas se o <bitmap> é o elemento raiz não é necessário quando o <bitmap> está aninhado dentro de um <item> .

android:src

Recurso Drawable . Obrigatório . Referência a um recurso drawable.

android: antialias

Boolean. Habilita ou desabilita antialiasing.

android: dither

Boolean. Habilita ou desabilita indecisão do bitmap se o bitmap não tem a mesma configuração de pixes da tela (por exemplo: um bitmap 8888 ARGB com 565 RGB tela).

android:filter

Boolean. Habilita ou desabilita a filtragem de bitmap. A filtragem é usada quando o bitmap é encolhido ou esticado para suavizar sua aparência.

Android:gravity

Palavra-chave. Define a gravidade para o bitmap. A gravidade indica onde posicionar o drawable no seu recipiente, se o mapa de bits é menor do que o recipiente.

Caso haja mais de um valor, separa-los por '|', os valores seguem as seguintes constantes:

Valor	Descrição
top	Coloca o objeto no topo do recipiente, não alterando o seu tamanho.
botton	Coloca o objeto no fundo do recipiente, não alterando o seu tamanho.
left	Coloca o objeto no lado esquerdo de seu recipiente, não alterando o seu tamanho.
right	Coloca o objeto no canto direito de seu recipiente, não alterando o seu tamanho.
center_vertical	Coloca objeto no centro vertical do seu recipiente, não alterando o seu tamanho.
fill_vertical	Aumenta o tamanho vertical do objecto, se necessário de modo que preenche completamente o seu recipiente.
center_horizontal	Coloca o objeto no centro horizontal do recipiente, não alterando o seu tamanho.
fill_horizontal	Aumenta o tamanho horizontal do objeto, se necessário para que ele preenche

	completamente seu recipiente.
center	Coloca o objeto no centro do recipiente nos eixos vertical e horizontal, não alterando o seu tamanho.
Fill	Aumenta o tamanho horizontal e vertical do objeto, se necessário para que ele preenche completamente seu recipiente. Este é o padrão.
clip_vertical	Opção adicional que pode ser ajustado para ter as bordas superior e/ou inferior dos limites de seu contêiner. O clipe é baseado na gravidade vertical: um clipe de gravidade de topo do bordo inferior, um clipe de gravidade do fundo do bordo de topo.
clip_horizontal	Opção adicional que pode ser configurado para ter as margens esquerda e/ ou direita dos limites do seu contêiner. O clipe é baseado na gravidade horizontal: um clipe de gravidade para a esquerda da extremidade direita, um clipe gravidade direita da borda esquerda.

android: TileMode

Palavra-chave . Define o modo de TileMode. Quando o modo lado a lado é habilitado, o bitmap é repetido. Gravidade é ignorado quando o modo lado a lado é ativado.

Deve ser um dos seguintes valores constantes:

Valor	Descrição
disabled	Não aloca o bitmap lado a lado. Este é o valor padrão.
clamp	Replica a cor da borda se o desenho chamado esta fora de seus limites originais
repeat	Repete imagem do desenho horizontalmente e verticalmente.
mirror	Repete imagem do desenho horizontalmente e verticalmente, alternando imagens de espelho para que as imagens adjacentes sempre costura.

exemplo:

```
<? Versão xml = "1.0" encoding = "UTF-8">
<XmInS          bitmap:          android          =
"http://schemas.android.com/apk/res/android"
    android: src = "@ drawable / icon"
    android: TileMode = "repeat" />
```

Nine-Patch

A NinePatch é uma imagem PNG em que você pode definir regiões elásticas com escalas Android quando o conteúdo dentro da Visão excede os limites de imagem normais. Você normalmente atribui este tipo de imagem como fundo de uma ideia que tem pelo menos um conjunto dimensão para "wrap_content" , e quando a view cresce para acomodar o conteúdo, a imagem Nine Patch também estara escalada para corresponder ao tamanho da View . Um exemplo do uso de uma imagem Nine Patch é o fundo usado por padrão do Android da widget botão, que deve esticar para acomodar o texto (ou imagem) dentro do botão.

Mesmo que com um bitmap normal, você pode consultar um arquivo Nine-Patch diretamente ou através de um recurso definido pelo XML.

Nine Patch-File

localização do arquivo:

res/drawable/filename.9.png

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma NinePatchDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

exemplo:

Com uma imagem salva no res/drawable/myninepatch.9.png , este XML disposição aplica o patch do nine para uma view:

```
<Button
```

```
    android:layout_height = "wrap_content"
```

```
    android:layout_width = "wrap_content"
```

```
    android:background = "@ drawable/myninepatch" />
```

Um XML Nine-Patch é um recurso definido em XML que aponta para um arquivo Nine Patch. O XML pode especificar pontilhamento para a imagem.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma NinePatchDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

sintaxe:

```
<? Versão xml = "1.0" encoding = "UTF-8">
```

```
< nove-patch
```

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

```
  android: src = "@ [pacote]: drawable / drawable_resource "
```

```
  android: dither = ["verdadeiro" | "false"] />
```

elementos:

```
<nine-patch>
```

Define a fonte Nine-Patch e suas propriedades.

atributos:

xmlns:android

String. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android".

android: src

Recurso Drawable. Obrigatório. Referência para um arquivo Nine Patch.

android:dither

Boolean . Habilita ou desabilita compatibilidade do bitmap se o bitmap não tem a configuração mesmo pixel como a tela (por exemplo: um bitmap 8888 ARGB com 565 RGB tela).

exemplo:

```

<? Versão xml = "1.0" encoding = "UTF-8">
<Nove-patch xmlns:android =
"http://schemas.android.com/apk/res/android"
    android:src = "@drawable/myninepatch"
    android:dither = "false" />

```

Layer List

A LayerDrawable é um objeto drawable que gerencia uma série de outros drawables. Cada drawable na lista é desenhado no fim da lista drawable.

Cada drawable é representado por um elemento <item> de um único elemento <layer-list>.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma LayerDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

sintaxe:

```

<?xml version="1.0" encoding="utf-8"?>
<layer-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:id="@+[package:]id/resource_name"

```



```
        android:top="dimension"
        android:right="dimension"
        android:bottom="dimension"
        android:left="dimension" />
</layer-list>
```

elementos:

<layer-list>

Obrigatório. Este deve ser o elemento raiz. Contém um ou mais elementos <item>

.

atributos:

xmlns: android

string. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

<item>

Define um drawable colocado na camada drawable, numa posição definida pelos seus atributos. Deve ser associado a um elemento <selector>. Aceita elementos <bitmap>.

atributos:

android: drawable

Recurso Drawable. Obrigatório. Referência a um recurso drawable.

android: id

Identificação de recurso. A identificação do recurso exclusivo para este drawable. Para criar um novo ID para o recurso para este item, use a forma: "@+id/nome". O sinal de mais indica que este deve ser criado como um novo ID. Você pode usar este identificador para recuperar e modificar o drawable com View.findViewById () ou Activity.findViewById () .

android: top

Inteiro. O deslocamento superior em pixels.

android:right

Inteiro. O direito deslocamento em pixels.

android:bottom

Inteiro. O fundo deslocamento em pixels.

Android:left

Inteiro. O deslocamento à esquerda em pixels.

Por padrão todos os itens drawable são escalados para se ajustarem ao tamanho da exibição. Assim, colocando as imagens em uma lista de camadas em diferentes posições pode-se aumentar o tamanho da vista em alguns as imagens conforme o caso. Para evitar itens de escala na lista, use um elemento <bitmap> dentro do elemento <item> para especificar o drawable e definir a gravidade para algo que não escala, como o "centro" . Por exemplo, o seguinte <item> define um item que as escalas cabem na sua View recipiente:

```
<item android:drawable="@drawable/image" />
```

Para evitar a escala, o exemplo a seguir usa um elemento <bitmap> com gravity centrado:

```
<item>
  <Bitmap
    android:src = "@image/drawable"
    android:gravity = "center" />
</item>
```

exemplo:

Arquivo XML salvo em res/drawable/layers.xml :

```

<?xml version="1.0" encoding="utf-8"?>
<layer-list
xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <bitmap android:src="@drawable/android_red"
            android:gravity="center" />
    </item>
    <item android:top="10dp" android:left="10dp">
        <bitmap android:src="@drawable/android_green"
            android:gravity="center" />
    </item>
    <item android:top="20dp" android:left="20dp">
        <bitmap android:src="@drawable/android_blue"
            android:gravity="center" />
    </item>
</layer-list>

```

Observe que este exemplo usa uma lista de elementos <bitmap> para definir o recurso drawable para cada item de gravity "center". Isto garante que nenhuma das imagens sejam dimensionadas para se ajustarem ao tamanho do recipiente, devido ao redimensionamento das imagens provocada por offset.

Esta disposição aplica-se a XML drawable para a View:

```

<ImageView
    android:layout_height = "wrap_content"
    android:layout_width = "wrap_content"
    android:src = "@ drawable/layers" />

```

O resultado é uma pilha de imagens cada offset:



State List

A `StateListDrawable` é um objecto drawable definido em XML que usa várias imagens diferentes para representar o mesmo gráfico, dependendo do estado do objecto. Por exemplo, um botão pode existir em um dos vários estados diferentes (pressionado, focado ou nenhum), usando um drawable lista local, você pode fornecer uma imagem de fundo diferente para cada estado.

Você pode descrever a lista de estados em um arquivo XML. Cada gráfico é representado por um elemento `<item>` dentro de um único elemento `<selector>`. Cada `<item>` usa vários atributos para descrever o estado em que ele deve ser usado como o gráfico do drawable.

Durante cada mudança de estado, a lista de estado é atravessada de cima para baixo e o primeiro item que corresponde ao estado atual é usado, a seleção é não baseado na "melhor posição", mas simplesmente o primeiro item que atenda os critérios mínimos do Estado .

localização do arquivo:

`res/drawable/filename.xml`

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro do recurso para uma `StateListDrawable` .

referência de recurso:

Em Java: `R.drawable arquivo`

em XML: `@ [pacote]: drawable / arquivo`

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android"
    android:constantSize=["true" | "false"]
    android:dither=["true" | "false"]
    android:variablePadding=["true" | "false"] >
    <item
        android:drawable="@[package:]drawable/drawable_resource"
        android:state_pressed=["true" | "false"]
        android:state_focused=["true" | "false"]
        android:state_hovered=["true" | "false"]
        android:state_selected=["true" | "false"]
        android:state_checkable=["true" | "false"]
        android:state_checked=["true" | "false"]
        android:state_enabled=["true" | "false"]
        android:state_activated=["true" | "false"]
        android:state_window_focused=["true" | "false"] />
    </selector>
```

elementos:

<selector>

Obrigatório. Este deve ser o elemento raiz. Contém um ou mais elementos <item>.

atributos:

xmlns: android

String. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

android:constantSize

Boolean . "true" se o tamanho relatou um drawable interno permanece constante como as mudanças de estado (o tamanho é o máximo de

todos os estados); "false" se o tamanho varia de acordo com o estado atual. O padrão é falso.

android:dither

Boolean. "true" para permitir a compatibilidade do bitmap se o bitmap não tem a configuração de pixel igual a tela (por exemplo, um bitmap 8888 ARGB com 565 RGB tela); "false" para desativar pontilhamento. O padrão é verdadeiro.

android:variablePadding

Boolean. "true" se o drawable deve mudar com base no estado atual que está selecionado; "false" se o preenchimento deve permanecer o mesmo (com base no preenchimento máximo de todos os estados). A ativação deste recurso exige que você lide com o acerto de layout quando as mudanças de estado acontecem, o que muitas vezes não é suportado. O padrão é falso.

<item>

Define um drawable para uso durante certos estados, como descrito pelos seus atributos. Deve ser um elemento <selector>.

atributos:

android:drawable

Recurso Drawable. Obrigatório. Referência a um recurso drawable.

android:state_pressed

Boolean. "True" se este item deve ser usado quando o objeto é pressionado (como quando um botão é tocado/clicado); "false" se este item deve ser usado no estado, padrão não pressionado.

android:state_focused

Boolean. "True" se este item deve ser usado quando o objeto tem o foco de entrada (por exemplo, quando o usuário seleciona uma entrada de texto); "false" se este item deve ser usado no estado, padrão não focalizada.

`android:state_hovered`

Boolean. "True" se este item deve ser usado quando o objeto está abaixo de um cursor; "false" se este item deve ser usado no estado padrão. Muitas vezes, isso pode ser usado o mesmo drawable para o estado "focado".

Introduzido em nível API 14.

`android: state_selected`

Boolean. "True" se este item deve ser usado quando o objeto é a seleção atual do usuário ao navegar com um controle direcional (como quando navegando por uma lista com um d-pad), "falso" se este item deve ser usado quando o objeto não está marcado.

O estado selecionado é usado quando o foco (`android:state_focused`) não é suficiente (como quando vista de lista tem foco e um item dentro ele é selecionado com um d-pad).

`android: state_checkable`

Boolean. "True" se este item deve ser usado quando o objeto é verificável, "falso" se este item deve ser usado quando o objeto não é verificável. (Apenas útil se o objeto pode fazer a transição entre um widget verificados e não-verificáveis.)

`android: state_checked`

Boolean. "True" se este item deve ser usado quando o objeto é verificado, "falso" se ela deve ser usada quando o objeto é não-marcado.

`android: state_enabled`

Boolean . "True" se este item deve ser usado quando o objeto é ativado (capaz de receber toque/clique eventos); "false" se ela deve ser usada quando o objeto é desativado.

android:state_activated

Boolean. "True" se este item deve ser usado quando o objeto é ativado como a seleção persistente (tal como para "destacar" o item da lista previamente selecionado em uma vista de navegação persistente); "false" se ele deve ser usado quando o objeto não é ativado.

Introduzido em nível API 11.

android:state_window_focused

Boolean. "True" se este item deve ser usado quando a janela do aplicativo tem o foco (o aplicativo está em primeiro plano), "falso" se este item deve ser usado quando a janela do aplicativo não tem foco (por exemplo, se a sombra é de notificação puxado para baixo ou uma caixa de diálogo).

exemplo:

Arquivo XML salvo em res/drawable/button.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
        android:drawable="@drawable/button_pressed" />    <!--
pressed -->
    <item android:state_focused="true"
        android:drawable="@drawable/button_focused" />    <!--
focused -->
    <item android:state_hovered="true"
        android:drawable="@drawable/button_focused" />    <!--
hovered -->
    <item    android:drawable="@drawable/button_normal" />    <!--
default -->
</selector>
```


Esta disposição aplica-se a XML lista local drawable a um botão:

```
<Button
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:background="@drawable/button" />
```

Level List

Um Drawable que gere um número de Drawables alternados, com um atribuído de valor numérico máximo. Definindo o valor do nível do drawable com `setLevel()` carrega o recurso drawable na lista que tem um nível `android:maxLevel` valor maior ou igual ao valor passado para o método.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma `LevelListDrawable` .

referência de recurso:

Em Java: `R.drawable arquivo`

em XML: `@ [pacote]: drawable / arquivo`

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<level-list
    xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:drawable="@drawable/drawable_resource"
        android:maxLevel="integer"
        android:minLevel="integer" />
```

</level-list>

elementos:

<level-list>

Este deve ser o elemento raiz. Contém um ou mais elementos <item>.

atributos:

xmlns: android

string. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

<item>

Define um drawable para usar em um determinado nível.

atributos:

android:drawable

Recurso Drawable. Obrigatório. A referência a um recurso drawable para ser inserida.

android: maxLevel

Inteiro. O nível máximo permitido para este item.

android: MinLevel

Inteiro. O nível mínimo permitido para este item.

exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<level-list
```

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

```
<item
```

```
    android:drawable="@drawable/status_off"
```

```
    android:maxLevel="0" />
```

```

<item
    android:drawable="@drawable/status_on"
    android:maxLevel="1" />
</level-list>

```

Uma vez que este é aplicado a uma view, o nível pode ser alterado com `setLevel()` ou `setImageLevel()` .

Transition Drawable

Um `TransitionDrawable` é um objeto que pode drawable cross-fade entre os dois recursos drawable.

Cada drawable é representado por um elemento `<item>` dentro de um único elemento `<transition>`. Não mais que dois itens são suportados. Para fazer a transição para a frente, chamar `startTransition()` . Para trás transição, chamar `reverseTransition()` .

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma `TransitionDrawable` .

referência de recurso:

Em Java: `R.drawable.filename`

em XML: `@ [pacote]: drawable/arquivo`

sintaxe:

```

<?xml version="1.0" encoding="utf-8"?>
<transition
    xmlns:android="http://schemas.android.com/apk/res/android" >

```

```
<item
    android:drawable="@[package:]drawable/drawable_resource"
    android:id="@+[package:]id/resource_name"
    android:top="dimension"
    android:right="dimension"
    android:bottom="dimension"
    android:left="dimension" />
</transition>
```

Elementos

</transition>

Obrigatório. Este deve ser o elemento raiz. Contém um ou mais elementos <item>.

atributos:

xmlns: android

String. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

<item>

Define um drawable para usar como parte da transição drawable. Deve ser um elemento <transition>. Aceita elementos <bitmap>.

atributos:

android:drawable

Recurso Drawable. Obrigatório. Referência a um recurso drawable.

android: id

Identificação de recurso. A identificação do recurso exclusivo para este drawable. Para criar um ID novo recurso para este item, use a forma: "@+id/nome". O sinal de mais indica que este deve ser criado como um novo ID. Você pode usar este identificador para recuperar e modificar o drawable com View.findViewById() ou Activity.findViewById().

android:top

Inteiro. O deslocamento superior em pixels.

android:right

Inteiro. O direito deslocamento em pixels.

android:bottom

Inteiro. O fundo deslocamento em pixels.

Android:left

Inteiro. O deslocamento à esquerda em pixels.

exemplo:

Arquivo XML salvo em res/drawable/transition.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<transition
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:drawable="@drawable/on" />
    <item android:drawable="@drawable/off" />
</transition>
```

Esta disposição aplica-se a XML drawable para a View:

```
<ImageButton
    android:id="@+id/button"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:src="@drawable/transition" />
```

E o código a seguir executa uma transição 500ms do primeiro item para o segundo:

```
ImageButton button = (ImageButton) findViewById(R.id.button);
TransitionDrawable drawable = (TransitionDrawable)
button.getDrawable();
drawable.startTransition(500);
```

Inset Drawable

Um drawable definido em XML que insere outro drawable a uma determinada distância. Isto é útil quando a View precisa de um pano de fundo que é menor do que os limites reais da view.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma InsetDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<inset
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:drawable="@drawable/drawable_resource"
  android:insetTop="dimension"
  android:insetRight="dimension"
  android:insetBottom="dimension"
  android:insetLeft="dimension" />
```

elementos:

<inset>

Define o drawable inserido. Este deve ser o elemento raiz.

atributos:

xmlns: android

String. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

android:drawable

Recurso Drawable. Obrigatório. A referência a um recurso drawable para ser inserida.

android:insetTop

Dimensão. Inserido acima, como um valor de dimensão ou recurso dimensão

android:insetRight

Dimensão. inserido a direita, como o valor de uma dimensão ou recurso dimensão

android:insetBottom

Dimensão. Inserido abaixo, como um valor de dimensão ou recurso dimensão

android:insetLeft

Dimensão. inserido à esquerda, como um valor de dimensão ou recurso dimensão

exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<inset xmlns:android="http://schemas.android.com/apk/res/android"
    android:drawable="@drawable/background"
    android:insetTop="10dp"
    android:insetLeft="10dp" />
```

Shape Drawable

Trata-se de uma forma genérica definida em XML.

localização do arquivo:

res/drawable/nome do arquivo.xml

O nome do arquivo é usado como o ID de recurso.

datatype recurso compilado:

Ponteiro recurso a uma GradientDrawable .

referência de recurso:

Em Java: . R.drawable arquivo

em XML: @ [pacote]: drawable / arquivo

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape=["rectangle" | "oval" | "line" | "ring"] >
    <corners
        android:radius="integer"
        android:topLeftRadius="integer"
        android:topRightRadius="integer"
        android:bottomLeftRadius="integer"
```



```

        android:bottomRightRadius="integer" />
<gradient
    android:angle="integer"
    android:centerX="integer"
    android:centerY="integer"
    android:centerColor="integer"
    android:endColor="color"
    android:gradientRadius="integer"
    android:startColor="color"
    android:type=["linear" | "radial" | "sweep"]
    android:useLevel=["true" | "false"] />
<padding
    android:left="integer"
    android:top="integer"
    android:right="integer"
    android:bottom="integer" />
<size
    android:width="integer"
    android:height="integer" />
<solid
    android:color="color" />
<stroke
    android:width="integer"
    android:color="color"
    android:dashWidth="integer"
    android:dashGap="integer" />
</shape>

```

elementos:

<shape>

O drawable forma. Este deve ser o elemento raiz.

atributos:

xmlns: android

String. Obrigatório. Define o namespace XML, que deve ser "http://schemas.android.com/apk/res/android" .

android:shape

Palavra-chave. Define o tipo de forma. Os valores válidos são:

Valor	Description
"rectangle"	Um retângulo que enche a view que contém. Esta é a forma padrão.
"oval"	Uma forma oval que se encaixa as dimensões da exibição que contém.
"line"	Uma linha horizontal, que abrange a largura da View. Isto requer que a forma <stroke> para definir a largura da linha.
"ring"	Uma forma de anel.

Os seguintes atributos são utilizados apenas quando android:shape for "ring" :

android:innerRadius

Dimensão. O raio da parte interior do anel (o buraco no meio), como um valor da dimensão ou dimensão dos recursos .

android:innerRadiusRatio

Numerico. O raio da parte interior do anel, expressa como uma razão entre o raio e a largura do anel. Por exemplo, se android:innerRadiusRatio = "5" , então o raio interno é igual a largura do anel dividido por 5. Este valor é substituído por android:innerRadius . O valor padrão é 9.

android:thickness

Dimensão . A espessura do anel, tal como um valor de dimensão ou de dimensão de recursos .

android:thicknessRatio

Numerico. A espessura do anel, expressa como uma razão entre a largura do anel. Por exemplo, se android: thicknessRatio = "2" , então a espessura é igual a largura do anel dividido por 2. Este valor é substituído por android:innerRadius . O valor padrão é 3.

android:useLevel

Boolean. "True" se este é usado como um LevelListDrawable . Este deve ser normalmente "falso" ou sua forma pode não aparecer.

<corners>

Cria cantos arredondados para a forma. Aplica-se somente quando a forma é um retângulo.

atributos:

android:radius

Dimensão. O raio para todos os cantos, como um valor de dimensão ou recurso dimensão . Este é substituído para cada canto pelos seguintes atributos.

android:topLeftRadius

Dimensão. O raio para o canto superior esquerdo, como um valor de dimensão ou recurso dimensão.

android:topRightRadius

Dimensão. O raio para o canto superior direito, como um valor de dimensão ou recurso dimensão.

android:bottomLeftRadius

Dimensão. O raio para o canto inferior esquerdo, como um valor de dimensão ou recurso dimensão .

android:bottomRightRadius

Dimensão. O raio para o canto inferior direito, como um valor de dimensão ou recurso dimensão.

<gradient>

Especifica um gradiente de cor para a forma.

atributos:

android:angle

Inteiro. O ângulo de inclinação, em graus. 0 é esquerda para a direita, 90 é baixo para cima. Ele deve ser um múltiplo de 45. O padrão é 0.

android:centerX

Numerico. A relação da posição x para o centro do gradiente (0 - 1,0).

android:centerY

Numerico . O relação da posição Y para o centro do gradiente (0 - 1,0).

android: CenterColor

Cor. Cor opcional que vem entre o início e as cores finais, como um valor hexadecimal ou recurso de cor .

android:EndColor

Cor . A cor final, como um valor hexadecimal ou recurso de cor .

android:gradientRadius

Numerico. O raio para o gradiente. Aplicada somente quando android: type = "radial" .

android:StartColor

Cor . A cor inicial, como um valor hexadecimal ou recurso de cor .

android:type

Palavra-chave . O tipo de padrão de gradiente de aplicar. Os valores válidos são:

Valor	Descrição
"linear"	Um gradiente linear. Este é o padrão.
"radial"	Um gradiente radial. A cor é a cor início centro.
"sweep"	Um gradiente linha de varredura.

android:useLevel

Boolean. "True" se este é usado como um LevelListDrawable .

<padding>

Preenchimento para aplicar ao elemento de exibição

atributos:

Android:left

Dimensão. Padding à esquerda, como um valor de dimensão ou recurso dimensão.

android: top

Dimensão. Padding-top, como um valor de dimensão ou recurso dimensão.

android:right

Dimensão. Padding-right, como um valor de dimensão ou recurso dimensão.

android:botton

Dimensão. Padding-bottom, como um valor de dimensão ou recurso dimensão .

<size>

O tamanho da forma.

atributos:

android:height

Dimensão. A altura da forma, tal como um valor de dimensão ou de dimensão de recursos .

android:width

Dimensão. A largura da forma, tal como um valor de dimensão ou de dimensão de recursos.

<solid>

Uma cor sólida para preencher a forma.

atributos:

android:color

Cor. A cor a ser aplicada com a forma, como um valor hexadecimal ou recurso de cor.

<stroke>

Uma linha de curso para a forma.

atributos:

android:width

Dimensão. A espessura da linha, tal como um valor de dimensão ou de dimensão de recursos .

android:color

Cor . A cor da linha, como um valor hexadecimal ou recurso de cor .

android: dashGap

Dimensão. A distância entre a linha e traços, como um valor da dimensão ou dimensão dos recursos . Válido somente se android:dashWidth está definido.

android: dashWidth

Dimensão . O tamanho de cada linha tracejada, como um valor da dimensão ou dimensão dos recursos . Válido somente se android: dashGap está definido.

exemplo:

Arquivo XML salvo em res / drawable / gradient_box.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <gradient
        android:startColor="#FFFF0000"
        android:endColor="#80FF00FF"
        android:angle="45"/>
    <padding android:left="7dp"
        android:top="7dp"
        android:right="7dp"
        android:bottom="7dp" />
    <corners android:radius="8dp" />
</shape>
```

Este layout aplica um XML a forma drawable para a View:

```
<TextView
    android:background="@drawable/gradient_box"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content" />
```

Este código do aplicativo recebe o drawable forma e aplica-o a uma view:

```
Resources res = getResources();
```

```
Drawable shape = res. getDrawable(R.drawable.gradient_box);
```

```
TextView tv = (TextView)findViewById(R.id.textview);  
tv.setBackground(shape);
```

Recursos de strings

Um recurso de string fornece os textos para a sua aplicação, podendo opcionalmente aplicar estilos e formatações ao texto. Existem três tipos de recursos que podem fornecer strings a sua aplicação:

String - XML recurso que fornece uma única sequência caracteres.

String Array - XML recurso que fornece um conjunto de strings.

Qualitiy strings (plurais) - recurso XML que carrega strings diferentes para diferentes idiomas de uma mesma palavra ou frase.

Todas os recursos associadas a strings são capazes de aplicar alguma marcação de estilo e argumentos de formatação.

String

Uma única cadeia que pode ser referenciada a partir do aplicativo ou a partir de outros arquivos de recursos (como um layout XML).

localização do arquivo:

res/valores/filename.xml

o nome do arquivo é arbitrário. O elemento <string> nome será usado como a identificação de recurso.

datatype recurso compilado:

Ponteiro recurso a uma corda .

referência de recurso:

Em Java: . R.string string_name

em XML: @ string / string_name

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string
        name="string_name"
        >text_string</string>
</resources>
```

elementos:

<resources>

Obrigatório. Este deve ser o nó raiz.

Sem atributos.

<string>

Uma cadeia de caracteres, que pode incluir tags de estilo. Cuidado
você deve evitar apóstrofes e aspas.

atributos:

name

string. Um nome para a cadeia. Este nome será usado como a
identificação de recurso.

exemplo:

Arquivo XML salvo em res/valores/strings.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
<string name="hello">Hello!</string>
</resources>
```

Este layout aplica uma seqüência no XML para a View:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
```

Este código do aplicativo recupera uma string:

```
String string = getString(R.string.hello);
```

Você pode usar o `getString(int)` ou `getText(int)` para recuperar uma string. `getText(int)` irá reter qualquer estilo de texto aplicada à string.

String Array

Uma matriz de strings que podem ser referenciados a partir da aplicação.

Nota: A matriz de string é um recurso simples que é referenciado usando o valor fornecido no nome de atributo (não o nome do arquivo XML). Como tal, você pode combinar recursos matriz de string com outros recursos simples no arquivo XML sob um elemento `<resources>`.

localização do arquivo:

res/valores/filename.xml

o nome do arquivo é arbitrária. O `<string-array>` elemento nome será usado como a identificação de recurso.

datatype recurso compilado:

Ponteiro de recursos para uma matriz de strings.

referência de recurso:

Em Java: . R.array string_array_name

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array
    name="string_array_name">
    <item
      >text_string</item>
    </string-array>
  </resources>
```

elementos:

<resources>

Obrigatório. Este deve ser o nó raiz.

Sem atributos.

<string-array>

Define uma matriz de strings. Contém um ou mais elementos

<item>.

atributos:

name

string. Um nome para a matriz. Este nome será usado como a identificação de recurso para fazer referência à matriz.

<item>

Uma cadeia de caracteres, que pode incluir tags de estilo. O valor pode ser uma referência a outro recurso de string. Deve ser um elemento <string-array>. Cuidado que você deve evitar apóstrofes e aspas.

Sem atributos.

exemplo:

Arquivo XML salvo em res/valores/strings.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
    </string-array>
</resources>
```

Este código aplicativo recupera uma matriz de cadeia:

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Quantity Strings (Plurals)

Línguas diferentes têm regras diferentes para acordo gramatical quanto a quantidades. Em Inglês, por exemplo, a quantidade de 1 é um caso especial. Nós escrevemos "um livro", mas para qualquer outra quantidade escrevemos "n livros". Esta distinção entre o singular e o plural é muito comum, mas outras linguagens fazem distinções mais finas. O conjunto completo é suportado pelo Android é zero, um, dois, poucos, muitos e outros .

As regras para decidir qual caso usar para um determinado idioma e quantidade pode ser muito complexa, por isso Android fornece métodos como `getQuantityString()` para selecionar o recurso adequado para você.

Embora historicamente chamado "string de quantidades", string de quantidade deve apenas ser utilizado para plurais. Seria um erro usar strings

de quantidade para implementar algo como Gmail "Caixa de entrada" versus "Caixa de Entrada (12)" quando há mensagens não lidas, por exemplo. Pode parecer conveniente usar string de quantidade em vez de um caso de declaração, mas é importante notar que alguns idiomas (como o chinês) não fazem essas distinções gramaticais em tudo, assim você sempre terá a outra string.

A seleção de qual string utilizar é feita exclusivamente com base na necessidade gramatical. Em Inglês, uma string para a zero será ignorado, mesmo que a quantidade seja 0, porque 0 não é gramaticalmente diferente de 2, ou qualquer outro número, exceto 1 ("zero livros", "um livro", "dois livros", e assim por diante).

Não se deixe enganar, quer pelo fato de que, digamos, dois soa como ele só poderia aplicar à quantidade 2: uma linguagem pode exigir que 2, 12, 102 (e assim por diante) são todas tratadas como um outro, mas de forma diferente para outras quantidades. Confie no seu tradutor para saber o que realmente distinções sua língua insiste.

Muitas vezes é possível evitar strings de quantidade utilizando formulações de quantidades neutras como "Livros: 1". Isso fará com que a sua vida e a vida de seus tradutores mais fácil, se é um estilo que está em consonância com a sua aplicação.

Nota: Uma coleção de plurais é um recurso simples que é referenciado usando o valor fornecido no nome do atributo (não o nome do arquivo XML). Como tal, você pode combinar recursos plurais com outros recursos simples no arquivo XML, sob um elemento <resources>.

localização do arquivo:

res/valores/filename.xml

o nome do arquivo é arbitrária. O elemento <plurals> nome será usado como a identificação de recurso.

referência de recurso:

Em Java: . R.plurals.plural_name

sintaxe:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <plurals
    name="plural_name">
    <item
      quantity=["zero" | "one" | "two" | "few" | "many" | "other"]
      >text_string</item>
    </plurals>
  </resources>
```

elementos:

<resources>

Obrigatório. Este deve ser o nó raiz.

Sem atributos.

<plurals>

Uma coleção de strings, de que, uma string é fornecida, dependendo da quantidade de algo. Contém um ou mais elementos <item>.

atributos:

nome

String. Um nome para o par de strings. Este nome será usado como a identificação de recurso.

<item>

Uma string plural ou singular. O valor pode ser uma referência a outro recurso de string. Deve ser um elemento <plurals>. Cuidado deve-se evitar apóstrofos e aspas.

atributos:

quantity

Palavra-chave. Um valor que indica quando esta string deve ser usada. Os valores válidos, com exemplos entre parênteses:

Valor	Descrição
zero	Quando o idioma requer tratamento especial do número 0 (como em árabe).
one	Quando o idioma requer tratamento especial de números como um (como acontece com o número 1 em línguas inglesa e mais outros, em russo, qualquer número que termina em 1, mas não termina em 11 é nesta classe).
two	Quando o idioma requer tratamento especial de números como dois (como 2 em galês, ou 102 em esloveno).
few	Quando a língua requer tratamento especial de "poucos" (como números com 2, 3, e 4, em Checa, ou números terminando 2, 3, ou 4, mas não de 12, 13, ou 14, em polaco).
many	Quando o idioma requer tratamento especial de "muitos" os números (como números que terminam com 11-99 em maltês).
other	Quando a linguagem não requer tratamento especial da quantidade determinada (como acontece com todos os números em chinês, ou 42 em Inglês).

exemplo:

Arquivo XML salvo em res/valores/strings.xml :

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">One song found.</item>
        <item quantity="other">%d songs found.</item>
    </plurals>
</resources>

```

Arquivo XML salvo em res/valores-pl/strings.xml :

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <plurals name="numberOfSongsAvailable">
        <item quantity="one">Znaleziono jedną piosenkę.</item>
        <item quantity="few">Znaleziono %d piosenki.</item>
        <item quantity="other">Znaleziono %d piosenek.</item>
    </plurals>
</resources>

```

Código Java:

```

int count = getNumberOfSongsAvailable();
Resources res = getResources();
String songsFound =
res.getQuantityString(R.plurals.numberOfSongsAvailable, count, count);

```

Ao usar o método `getQuantityString()`, você precisa passar a contagem duas vezes se a sua sequência inclui formatação de strings com um número. Por exemplo, para a sequência `%d` músicas encontradas, a primeira contagem de parâmetro seleciona a string plural adequada e a segunda contagem parâmetro é inserido no espaço reservado `%d`. Se suas strings de plural não incluem formatação de strings, você não precisa passar o terceiro parâmetro para `getQuantityString`.

Formatação e Styling

Aqui estão algumas coisas importantes que você deve saber sobre a maneira correta de formato e estilo seus recursos string.

Apóstrofos e aspas

Se você tem um apóstrofo ou uma aspa em sua string, você deve colocar na string toda na citações. Por exemplo, aqui estão alguns exemplos que fazem isso:

```
< <string name="good_example">"This'll work"</string>
<string name="good_example_2">This'll also work</string>
<string name="bad_example">This doesn't work</string>
<string name="bad_example_2">XML encodings don't
work</string>
```

Formatação de Strings

Se você precisar formatar suas strings usando String.format (String, Object ...), então você pode fazê-lo, colocando seus argumentos de formato no recurso da string. Por exemplo, com o recurso a seguir:

```
<string name="welcome_messages">Hello, %1$s! You have %2$d
new messages.</string>
```

Neste exemplo, a cadeia de formato tem dois argumentos: %1\$s é uma string e %2\$d é um número decimal. Você pode formatar o texto com argumentos de sua aplicação como esta:

```
Resources res = getResources();
String text =
String.format(res.getString(R.string.welcome_messages), username,
mailCount);
```

Styling com marcação HTML

Você pode adicionar um estilo de suas strings com marcação HTML. Por exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="welcome">Welcome to <b>Android</b>!</string>
</resources>
```

Elementos suportados HTML incluem:

- para negrito texto.
- <i> para itálico texto.
- <u> para sublinhar o texto.

Às vezes, você pode querer criar um recurso de texto estilo que também é usado como uma string formatada. Normalmente, isso não vai funcionar porque o método `String.format (String, Object ...)` irá retirar todas as informações de estilo da string. O trabalho em torno disso é escrever as tags HTML com entidades, e que são, então, recuperados com `fromHtml(String)`, após ocorre a formatação. Por exemplo:

Guarde o seu recurso de texto estilo como uma string HTML:

```
<resources>
    <string name="welcome_messages">Hello, %1$s! You have
    &lt;b>%2$d new messages&lt;/b>.</string>
</resources>
```

Nesta string formatada, um elemento é adicionado. Observe que o suporte de abertura é HTML, usando a notação <.

Em seguida, chamar `fromHtml (String)` para converter o texto HTML em texto com estilo:

```
Resources res = getResources();
String text =
String.format(res.getString(R.string.welcome_messages), username,
mailCount);
CharSequence styledText = Html.fromHtml(text);
```

Porque o método `fromHtml(String)` irá formatar todas as entidades HTML, certifique-se para quaisquer caracteres possíveis HTML nas strings você usa com o texto formatado, usando `HtmlEncode(String)` . Por exemplo, se você estará passando uma string como argumento para `String.format()` que pode conter caracteres como "<" ou "&", então eles devem ser informados antes da formatação, para que quando a sequência formatada for passada através `fromHtml(String)` , as palavras saiam da maneira que foram escritos originalmente. Por exemplo:

```
String escapedUsername = TextUtil.htmlEncode(username);
```

```
Resources res = getResources();
String text = String.format(res.getString(R.string.welcome_messages),
escapedUsername, mailCount);
CharSequence styledText = Html.fromHtml(text);
```

Componentes do Android: Processos e Threads, Ciclo de Vida

Processos

Quando um componente de uma aplicação se inicia e a aplicação não tem quaisquer outros componentes em execução, o sistema Android inicia um novo processo Linux para o aplicativo com um único segmento de execução. Por padrão, todos os componentes do mesmo aplicativo são executados no mesmo processo e thread (chamada de thread "main"). Se um componente de aplicação começa e já existe um processo para essa aplicação (porque outro componente da aplicação o iniciou), em seguida, o componente é iniciado dentro desse processo e usa a mesma linha de execução. No entanto, você pode mandar diferentes componentes em seu aplicativo para executar em processos separados, e você pode criar tópicos adicionais para qualquer processo.

Processos

Por padrão, todos os componentes do mesmo aplicativo são executados no mesmo processo e a maioria das aplicações não deve mudar isso. No entanto, se você achar que precisa para controlar quais processo de um determinado componente pertence, você pode fazê-lo no arquivo de manifesto.

Uma entrada no manifesto para cada tipo de componente <activity>, <service>, <receiver> e <provider> suportam um atributo `android:process` que pode especificar um processo em que esse componente deve ser executado. Você pode definir esse atributo para que cada componente é executado em seu próprio processo, ou que alguns componentes partes de um processo, enquanto outros não. Você também pode configurar `android:process` para que os componentes de diferentes aplicações sejam executadas no mesmo processo, desde que as aplicações possam compartilhar o mesmo ID de usuário do Linux e são assinados com os mesmos certificados.

O elemento `<application>` também suporta um atributo `android:process`, para definir um valor padrão que se aplica a todos os componentes.

Android pode decidir encerrar um processo em algum momento, quando houver pouca memória e exigido por outros processos que estão mais imediatamente próximo do usuário. Os componentes do aplicativo em execução no processo que encerrou são consequentemente destruídos. Um processo é iniciado novamente para os componentes quando há novamente trabalho para que eles façam.

Ao decidir quais processos encerrar, o sistema Android pesa sua importância relativa para o usuário. Por exemplo, mais facilmente desliga um processo de hospedagem atividades que não são mais visíveis na tela, em comparação a um processo de hospedagem de atividades visíveis. A decisão de terminar um processo, por conseguinte, depende do estado de funcionamento dos componentes deste processo. As regras usadas para decidir que processo para terminar são discutidas abaixo.

Processo de ciclo de vida

O sistema Android tenta manter um processo de aplicação por tanto tempo quanto possível, mas, eventualmente, tem de remover os processos antigos para recuperar a memória para os processos novos, ou mais importantes. Para determinar quais os processos deve manter e qual encerrar, o sistema coloca cada processo em uma "hierarquia importância" com base nos componentes de funcionamento do processo e no estado dos componentes. Processos com o menor importância são eliminadas em primeiro lugar, em seguida, aqueles com a menor importância seguinte, e assim por diante, para recuperar os recursos do sistema.

Existem cinco níveis na hierarquia de importância. A lista que se segue apresenta os diferentes tipos de processos, em ordem de importância (o primeiro processo é o mais importante e é morto passada):

Processo em primeiro plano

Um processo que é necessário para que o usuário que está atualmente executando. Um processo é considerado como estando em primeiro plano, em qualquer uma das condições que se seguem:

- Abriga uma atividade que o usuário está interagindo (método `onResume()` da atividade foi chamado).
- Abriga um serviço que é vinculado à atividade que o usuário está interagindo.
- Abriga um serviço que está sendo executado "em primeiro plano", o serviço chamou `startForeground()`.
- Abriga um serviço que está executando um ciclo de vida de seus callbacks (`onCreate()`, `onStart()`, ou `onDestroy()`).
- Abriga uma `BroadcastReceiver` que está executando o seu método `onReceive()`.

Em geral, apenas alguns poucos processos podem existir em primeiro plano num dado momento. Eles são encerrados apenas como um último recurso, se a memória não está tão baixa que eles não podem continuar executando. Geralmente, nesse ponto, o dispositivo atingiu um estado de paginação de memória, assim é necessário encerrar alguns processos de primeiro plano para manter a interface do usuário responsiva.

Processo visível

Um processo que não tem quaisquer componentes do primeiro plano, mas ainda assim podem afetar o que o usuário vê na tela. Um processo é considerado visível em uma das seguintes condições:

- Abriga uma atividade que não está em primeiro plano, mas ainda é visível para o usuário (seu método `onPause()` foi chamado). Isto pode ocorrer, por exemplo, quando a actividade do primeiro plano inicia um diálogo, o que permite que a atividade anterior possa ser vista por trás.
- Abriga um serviço que é vinculado a uma atividade (ou plano) visível.

Um processo visível é considerado extremamente importante e não vai ser encerrado a menos que isso seja necessário para manter todos os processos de primeiro plano em execução.

Processo de serviço

Um processo que está executando um serviço e foi iniciado com o método `StartService()` e não está em nenhuma das duas categorias mais elevadas. Embora os processos de serviço não estejam diretamente ligados a qualquer coisa que o usuário vê, eles estão geralmente fazendo coisas que o usuário se preocupa (como a reprodução de música em segundo plano ou download de dados na rede), de modo que o sistema os mantém funcionando, a menos que não tenha memória suficiente para fazê-lo, juntamente com todos os novos processos e processos visíveis.

Processo de fundo

Um processo que tem uma atividade que não é visível no momento para o usuário (método `onStop()` a atividade tenham sido chamado). Esses processos não têm impacto direto sobre a experiência do usuário, e o sistema pode encerra-los a qualquer momento para recuperar a memória de primeiro plano, visível, ou processo de serviço. Normalmente existem muitos processos em segundo plano em execução, para que eles sejam mantidos em uma lista LRU (menos utilizado recentemente) para garantir que o processo com a atividade que mais recentemente foi visto pelo usuário é o último a ser encerrado. Se uma atividade implementa seus métodos de ciclo de vida corretamente, e salva seu estado atual, encerrar o seu processo não terá um

efeito visível sobre a experiência do usuário, pois quando o usuário navega de volta para a atividade, a atividade restaura todo o seu estado visível.

Processo vazio

Um processo é o que não detém quaisquer componentes de aplicativos ativos. A única razão para manter esse tipo de processo é vivo para fins de cache, para melhorar o tempo de inicialização da próxima vez que um componente precisa ser executado. O sistema de encerra frequentemente desses processos, a fim de equilibrar os recursos gerais do sistema entre os caches de processos e os caches do kernel subjacentes.

Android ocupa um processo no nível mais alto que pode, com base na importância dos componentes ativos no momento durante o processo. Por exemplo, se um processo hospeda um serviço e uma atividade visível, o processo é classificada como um processo não visível, um processo de serviço.

Além disso, o ranking de processos pode ser aumentado, porque os outros processos são dependentes dela, um processo que está servindo um outro processo que nunca pode ser classificado menor do que o processo que está servindo. Por exemplo, se um provedor de conteúdos no processo A está servindo um cliente no processo B, ou se um serviço no processo A é ligado a um componente de processo B, o processo A é sempre considerado pelo menos tão importante quanto o processo B.

Porque um processo de execução de um serviço é mais alto do que um processo com atividades de fundo, uma atividade que inicia uma operação de longa duração pode iniciar um serviço para essa operação, em vez de simplesmente criar um segmento de trabalho, particularmente se a operação, provavelmente, durar a atividade. Por exemplo, uma atividade que está fazendo upload de uma foto em um site deve iniciar um serviço para realizar o upload para que o upload possa continuar em segundo plano, mesmo se o usuário deixar a atividade. Usando um serviço garante que a operação terá

pelo menos um "processo de serviço" prioritário, independentemente do que acontece com a atividade. Este é o mesmo motivo que os receptores de broadcast devem utilizar serviços, em vez de simplesmente colocar operações demoradas em uma discussão.

Threads

Quando um aplicativo é iniciado, o sistema cria um segmento de execução para a aplicação, chamado de "main". Isto é muito importante, pois é responsável por enviar eventos para os widgets da interface de usuário apropriados, incluindo os eventos. Também é o segmento em que seu aplicativo interage com os componentes da interface do usuário com as ferramentas Android kit (componentes do `android.widget` e pacotes `android.view`). Como tal, a thread principal também é chamada às vezes de thread da interface do usuário.

O sistema não cria uma thread separada para cada instância de um componente. Todos os componentes que são executados no mesmo processo são instanciados na thread de interface do usuário, e chamados do sistema para cada componente que são enviados dessa thread. Consequentemente, os métodos que respondam às chamadas de retorno do sistema (como `onKeyDown()` demonstram as ações do usuário ou um método de retorno de chamada do ciclo de vida) sempre executados na thread interface do usuário do processo.

Por exemplo, quando o usuário toca um botão na tela, envia para seu aplicativo da interface do usuário um evento de toque no widget botão, que por sua vez define seu estado pressionado e mensagens de um pedido de verificação na fila de eventos. A thread do pedido notifica o widget que ele deve se redesenhar.

Quando seu aplicativo realiza um trabalho intensivo em resposta à interação do usuário, este modelo único thread pode comprometer o desempenho, a menos que você implemente o aplicativo corretamente.

Especificamente, se tudo está acontecendo no segmento de interface do usuário, a realização de operações longas, como o acesso à rede ou de consultas de dados irá bloquear toda a UI. Quando a linha é bloqueada, os eventos podem ser executados, incluindo eventos de desenho. Da perspectiva do usuário, o aplicativo parece travar. Pior ainda, se a thread está bloqueada por mais de alguns segundos (cerca de 5 segundos atualmente), é apresentado ao usuário a mensagem "aplicação não está respondendo". O usuário pode, então, decidir encerrar o seu pedido e desinstalá-lo.

Além disso, a UI Android toolkit não é thread-safe. Então, você não deve manipular a interface do usuário a partir de uma thread de trabalho, você tem que fazer toda a manipulação de sua interface de usuário da thread. Assim, há apenas duas regras para o modelo do Android única thread:

- Não bloqueie a thread
- Não acessar o Android UI toolkit de fora da thread

Threads de trabalho

Por causa do modelo único de thread descrito acima, é vital para a capacidade de resposta da interface do usuário do aplicativo que você não bloqueie a thread. Se você tem que realizar operações que não são instantâneos, você deve certificar-se de fazê-las em tópicos separados ("fundo" ou "trabalhado" threads).

Por exemplo, a seguir um código para um processo que faz o download ao clicar em uma imagem de uma thread separada e exibe em um ImageView:

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");  
            mImageView.setImageBitmap(b);  
        }  
    }).start();  
}
```

```

    }
    }).start();
}

```

No início, isso parece funcionar bem, porque cria uma nova thread para lidar com a operação da rede. No entanto, ele viola a regra segundo o modelo single-threaded: não acessar o Android UI toolkit de fora da UI thread esta amostra modifica o ImageView da thread de trabalho. Isso pode resultar em comportamento indefinido e inesperado, o que pode ser difícil e demorado para rastrear.

Para corrigir esse problema, o Android oferece várias maneiras de acessar uma thread. Aqui está uma lista de métodos que podem ajudar:

- Activity.runOnUiThread (Runnable)
- View.post (Runnable)
- View.postDelayed (Runnable, long)

Por exemplo, você pode corrigir o código acima, usando o método View.post (Runnable):

```

public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap =
loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}

```

Agora esta implementação é thread-safe: o funcionamento da rede é feito a partir de uma thread separada, enquanto o ImageView é manipulado a partir da thread.

No entanto, como a complexidade da operação aumenta, este tipo de código pode ser complicado e difícil de manter. Para lidar com interações mais complexas com uma thread de trabalho, você pode considerar o uso de um manipulador em sua thread de trabalho, para processar mensagens entregues a partir de uma thread. Talvez a melhor solução, porém, é estender a classe AsyncTask, o que simplifica a execução de tarefas com threads de trabalho que precisam interagir com a interface do usuário.

Usando AsyncTask

AsyncTask permite executar trabalho assíncrono em sua interface de usuário. Ele executa as operações de bloqueio em uma thread de trabalho e depois publica os resultados no segmento de interface do usuário, sem a necessidade de lidar com tópicos e/ou manipuladores de si mesmo.

Para usá-lo, você deve implementar a subclasse AsyncTask com o método de retorno `doInBackground()`, que é executado em um pool de threads de fundo. Para atualizar a interface do usuário, você deve implementar `onPostExecute()`, que fornece o resultado de `doInBackground()` e é executado na thread de interface do usuário, assim você pode seguramente atualizar a interface do usuário. Você pode, então, executar a tarefa chamando `execute()` da thread.

Por exemplo, você pode implementar o exemplo anterior usando AsyncTask desta forma:

```
public void onClick(View v) {  
    new DownloadImageTask().execute("http://example.com/image.png");  
}
```

```

private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    /** The system calls this to perform work in a worker thread and
        * delivers it the parameters given to AsyncTask.execute() */
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }

    /** The system calls this to perform work in the UI thread and delivers
        * the result from doInBackground() */
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}

```

Agora a interface do usuário é segura e o código é mais simples, porque separa o trabalho, parte que deve ser feita em uma thread de trabalho e a parte que deve ser feita na thread.

Você deve ler a referência `AsyncTask` para um entendimento completo de como usar essa classe, mas aqui está uma visão geral de como ela funciona:

- Você pode especificar o tipo dos parâmetros, os valores de progresso, e o valor final da tarefa
- O método `doInBackground()` executa automaticamente em uma thread de trabalho
- `onPreExecute()`, `onPostExecute()` e `onProgressUpdate()` são todos chamados na thread interface do usuário
- O valor retornado por `doInBackground()` é enviado para `onPostExecute()`

- Você pode chamar `publishProgress()` a qualquer momento em `doInBackground()` para executar `onProgressUpdate ()` sobre a thread
- Você pode cancelar a tarefa a qualquer tempo, de qualquer thread

Cuidado: Outro problema que você pode encontrar ao usar uma thread de trabalho é reinicializações inesperadas em sua atividade devido a uma alteração de configuração de tempo de execução (como quando o usuário muda a orientação da tela), o que pode destruir sua thread de trabalho.

Metodos Thread-safe

Em algumas situações, os métodos implementados podem ser chamados a partir de mais de uma thread e, portanto, devem ser escritos para ser thread-safe.

Isto é especialmente verdade para os métodos que podem ser chamados remotamente, como métodos em um serviço. Quando uma chamada em um método implementado em um `IBinder` origina no mesmo processo em que o `IBinder` está em execução, o método é executado na thread do chamador. No entanto, quando a chamada tem origem num outro processo, o método é executado em uma thread escolhida de um conjunto de threads que o sistema mantém no mesmo processo que o `IBinder` (não é executado no segmento UI do processo). Por exemplo, enquanto método de um serviço `onBind()` seria chamado de uma thread de processo do serviço, os métodos implementados no objeto que `onBind()` retorna (por exemplo, uma subclasse que implementa métodos RPC) seria chamado de threads em um pool. Uma vez que um serviço pode ter mais do que um cliente, mais do que uma thread de pool pode envolver o mesmo método `IBinder` ao mesmo tempo. Métodos `IBinder` devem, portanto, ser implementado para ser thread-safe.

Da mesma forma, um provedor de conteúdo pode receber solicitações de dados que se originam em outros processos. Embora o `ContentResolver` e classes `ContentProvider` escondam os detalhes da comunicação, métodos `ContentProvider` que respondem aos pedidos de consulta com métodos `insert()`, `delete()`, `update()` e `getType()` são chamados de um pool de threads em processo do provedor de conteúdo, e não da thread para o processo. Como esses métodos podem ser chamados a partir de qualquer número de threads ao mesmo tempo, eles também devem ser implementadas para ser thread-safe.

Ciclo de Vida da Atividade

Gerenciando o Ciclo de Vida Atividade

Gestão do ciclo de vida de suas atividades através da implementação de métodos de retorno é crucial para o desenvolvimento de uma aplicação forte e flexível. O ciclo de vida de uma atividade é diretamente afetada por sua associação com outras atividades, a sua tarefa e volta pilha.

Uma atividade pode existir essencialmente em três estados:

Retomada

A atividade está em primeiro plano da tela e tem o foco do usuário. (Este estado é também por vezes referido como "execução".)

Pausada

Outra atividade é em primeiro plano e tem o foco, mas este é ainda visível. Isto é, outra atividade é visível em cima de uma presente e a atividade é parcialmente transparente ou não cobre toda a tela. A atividade fez uma pausa e esta completamente viva (o objeto de Atividade é retida na memória, ele mantém todas as informações do Estado e membro, e permanece ligado ao

gerenciador de janelas), mas pode ser encerrado pelo sistema em situações de memória extremamente baixos.

Parada

A atividade é completamente obscurecida por outra atividade (a atividade está agora no "fundo"). A atividade também é parada (o objeto de Atividade é retido na memória, ele mantém todas as informações do Estado e membro, mas não está ligado ao gerenciador de janelas). Contudo, já não é visível para o utilizador e pode ser encerrada pelo sistema quando a memória é necessário em outros lugares.

Se uma atividade está pausada ou interrompida, o sistema pode remove-la da memória, pedindo para terminar (chamando seu método finish()), ou simplesmente encerrando o seu processo. Quando a atividade é aberta novamente (depois de ter sido terminado ou encerrada), ele deve ser recriado.

Implementando retornos do ciclo de vida

Quando uma atividade transita para dentro e fora dos diferentes estados descrito acima, ela é notificada através de vários métodos de retorno. Todos os métodos de retorno são ganchos que você pode substituir para fazer o trabalho adequado quando ocorrem mudanças de estado na sua atividade. O esqueleto da atividade a seguir inclui cada um dos métodos de ciclo de vida fundamentais:

```
ExampleActivity public class Atividade {  
    @ Override  
    public void onCreate (Bundle savedInstanceState) {  
        super.onCreate (savedInstanceState);  
        // A atividade está sendo criado.  
    }  
    @ Override  
    onStart protected void () {
```



```

        super.onStart ();
        // A atividade está prestes a tornar-se visível.
    }
    @ Override
    onResume protected void () {
        super.onResume ();
        // A atividade tornou-se visível (é agora "retomada").
    }
    @ Override
    onPause protected void () {
        super.onPause ();
        // Outra atividade está sendo foco (esta atividade está prestes a ser
"pausa").
    }
    @ Override
    onStop protected void () {
        super.onStop ();
        // A atividade não é mais visível (agora está "parado")
    }
    @ Override
    onDestroy protected void () {
        super.onDestroy ();
        // A atividade está prestes a ser destruída.
    }
}

```

Nota: A implementação destes métodos do ciclo de vida devem sempre chamar a implementação de superclasse antes de fazer qualquer trabalho, como mostram os exemplos acima.

Em conjunto, estes métodos definem o ciclo de vida completo de uma atividade. Ao implementar esses métodos, você pode monitorar três loops aninhados no ciclo de vida de atividade:

A vida inteira de uma atividade acontece entre a chamada para `onCreate()` e a chamada para `OnDestroy()`. Sua atividade deve realizar a configuração de estado "global" (como a definição de layout) no `onCreate()`, e liberar todos os recursos restantes no `onDestroy()`. Por exemplo, se sua atividade tem um segmento em execução em segundo plano para fazer download de dados a partir da rede, pode criar esse segmento no `onCreate()` e, em seguida, parar o segmento em `onDestroy()`.

O tempo de vida visível de uma atividade acontece entre a chamada de `OnStart()` e a chamada de `onStop()`. Durante este tempo, o usuário pode ver a atividade na tela e interagir com ela. Por exemplo, `onStop()` é chamado quando uma nova atividade começa e este não é mais visível. Entre estes dois métodos, você pode manter os recursos que são necessários para mostrar a atividade para o usuário. Por exemplo, você pode registrar um `BroadcastReceiver` em `onStart()` para monitorar as mudanças que impactam sua interface, e cancelar o registro em `onStop()` quando o usuário não pode mais ver o que você está exibindo. O sistema pode chamar `onStart()` e `OnStop()` várias vezes durante toda a vida útil da atividade, como os suplentes de atividade entre estar visíveis e ocultas para o usuário.

O tempo de vida de primeiro plano de uma atividade acontece entre a chamada de `onResume()` e a chamada de `OnPause()`. Durante este tempo, a atividade está na frente de todas as outras atividades na tela e tem o foco de entrada do usuário. Uma atividade pode freqüentemente transitar dentro e fora do plano, por exemplo, `onPause()` é chamado quando o dispositivo vai hibernar ou quando uma caixa de diálogo aparece.

A figura 1 ilustra estes laços e os caminhos de uma atividade pode demorar entre estados. Os retângulos representam os métodos de retorno que você pode implementar para realizar operações quando as transições entre os estados de atividade.

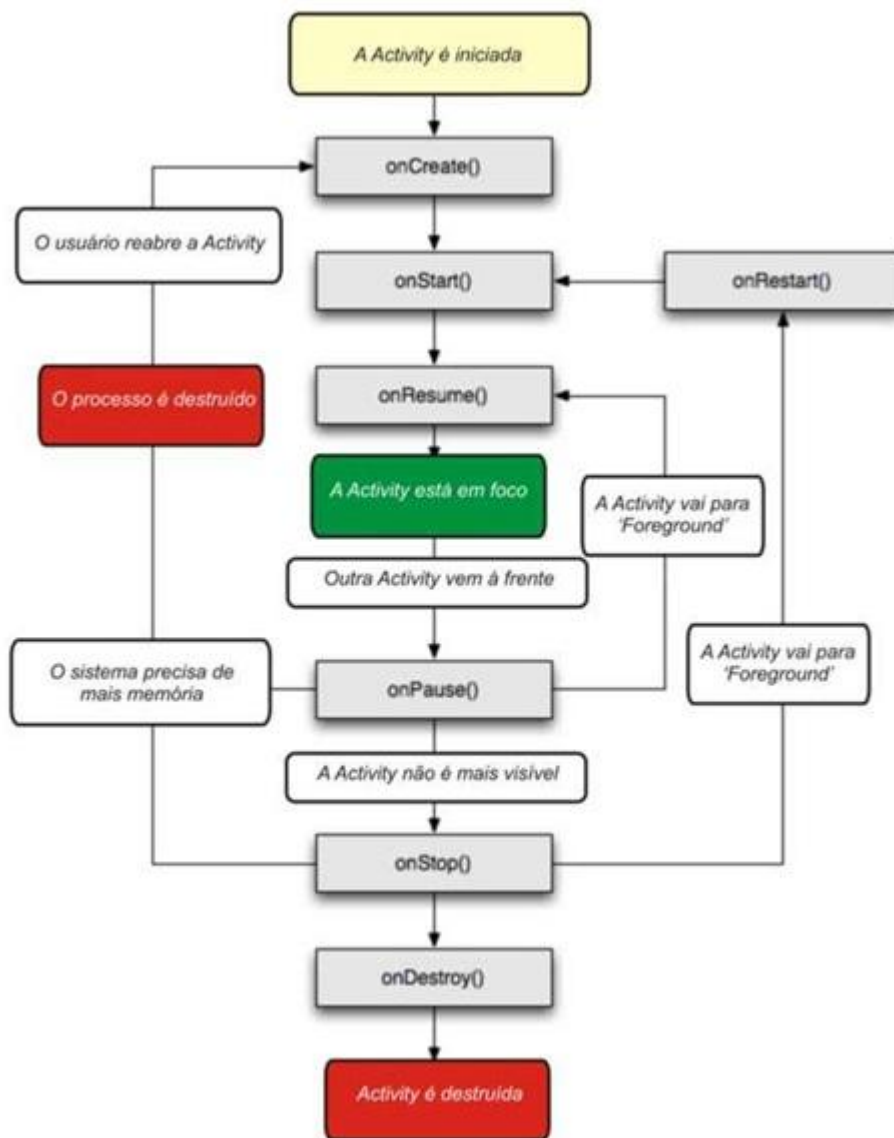


Figura 1. O ciclo de vida de atividade.

Os mesmos métodos de retorno de ciclo de vida estão listados abaixo, com descrição de cada um dos métodos de retorno de forma mais pormenorizada e localiza cada um dentro do ciclo de vida total da actividade, incluindo se o sistema pode matar a actividade após o método de retorno é concluída.

`onCreate()` Chamado quando a actividade é criada pela primeira vez. Este é o lugar onde você deve fazer todo o seu conjunto estático normal - criar pontos de vista, vincular dados a listas, e assim por diante. Neste método é

passado um objeto Bundle contendo estado anterior da atividade, se esse estado foi capturado anteriormente.

`onResume()` Chamado logo antes da atividade começa a interagir com o usuário. Neste ponto, a atividade está no topo da pilha de atividade, com a entrada do utilizador indo a ele.

`onPause()` Chamado quando o sistema está prestes a começar a retomar uma outra atividade. Este método é geralmente usado para confirmar as alterações não salvas dados persistentes, parar animações e outras coisas que podem estar consumindo CPU, e assim por diante. Ele deve fazer o que ele faz muito rapidamente, porque a próxima atividade não será retomado até que ele retorne.

`onStop()` Chamado quando a atividade já não é visível para o usuário. Isso pode acontecer porque ele está sendo destruído, ou porque outra atividade (seja um existente ou uma nova) foi retomada e está cobrindo.

`onDestroy()` Chamado antes da atividade é destruída. Esta é a última chamada que a atividade irá receber. Poderia ser chamado ou porque a atividade está terminando (um chamado `finish()` sobre ela), ou porque o sistema está temporariamente encerrando essa instância da atividade para economizar espaço. Você pode distinguir entre estes dois cenários com o método `isFinishing()`.

Salvando estado de atividade

A introdução de Gerenciamento do Ciclo de Vida da Atividade menciona brevemente que quando uma atividade é pausada ou interrompida, o estado da atividade é mantido. Isto é verdade, porque o objeto atividade ainda é mantida na memória quando é interrompido ou parado, todas as informações sobre os seus membros e o estado atual ainda estão vivos. Assim, quaisquer alterações do utilizador feitos dentro da atividade são mantidas de modo a que, quando a

atividade retorna para o primeiro plano (quando "recomeça"), tais alterações estão ainda lá.

No entanto, quando o sistema destrói uma actividade, a fim de recuperar a memória, o objeto da Atividade é destruído, de modo que o sistema não pode simplesmente continuar ela com o seu estado intacto. Em vez disso, o sistema deve recriar o objeto Activity se o usuário navega de volta para ele. No entanto, o usuário não tem conhecimento de que o sistema destruiu a atividade e recriou e, assim, provavelmente espera que a atividade a ser exatamente como era. Nesta situação, você pode garantir que informações importantes sobre o estado de atividade seja preservado pela implementação de um método de retorno adicional que permite que você salve as informações sobre o estado de sua atividade: `onSaveInstanceState()`.

O sistema chama `onSaveInstanceState()` antes de fazer a atividade vulnerável à destruição. O sistema passa este método num pacote em que você pode salvar informações de estado sobre a atividade como pares nome-valor, usando métodos como `putString()` e `putInt()`. Então, se o sistema mata o seu processo e o usuário navega de volta para sua atividade, o sistema recria a atividade e passa o pacote para ambos `onCreate()` e `onRestoreInstanceState()`. Usando um desses métodos, você pode extrair o seu estado salvo do pacote e restaurar o estado de atividade. Se não houver nenhuma informação de estado para restaurar, em seguida, o pacote transmitido para `lhe` é nulo (que é o caso quando a actividade é criado pela primeira vez).

Nota: Não há nenhuma garantia de que `onSaveInstanceState()` será chamado antes de sua atividade é destruída, porque há casos em que não será necessário salvar o estado (como quando o usuário deixa a sua actividade com o botão Voltar, porque o usuário deseja explicitamente fechar a atividade). Se o sistema chama `onSaveInstanceState()`, ele faz isso antes `onStop()` e possivelmente antes de `onPause()`.

No entanto, mesmo se você não fizer nada e não implementar `onSaveInstanceState()`, parte da atividade atual é restaurado pela

implementação da classe Atividade padrão de `onSaveInstanceState()`. Especificamente, a implementação padrão chama o `onSaveInstanceState` correspondente() para cada view no layout, que permite a cada atividade fornecer informações sobre si e que deve ser salvo. Quase todos os Widget no âmbito Android implementam este método adequadamente, de tal forma que nenhuma alteração visível na interface do usuário, sendo automaticamente salvas e restauradas quando sua atividade é recriada. Por exemplo, o widget `EditText` salva qualquer texto digitado pelo usuário e o widget `CheckBox` salva se ele está marcado ou não. O único trabalho exigido por você é fornecer uma identificação única (com o atributo `Android:id`) para cada widget que deseja salvar o estado. Se uma janela não tem um ID, não é possível ao sistema guardar o seu estado.

Você também pode parar explicitamente uma view no seu layout e salvar seu estado, definindo o atributo `android:saveEnabled` ou chamando o método `setSaveEnabled()`. Normalmente, você não deve desabilitar isso, mas você pode desejar restaurar o estado da interface do usuário atividade diferente.

Embora a implementação do padrão de `onSaveInstanceState()` salva as informações úteis sobre a interface do usuário de sua atividade, você ainda pode precisar substituí-lo para salvar informações adicionais. Por exemplo, você pode precisar salvar valores dos membros que mudaram durante a vida da atividade (que pode correlacionar os valores restaurados na interface do usuário, mas os membros que detêm esses valores de UI não são restaurados, por padrão).

Devido ao fato de que a implementação padrão de `onSaveInstanceState()` ajuda a salvar o estado da interface do usuário, se você substituir o método, a fim de salvar as informações adicionais de estado, você deve sempre chamar a implementação de superclasse de `onSaveInstanceState()` antes de fazer qualquer trabalho. Da mesma forma, você também deve chamar a implementação de superclasse de `onRestoreInstanceState()` se você substituí-lo, de forma a implementação padrão pode restaurar os estados.

Nota: Como `onSaveInstanceState()` não possui garantia de ser chamado, você deve usá-lo apenas para gravar o estado transitório da atividade (o estado da interface do usuário), você nunca deve usá-lo para armazenar dados persistentes. Em vez disso, você deve usar `onPause()` para armazenar dados persistentes (como dados que devem ser salvos em um banco de dados) quando o usuário deixa a atividade.

Uma boa maneira de testar a capacidade do seu aplicativo para restaurar seu estado é simplesmente rodar o dispositivo de modo que as mudanças de orientação de tela. Quando as mudanças de orientação da tela, o sistema destrói e recria a atividade, a fim de aplicar recursos alternativos que possam estar disponíveis para a configuração nova tela. Por esta razão, é muito importante que a sua actividade restaure completamente o seu estado quando ele é recriado, porque os usuários regularmente giram a tela ao usar aplicações.

Tratamento de alterações de configuração

Algumas configurações de dispositivo podem mudar durante a execução (como a orientação da tela, a disponibilidade, teclado e linguagem). Quando tal mudança ocorre, o Android recria a atividade em execução (o sistema chama `onDestroy()`, logo em seguida chama `onCreate()`). Este comportamento é projetado para ajudar a sua aplicação se a adaptar às novas configurações automaticamente recarregando sua aplicação com recursos alternativos que você forneceu (como layouts diferentes para orientações da tela e tamanhos diferentes).

Se você projetar corretamente sua atividade para lidar com um reiniciar devido a uma mudança de orientação da tela e restaurar o estado de atividade, como descrito acima, o aplicativo vai ser mais resistentes a outros eventos inesperados no ciclo de vida de atividade.

A melhor maneira de lidar com esse problema é salvar e restaurar o estado de sua atividade usando `onSaveInstanceState()` e `onRestoreInstanceState()` (ou `onCreate()`), como discutido na seção anterior.

Atividades de coordenação

Quando uma atividade começa outra, ambas transitam de ciclo de vida de experiência. A primeira atividade faz uma pausa e para (embora, não vá parar se ainda é visível no fundo), enquanto a outra atividade é criada. Caso haja dados compartilhados entre as atividades salvas no disco ou em outro lugar, é importante entender que a primeira atividade não está completamente parado antes do segunda ser criada. Em vez disso, o processo de começar a segunda se sobrepõe ao processo de paragem da primeira.

A ordem de chamadas de retorno do ciclo de vida é bem definido, particularmente quando as duas atividades estão no mesmo processo e uma é partida da outra. Aqui está a ordem das operações que ocorrem quando atividade A começa a atividade B:

O método `onPause()` da atividade A executa.

`OnCreate` da atividade de B chama os métodos `onStart()` e `onResume()` e executam em seqüência. (Atividade B agora tem o foco do usuário.)

Então, se Atividade A não é mais visível na tela, seu método `onStop()` executo.

Esta sequência previsível de callbacks do ciclo de vida permite-lhe gerir a transição de informações de uma atividade para outra. Por exemplo, se você tem que escrever para um banco de dados quando a primeira atividade é interrompida para que a atividade a seguir possa lê-lo, então você deve escrever para o banco de dados durante `onPause()` em vez de durante `onStop()`.

O ciclo de vida pode ser representado pelas seguintes funções:

`onCreate()` É a primeira função a ser executada quando uma Activity é iniciada. Geralmente é a responsável por carregar os layouts XML e outras operações de inicialização.

`onStart()` É chamada imediatamente após a `onCreate()` – e também quando uma Activity que estava em background volta a ter foco.

`onResume()` Assim como a `onStart()`, é chamada na inicialização da Activity e também quando uma Activity volta a ter foco.

`onPause()` É a primeira função a ser invocada quando a Activity perde o foco.

`onStop()` – Análoga à `onPause()`, só é chamada quando a Activity fica completamente encoberta por outra Activity (não é mais visível).

`onDestroy()` A última função a ser executada. Depois dela, a Activity é considerada “encerrada”.

`onRestart()` Chamada imediatamente antes da `onStart()`, quando uma Activity volta a ter o foco depois de estar em background.

Interface com Usuário Android

A Interface Gráfica com o Usuário (GUI) é um dos aspectos essenciais em um Sistema de Informação, permitindo que o usuário final desfrute de forma fácil e muitas vezes intuitiva das funcionalidades fornecidas pelo software (graças aos seus widgets).

Acompanhando sua evolução por meio das Eras da Computação (desde que os comandos eram realizados em uma tela de texto), percebemos que as Linguagens de Programação de alto nível têm enfatizado o seu uso e sua melhoria por meio de suas versões, sendo possível hoje em dia construirmos GUIs tão complexas quanto os requisitos exigidos pelo mundo real.

E quando o contexto é mobilidade, o desenvolvimento da mesma não pode ser adotado como é para Desktops ou Web, haja vista as restrições de recursos gráficos inerentes aos dispositivos móveis. Porém, esta realidade está mudando. Idealizadores de plataformas móveis vêm estudando novas formas de construção de interfaces gráficas para permitir mais flexibilidade e riqueza no desenvolvimento de aplicações móveis. E uma delas é Android, por meio de seus layouts que são totalmente definidos em XML.

Primeiramente, será descrito como a plataforma implementa a hierarquia de componentes de tela. Depois, serão explicadas as características dos principais layouts, que são:

LinearLayout, FrameLayout, AbsoluteLayout, RelativeLayout e TableLayout. Finalmente, será mostrado como se criar layouts complexos e como estendê-los para que estruturas gráficas mais complexas possam ser desenvolvidas, sendo para isso utilizado uma aplicação que simula a previsão do tempo para cidades do Brasil.

Hierarquia de Views e ViewGroups

Em Android, todos os componentes de interface gráfica são representados por subclasses de `android.view.View` que representam os componentes gráficos (os chamados widgets) como `TextView`, `Button`, `TextEdit`, `RadioButton`, `Checkbox`, etc; e a classe `android.view.ViewGroup`, que representa um container de Views e também ViewGroups.

Ela é a classe base para componentes de layouts, como `LinearLayout`, `FrameLayout`, `AbsoluteLayout`, `RelativeLayout`, `TableLayout`, etc.

O fato de um `ViewGroup` também ser composto por um ou mais `ViewGroups` é o fator que permite que layouts complexos (layouts aninhados) sejam desenvolvidos, como podemos observar na Figura 1.

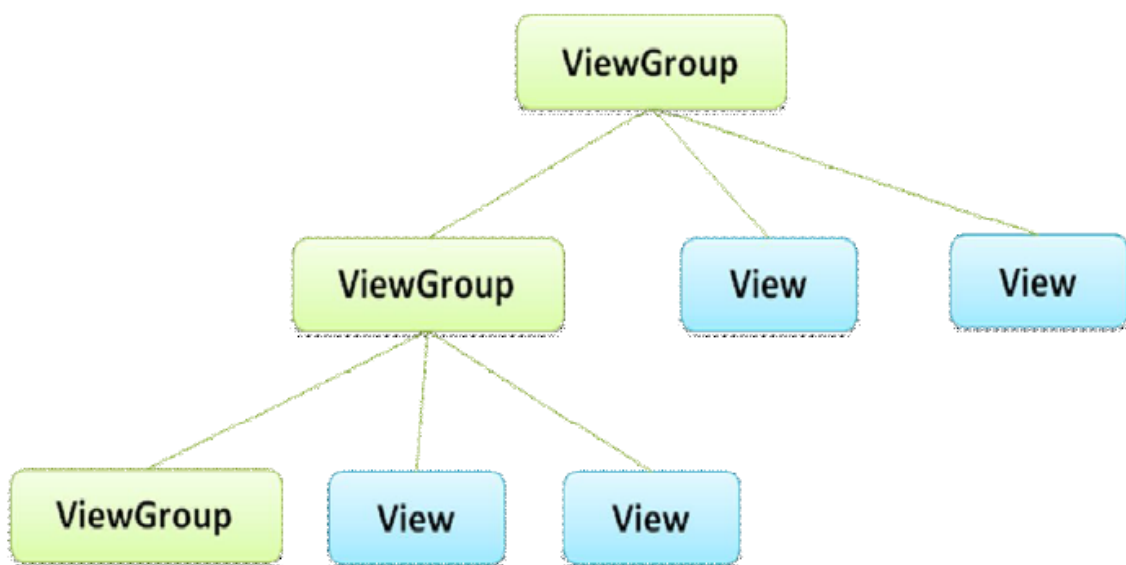


Figura 1. Hierarquia de componentes de tela.

Respeitando os parâmetros de layouts

Para que os componentes possam ser acomodados de acordo com o layout de seu pai, os mesmos devem manter uma relação de obediência. Sendo assim, cada filho deve configurar os seus parâmetros de layouts, por

meio da classe `ViewGroup.LayoutParams`, que permite que estes determinem suas propriedades de posição e tamanho referente ao layout de seus pais.

Portanto, todo layout deve possuir uma classe interna que obrigatoriamente estenda esta classe. Por exemplo, na Figura 2, podemos perceber que os filhos de `LinearLayout` (a raiz da árvore) possuem os seus parâmetros (`LinearLayout.LayoutParams`) para serem utilizados dentro de `LinearLayout`, como ocorre também para os filhos de `RelativeLayout`.

Todo layout possui propriedades de largura e altura (atributos `layout_width` e `layout_height`, respectivamente), porém outros podem declarar aspectos de margens e bordas.

É possível que estas propriedades sejam ajustadas manualmente (informando o valor exato), porém, é desejável que elas sejam configuradas possibilitando que o componente preencha o espaço do seu pai ou se ajuste de acordo com o seu conteúdo (inserindo o valor `fill_parent` ou o valor `wrap_content` em `android:layout_width` ou `android:layout_height`, respectivamente).

A seguir, serão descritos os principais layouts e suas características. Um aviso ao leitor: daqui por diante neste artigo, todos os layouts que serão explicados, serão definidos em arquivos XML separados (`linearlayout.xml`, `framelayout.xml`, etc). Dessa maneira, não é preciso sobrescrever o layout anterior para ver como ele se comporta na tela do emulador, sendo somente necessário alterar a referência (`R.layout.main`) no método `setContentView()`, de acordo com o layout a ser visualizado (`R.layout.linearlayout`, por exemplo). Isso nos permite construir vários layouts para serem utilizados em nossos projetos Android.

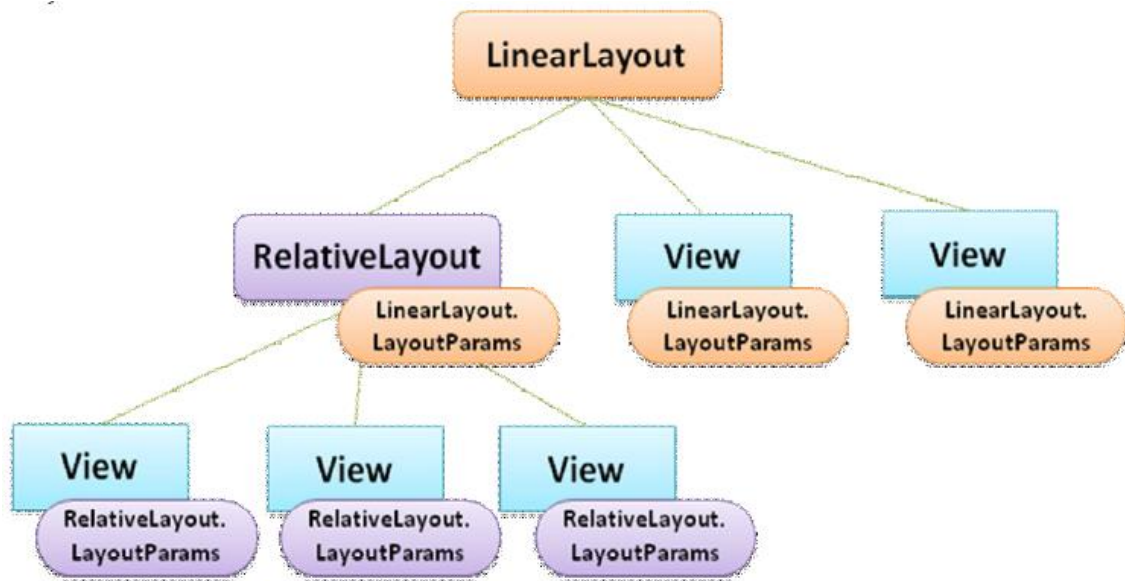


Figura 2. Os componentes respeitando os parâmetros de layouts dos seus pais.

LinearLayout

Este layout (criado por padrão no arquivo main.xml quando construímos um projeto) é utilizado para dispor seus componentes em uma única direção (por meio do atributo `android:layout_orientation`): vertical ou horizontal (Figura 3). Sendo assim, sua aplicação conterá somente uma lista de componentes caso estes sejam dispostos na vertical (Figura 4), passando o valor “vertical” para o atributo `android:orientation` de `<LinearLayout>`, ou uma única linha de componentes localizada na parte de cima do layout caso estes sejam dispostos na horizontal (Figura 5), atribuindo a este atributo o valor “horizontal”. Este layout respeita as margens entre os seus filhos e alinhamento (ao centro, à esquerda ou à direita. Em Android, chamamos este atributo de gravity)

É possível também atribuir, individualmente, pesos (Figura 6) para os componentes para que estes possam ocupar o restante do espaço do layout, evitando que pequenos objetos deixem espaço desnecessário no layout. Por padrão, todos os componentes adicionados possuem peso igual a 0. Porém, caso mais de um objeto utilize peso (Figura 7), então os componentes serão ajustados para preencherem igualmente o espaço de seu

pai. Para ver isto na prática, basta você adicionar o atributo `android:layout_weight="1"` para um ou mais Views e garantir que `android:orientation` de `<LinearLayout>` está com o valor "vertical". A Listagem 1 mostra um exemplo de `LinearLayout` em XML.

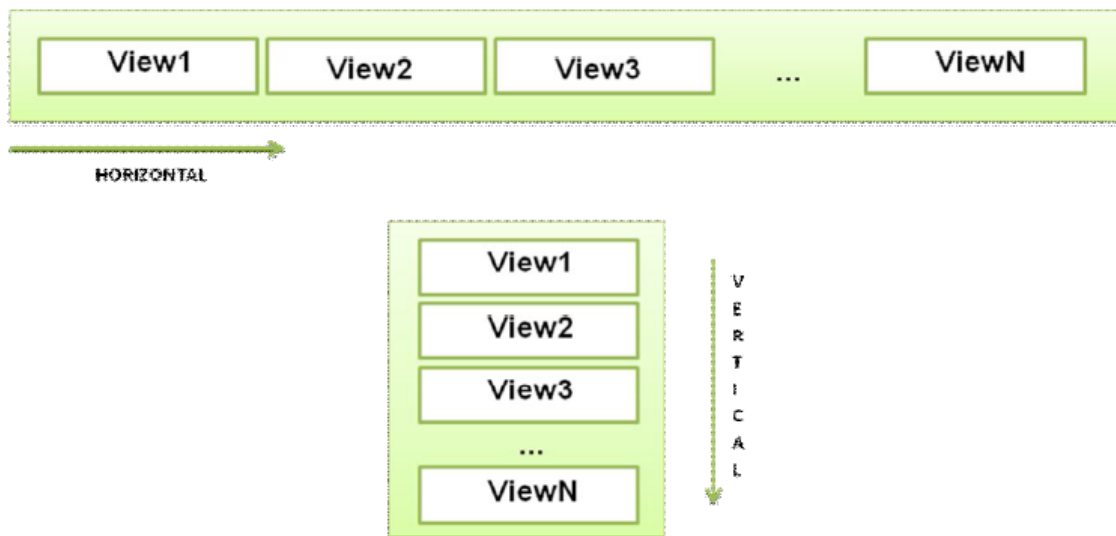


Figura 3. Adicionando componentes em uma única direção com `LinearLayout`.

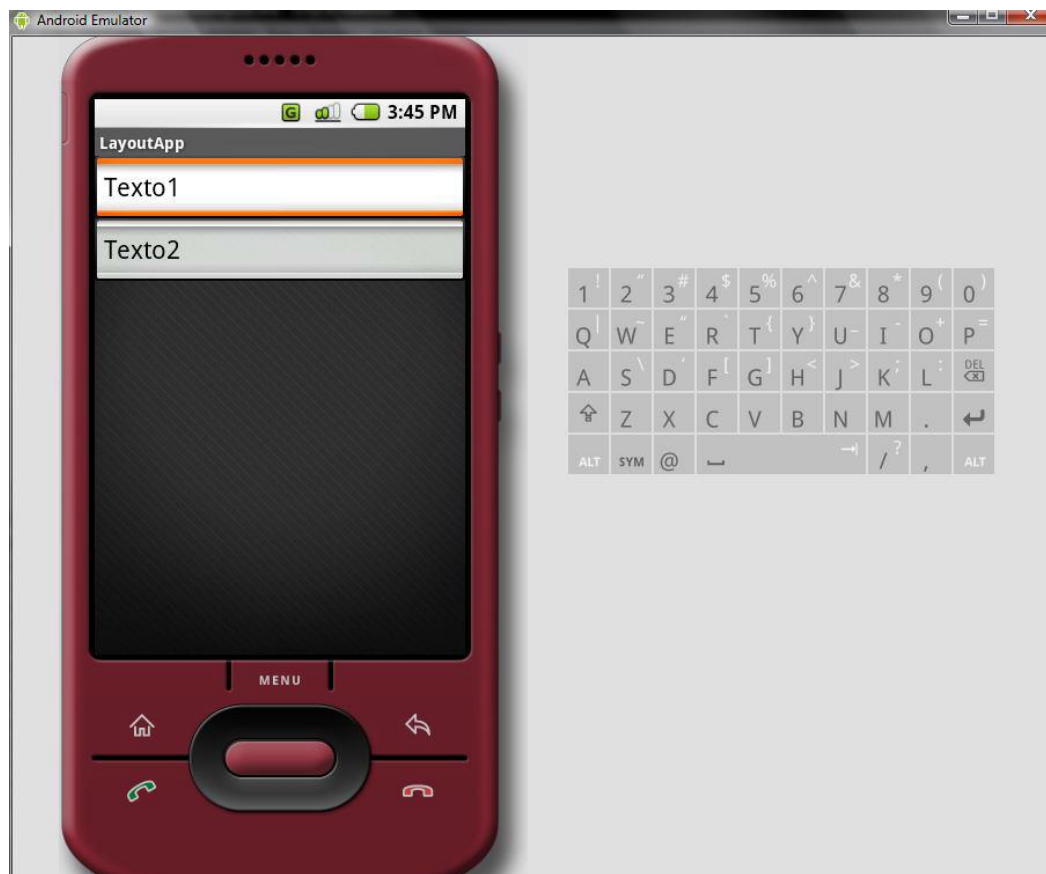


Figura 4. Exemplificando LinearLayout na direção vertical.

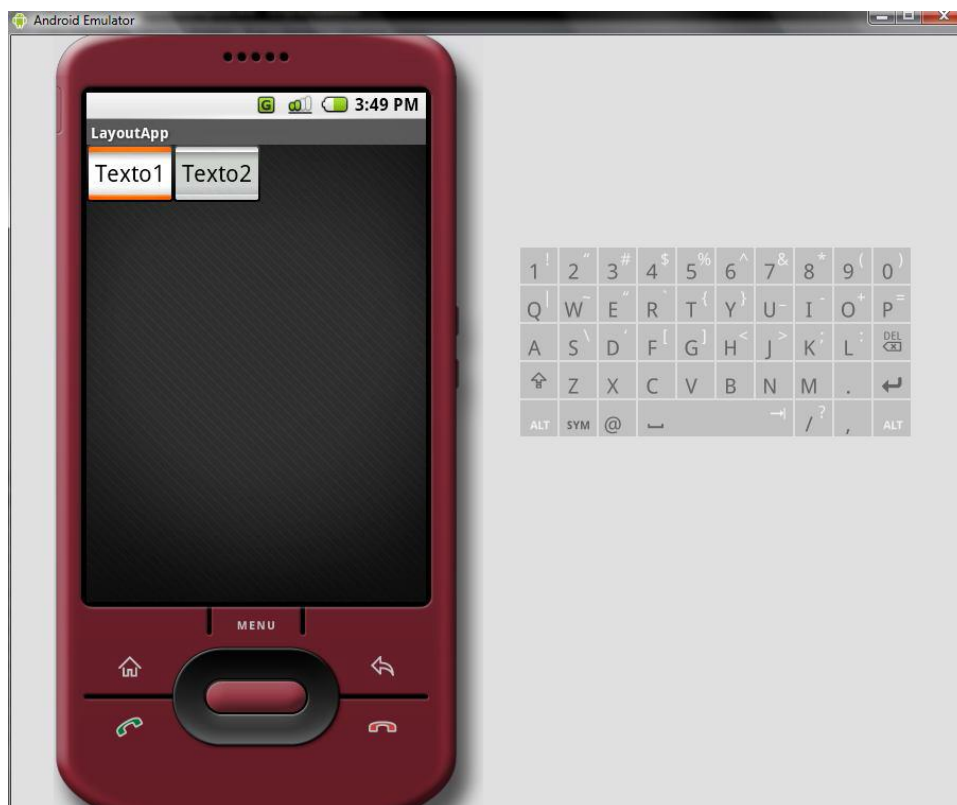


Figura 5. Exemplificando LinearLayout na direção horizontal.

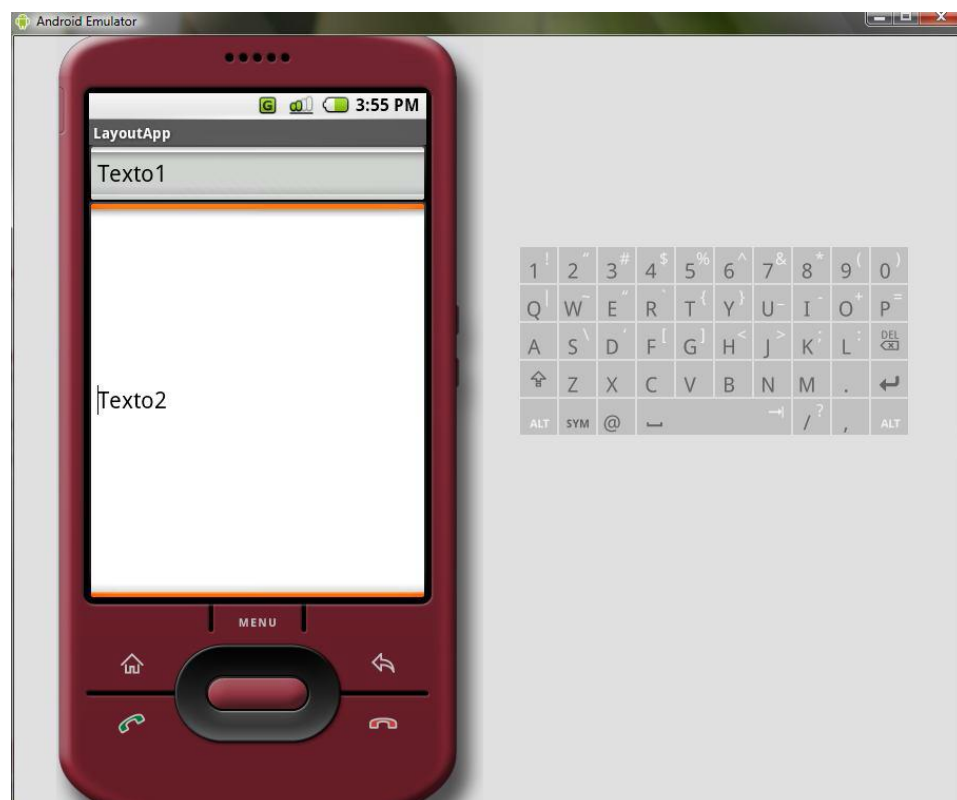


Figura 6. Atribuindo peso a um componente.

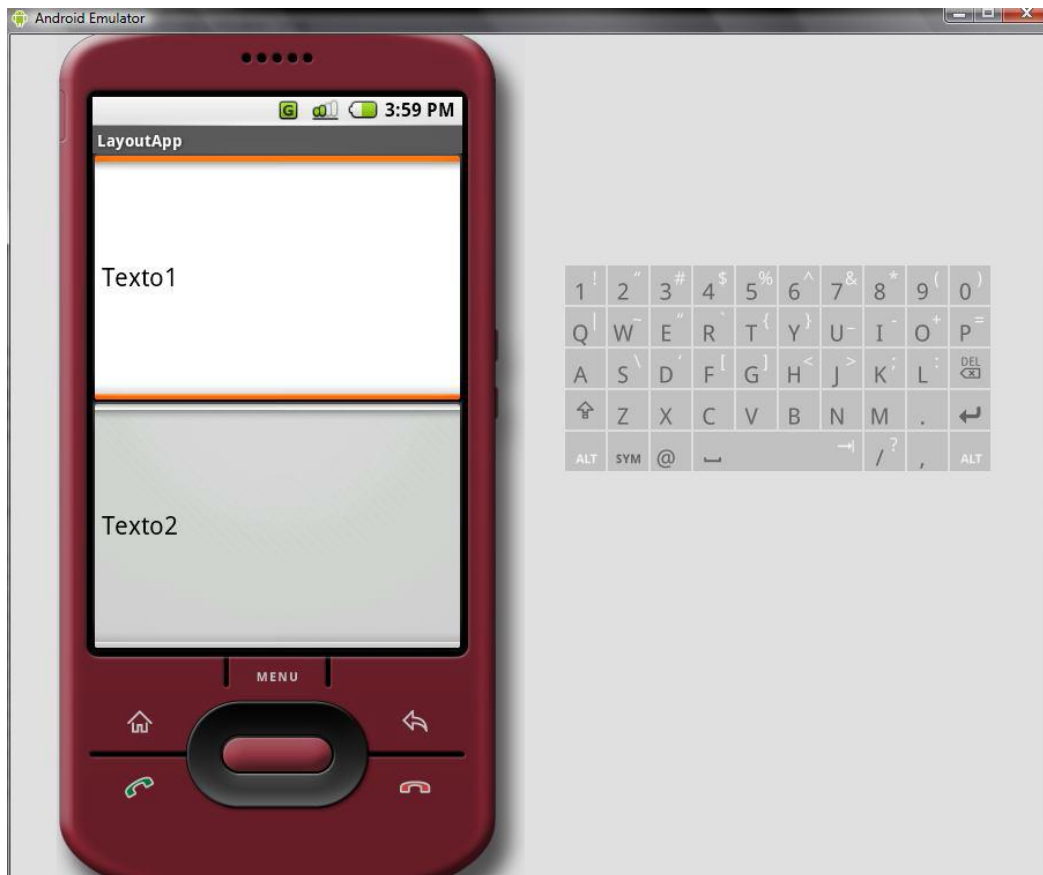


Figura 7. Atribuindo peso a mais de um componente.

Listagem 1. LinearLayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="fill_parent">
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Texto1" />
    <EditText android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Texto2" />
</LinearLayout>
```


FrameLayout

O FrameLayout arranja seus filhos de acordo com uma pilha de componentes que são adicionados, sendo que o topo da pilha contém o objeto que foi adicionado por último, como podemos perceber na pilha ao lado de FrameLayout (Figura 8), em que o primeiro componente na pilha é um View de cor azul e o do topo, de cor roxa. Podemos utilizá-lo quando, por exemplo, queremos usar várias imagens, onde uma é trocada (sobreposta) pela outra (como um slide de imagens) conforme vão sendo adicionadas.

O tamanho total de um FrameLayout é definido pelo seu maior filho mais o espaçamento (padding) e todos os componentes são agrupados no canto superior esquerdo do layout. Porém, a sua utilização comumente dá-se a partir de suas subclasses, como ImageSwitcher, ViewAnimator, ViewSwitcher, ScrollView, TabHost, etc (para maiores detalhes confira a referência desta classe em <http://code.google.com/android/reference/android/widget/FrameLayout.html>). A Listagem 2 mostra como este layout é implementado em XML e a Figura 9, exibe como ele é visualizado na tela do emulador.

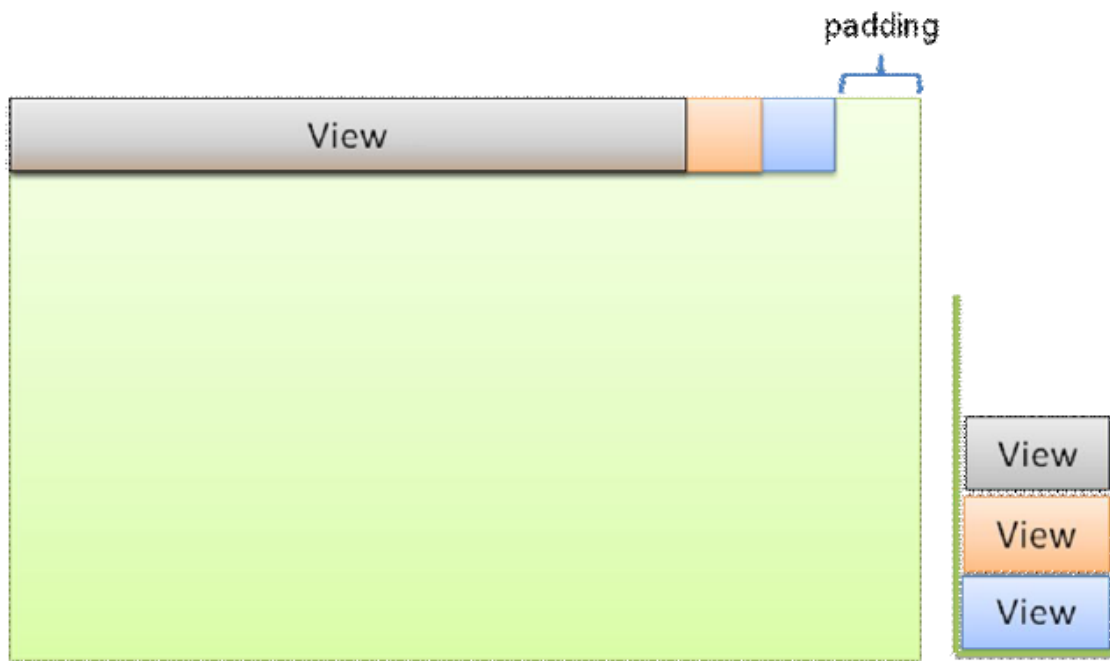


Figura 8. Criando uma pilha de componentes com FrameLayout.

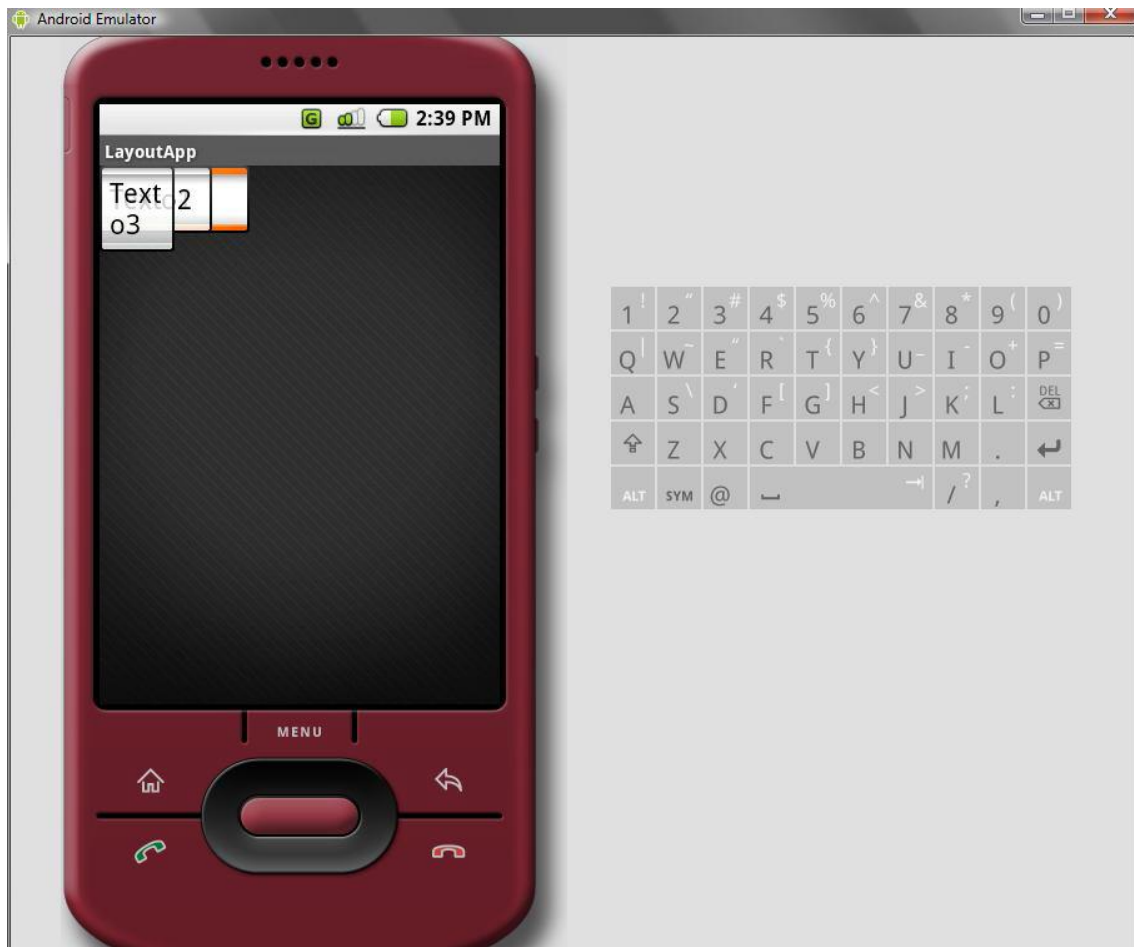


Figura 9. Sobreposição de componentes com FrameLayout.

Listagem 2. framelayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <EditText android:layout_width="120px"
        android:layout_height="wrap_content" android:text="Texto1"
        android:layout_weight="1" />
    <EditText android:layout_width="90px"
        android:layout_height="wrap_content" android:text="Texto2"
        android:layout_weight="1" />
    <EditText android:layout_width="60px"
```

```
        android:layout_height="wrap_content" android:text="Texto3" />
</FrameLayout>
```

AbsoluteLayout

Este tipo de layout organiza seus componentes de forma a implementar um plano cartesiano, no qual as suas posições x e y devem ser definidas para que estes possam ser posicionados de forma absoluta (Figura 10), sendo que os valores da coordenada x crescem da esquerda para a direita e da coordenada y, de cima para baixo. Caso os componentes não declarem as suas posições explicitamente, eles serão dispostos na posição (0,0). Este layout deve ser utilizado com cautela, pois, além dos objetos serem sobrepostos (como podemos ver os Views na posição (x4,y4) e (x7,y7)) caso as suas posições sejam informadas incorretamente, o layout da sua aplicação pode não se comportar como esperado caso ela seja desenvolvida para rodar em dispositivos com resoluções de telas diferentes (Figura 11 e Figura 12).

Para simularmos este efeito, mudaremos o skin do emulador, que é por padrão o HVGA-P 320x420 em formato retrato (vermelho), para o skin QVGA 240x320 em formato retrato (branco). Para isso, clicamos com botão direito do mouse em cima do projeto e vamos em Open Run Dialog....Depois, do lado direito, clicamos na aba Target e em Screen Size selecionamos QVGA-P. Rodamos a aplicação e podemos agora ver, que alguns componentes são posicionados quase fora da tela do dispositivo. A Listagem 3 exibe o layout em XML que foi utilizado nas figuras anteriores.

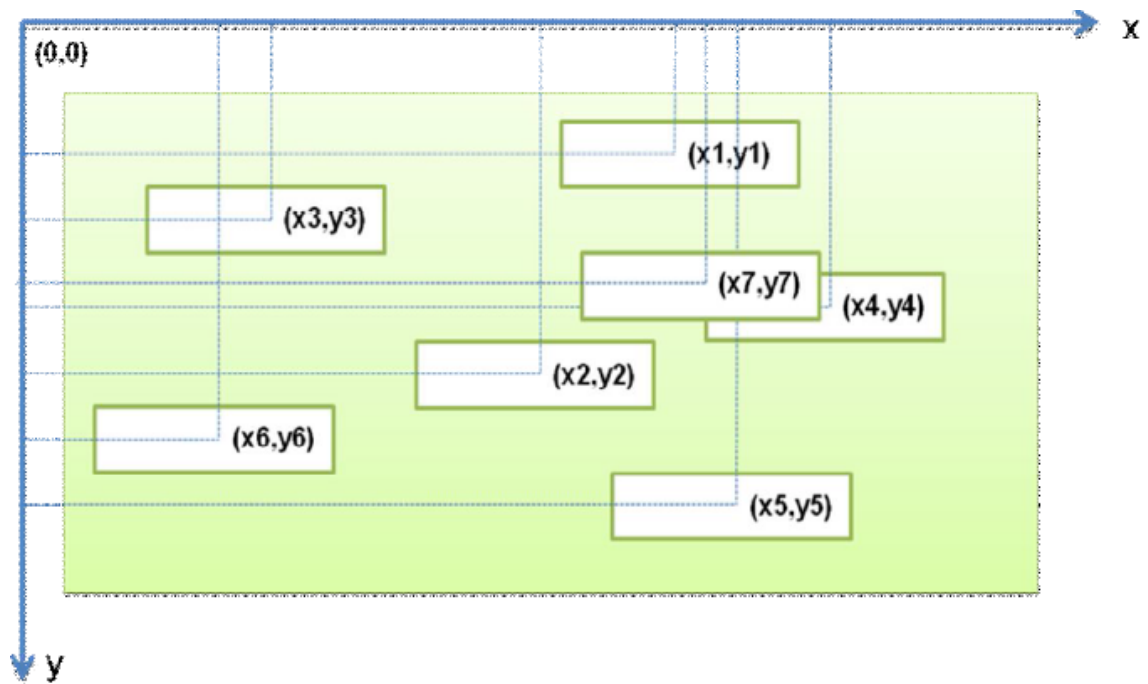


Figura 10. Utilizando o AbsoluteLayout.

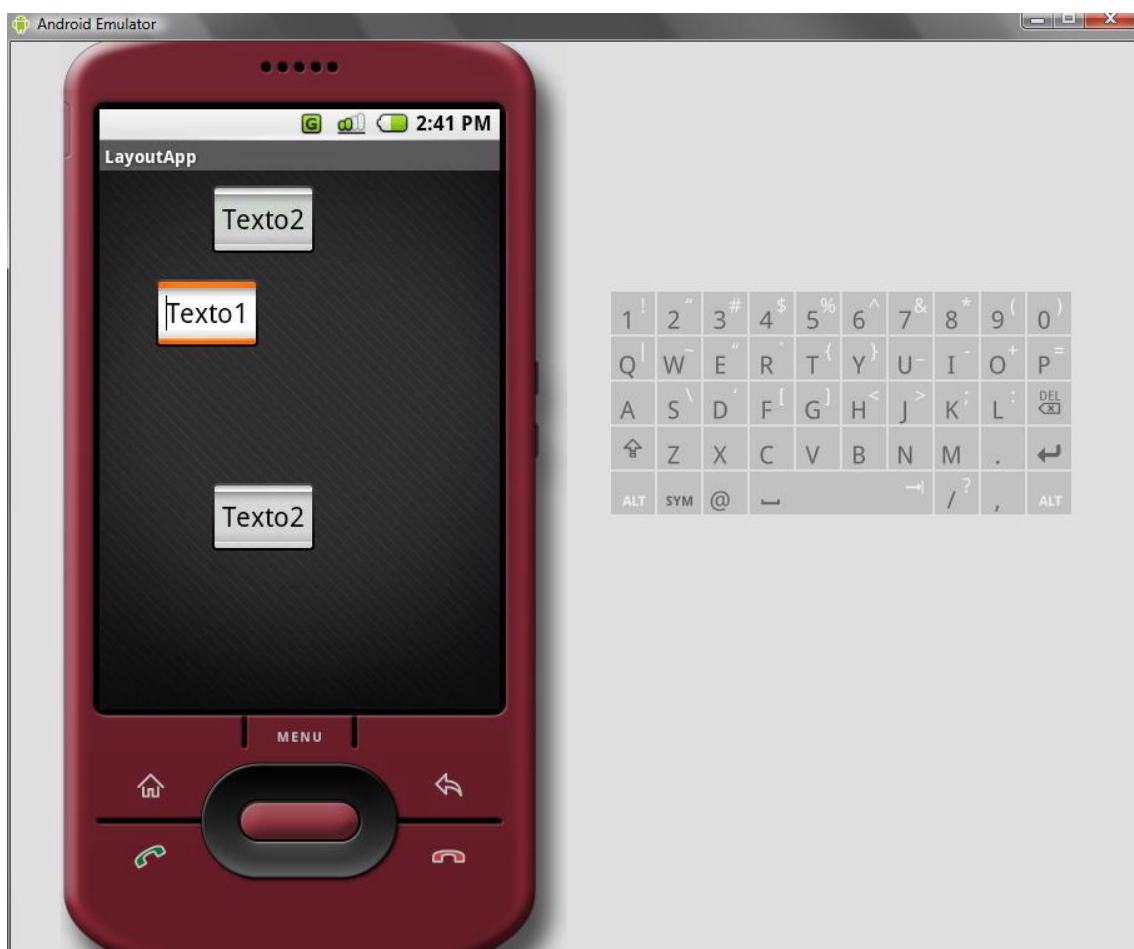


Figura 11. Dispondo os componentes de forma absoluta com AbsoluteLayout.



Figura 12. Problema que pode ocorrer em dispositivos com resoluções de tela diferente.

Listagem 3. absolutelayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="fill_parent">
    <EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Texto1"
```

```
        android:layout_x="45px"
        android:layout_y="87px" />
<EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Texto2"
        android:layout_x="90px"
        android:layout_y="12px" />
<EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Texto 3"
        android:layout_x="90px"
        android:layout_y="250px" />
</AbsoluteLayout>
```

RelativeLayout

Neste layout, os componentes são ajustados através de relacionamentos entre si ou ao seu pai. A Figura 13 exibe quatro componentes: um TextView, um EditText e dois Buttons. Perceba que o EditText com id textEntry possui um atributo below, informando que o mesmo deve ser posicionado embaixo do TextView label1. O mesmo ocorre com o Button okButton, onde este é ajustado em baixo do EditText textEntry e também é alinhado à direita referente ao seu pai. Um detalhe importante: caso um componente referencie (por meio da propriedade below, por exemplo) algum outro componente, este primeiro deve obrigatoriamente ser definido antes (definindo um nome no atributo android:id), caso contrário ocorrerá um erro em sua estrutura de layout. Porém, nem todos componentes precisam definir um id (como podemos ver o Button Cancel), sendo isso somente necessário se em algum local de seu layout este componente será referenciado. A Listagem 4 mostra um exemplo de tela de autenticação que utiliza RelativeLayout.

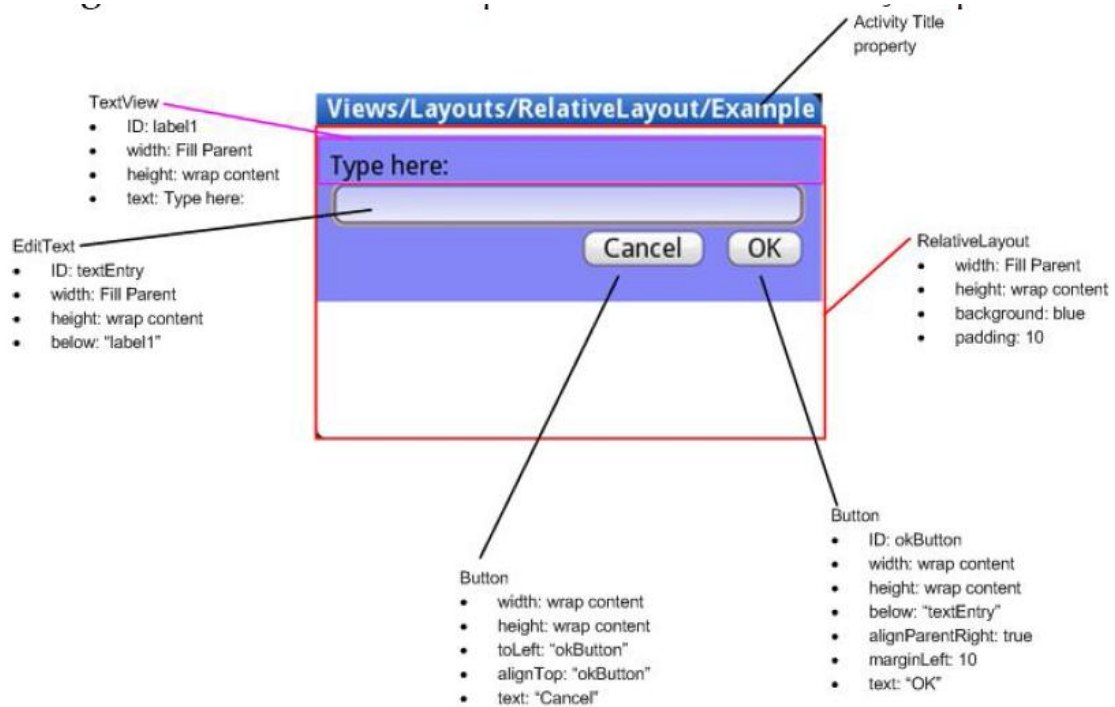


Figura 13. Utilizando componentes relativamente com RelativeLayout.

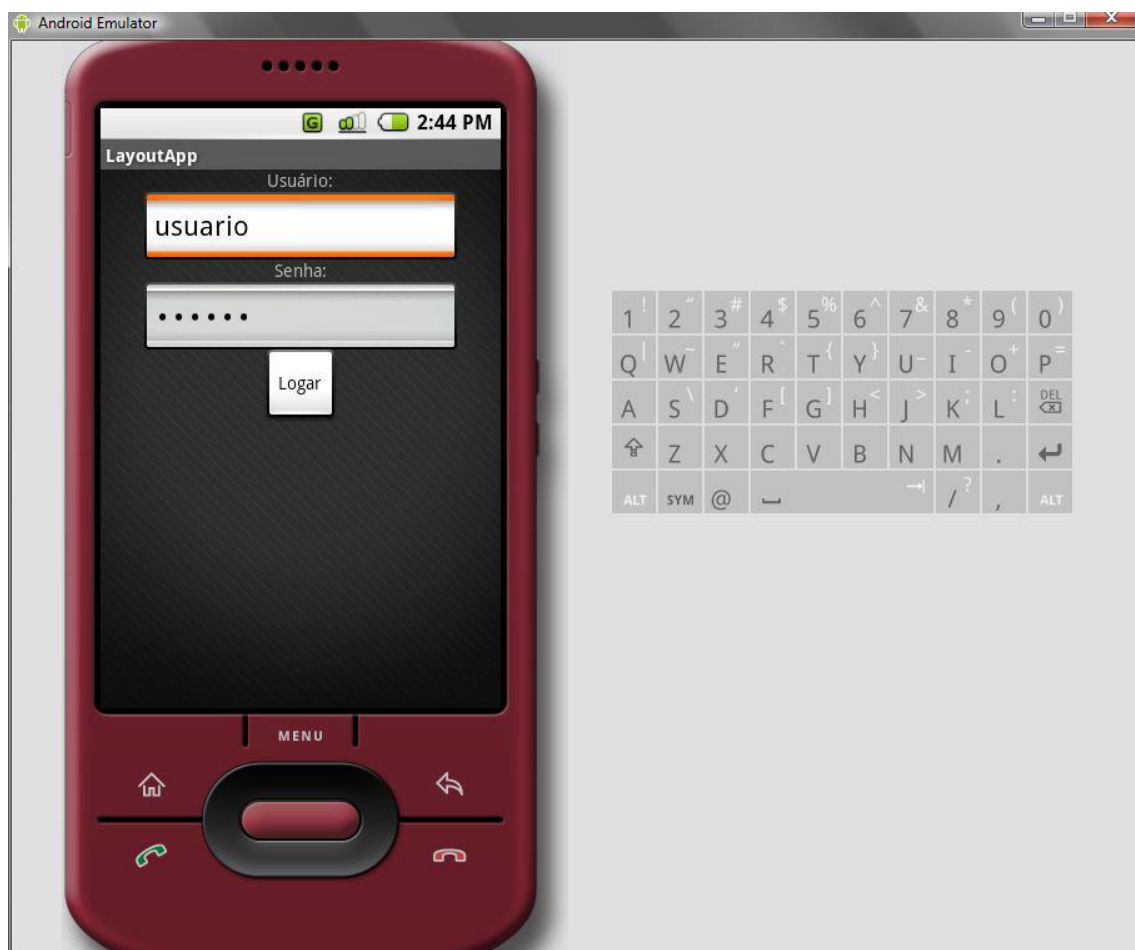


Figura 14. Ajustando os componentes relativamente com RelativeLayout.

Listagem 4. relativelayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="fill_parent">
    <TextView android:id="@+id/tvLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" android:text="Usuário:" />
    <EditText android:id="@+id/etLogin"
        android:layout_width="250px" android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" android:text="usuario"
        android:layout_below="@id/tvLogin" />
    <TextView android:id="@+id/tvSenha"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" android:text="Senha:"
        android:layout_below="@id/etLogin" />
    <EditText android:id="@+id/etSenha"
        android:layout_width="250px" android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" android:text="Texto2"
        android:password="true" android:layout_below="@id/tvSenha" />
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="Logar"
        android:layout_centerHorizontal="true"
        android:layout_below="@+id/etSenha" />
</RelativeLayout>
```


TableLayout

O TableLayout comporta seus filhos em linhas e colunas. Cada filho é representado pelo componente TableRow (que também é uma espécie de LinearLayout restrito na direção horizontal) que permite que uma ou mais células sejam adicionados horizontalmente, sendo que cada célula pode conter somente um único View (Figura 15).

O número de colunas é definido pela linha que tiver mais células. Este layout não mostra as linhas utilizadas para dividir TableRows, colunas ou células (mostradas em linhas tracejadas vermelhas). Conforme uma TableRow for sendo adicionado, o próximo será adicionado abaixo da anterior e assim sucessivamente. As células podem ser vazias e as colunas podem ser ocultadas, podem ser marcadas para preencherem o espaço restante da tela ou para que sejam compressíveis para forçar que estas sejam ajustadas até que completem o espaço restante da tela. Caso o atributo android:layout_width e android:layout_height não sejam declarados, este layout irá forçar para que a largura de cada componente seja automaticamente FILL_PARENT e a altura WRAP_CONTENT. A Listagem 5 exibe o arquivo XML definido para TableLayout e a Figura 16 exibe este layout no emulador.

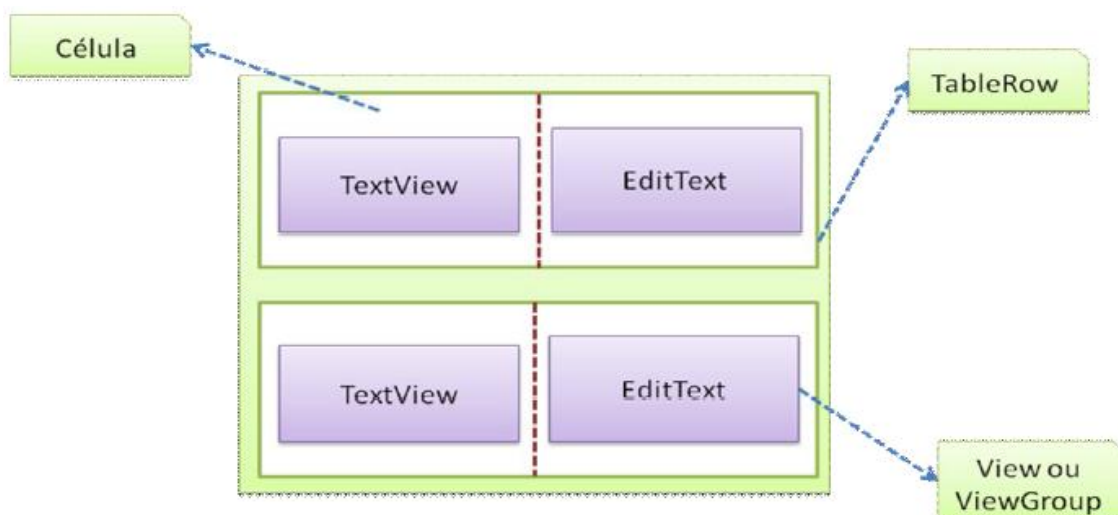


Figura 15. Representação de tabela como layout para componentes com TableLayout.

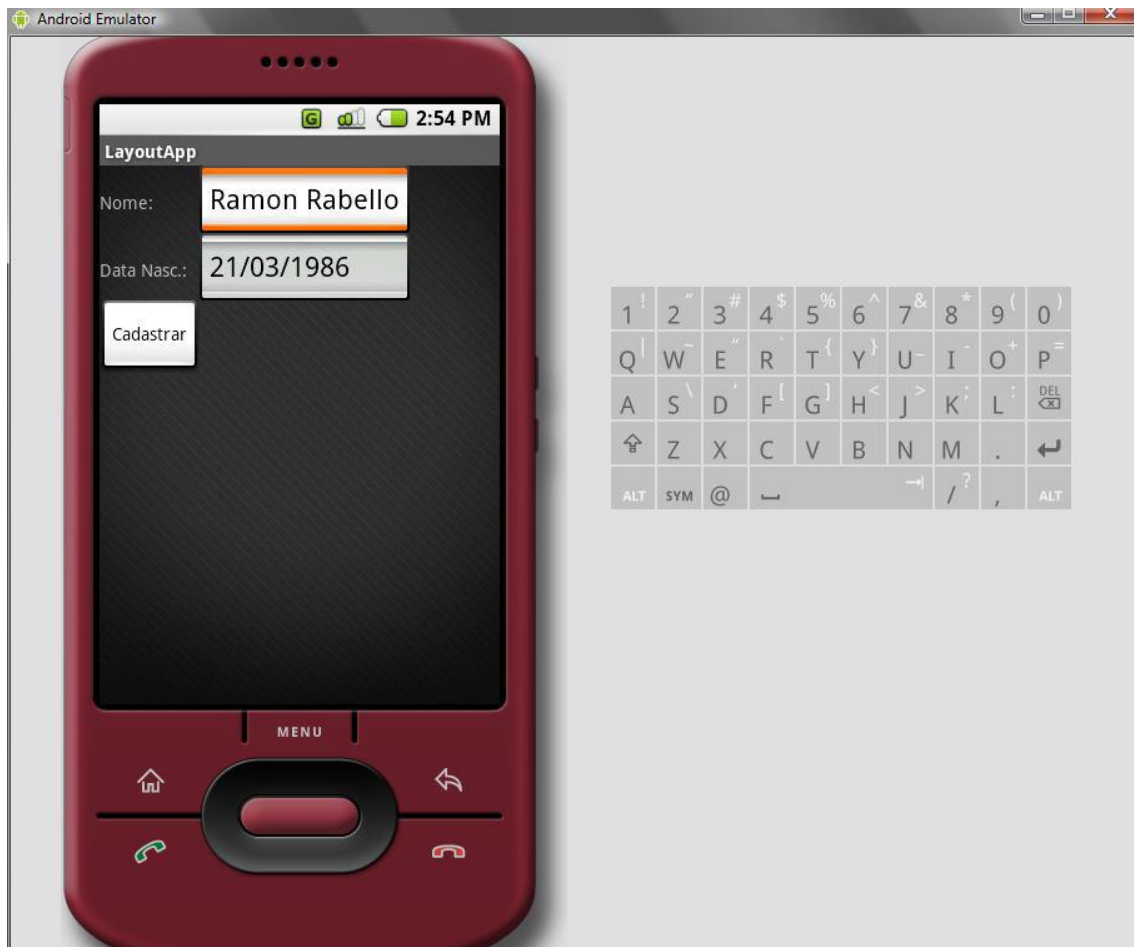


Figura 16. Exemplo de TableLayout no emulador.

Listagem 5. tablelayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="wrap_content"
    android:layout_height="fill_parent">
    <TableRow>
        <TextView android:text="Nome:" />
        <EditText android:text="Ramon Rabello" />
    </TableRow>
    <TableRow>
        <TextView android:text="Data Nasc.:" />
        <EditText android:text="21/03/1986" />
    </TableRow>
</TableLayout>
```

```
<TableRow>
    <Button android:text="Cadastrar" />
</TableRow>
</TableLayout>
```

Criando layouts complexos

Agora, reunindo tudo que foi aprendido até esta etapa, podemos construir aninhamentos de layouts para desenvolvermos nossas interfaces gráficas mais ricas e complexas, como mostra a Figura 17, na qual mais externamente temos um `TableLayout` que possui um `TableRow` que possui dois filhos: do lado esquerdo um `AbsoluteLayout` e do outro lado, um `LinearLayout` com direção vertical, ambos comportando `Views`. Perceba que, mesmo com esse nível de complexidade, todos os filhos (tantos os `Views` quanto os `ViewGroups`) se comportam especificamente de acordo com o layout pai graças às suas propriedades de parâmetros de layout (definidas nas subclasses de `ViewGroup.LayoutParams`). Às vezes, construir layouts pode se tornar uma atividade cansativa, haja vista o número de variáveis que deverão ser ajustadas manualmente no arquivo XML. Pensando nisso, foi desenvolvida uma ferramenta Open Source chamada `DroidDraw` que facilita o desenvolvimento não só destes layouts mas também dos widgets. Ela pode ser carregada tanto como um Applet ou em modo Standalone (executando um jar). Para baixá-la e maiores informações, acesse o link <http://droiddraw.org>. A Listagem 6 mostra como definimos esta estrutura no arquivo XML de layout e a Figura 18 mostra este layout no emulador.

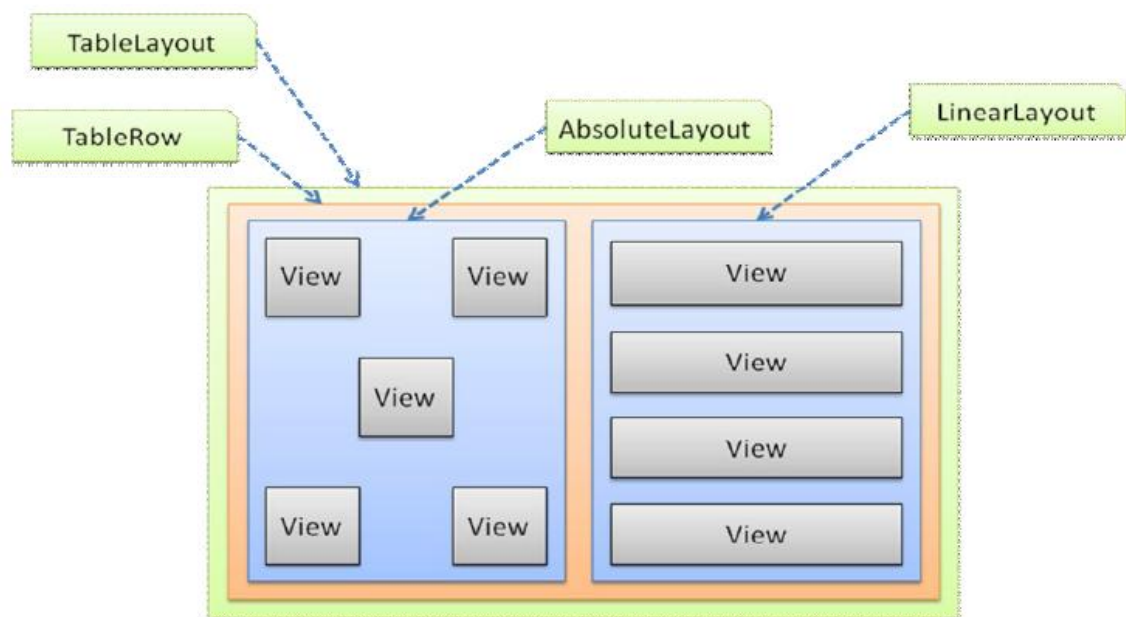


Figura 17. Aninhando layouts para construir layouts complexos.



Figura 18. Reproduzindo layouts complexos.

Listagem 6. complexlayout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TableRow>
        <AbsoluteLayout android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View"
                android:layout_x="10px" android:layout_y="123px" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View"
                android:layout_x="10px" android:layout_y="12px" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View"
                android:layout_x="66px" android:layout_y="67px" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View"
                android:layout_x="120px" android:layout_y="12px" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View"
                android:layout_x="120px" android:layout_y="123px">
            </Button>
        </AbsoluteLayout>
        <LinearLayout android:orientation="vertical">
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View" />
            <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="View" />
            <Button android:layout_width="wrap_content" />
        </LinearLayout>
    </TableRow>
</TableLayout>
```

```
        android:layout_height="wrap_content" android:text="View" />
    </LinearLayout>
</TableRow>
</TableLayout>
```

Estendendo layouts

Android não possui somente os componentes de layouts que foram explicados em seções anteriores. Eles são os principais e a plataforma ainda disponibiliza uma gama de layouts que utilizaram como base estes. A Tabela 1 mostra alguns exemplos destes layouts.

Porém, caso nenhum destes se enquadrem nas especificações gráficas da sua aplicação, a plataforma possui um framework que permite que layouts sejam estendidos para suprir as necessidades do seu projeto.

Para exemplificarmos, criaremos uma aplicação simples que permite simular (utilizando mocks) a previsão do tempo das cidades do Brasil. Para isso, será utilizada uma lista que, em vez de ser representada por um simples `ListView` (que é o componente que visualiza uma listagem de Views - por padrão, este componente exibe `TextViews` em sua listagem), conterá elementos que representam um layout, que mostrará o nome da cidade, a temperatura e uma imagem representando o status do clima. Achou interessante? Então vamos à prática! Criaremos um novo projeto e definiremos as nossas classes que serão utilizadas no novo layout. A seguir serão descritas cada classe.

Primeiramente, definiremos uma `Enumeration` (Listagem 7) que representará os possíveis status do clima: que no nosso exemplo será somente chuvoso, ensolarado ou nublado. Depois definimos uma classe (Listagem 8) que será o bean (modelo) responsável pela representação dos dados climáticos. Foi criado também o método utilitário `getStatusClimaResource()`, que será utilizado para referenciar as imagens (que estão no diretório `res/drawable/`) de acordo com a classe `R.java`, os quais representarão os status

do clima, sendo carregada uma imagem padrão já disponível na plataforma (`android.R.drawable.unknown_image`) caso nenhuma imagem possa ser carregada.

Seguindo, criamos outra classe, `ClimaAdapterView`, (Listagem 9) que irá estender `LinearLayout`. Tivemos que estendê-la, pois em Android não existe (ainda) um componente que exiba, em uma única linha da lista, tanto textos como imagens.

Então, definimos o construtor desta classe passando como parâmetro um `Context` (Interface que permite funcionalidades de um ambiente de aplicação, como carregamento de `Activities`, disparo de `Intents`, etc), que será a nossa `ListActivity` (subclasse indireta de `Context`) que será explicada futuramente (Listagem 11). Outro parâmetro é uma referência para o bean `Clima` que contém os dados climáticos encapsulados.

Chamamos o construtor de `LinearLayout` (`super(context)`) e passamos o `context`.

Agora, ajustamos a direção deste layout para horizontal, por meio do método `setOrientation()` (equivalente ao atributo `android:layout_orientation`) e configuramos os parâmetros de layout para cada objeto utilizando `LinearLayout.LayoutParams`, começando primeiramente para a cidade (`cidadeParams`).

Em `LinearLayout.Params`, passamos no seu construtor as informações de largura (100) e altura (`LayoutParams.WRAP_CONTENT`) e depois, no métodos `setMargins()` ajustamos as margens (todos com valor 1) da esquerda, de cima, da direita e de baixo deste objeto, respectivamente. Instanciamos o `TextView` `tvCidade`, passamos ao seu construtor um `context`, setamos seu texto utilizando o bean `clima`, o tamanho do texto (14f), a cor (`Color.WHITE`) e o tipo da fonte como negrito (`Typeface.DEFAULT_BOLD`). Adicionamos este `TextView` com os parâmetros de layout ao nosso layout chamando o método `addView()`. Configuramos depois os parâmetros de layouts

de `tvTemperatura` (`temperaturaParams`) e `ivStatusClima` (`statusClimaParams`), a seguir os adicionamos no nosso layout.

Definimos agora a nossa classe `ClimaAdapter` (Listagem 10) que realizará, de fato, a listagem de objetos `Climas`. Se você observar, ela estende `BaseAdapter`, que é a classe base para objetos que queiram servir como ponte entre os dados e a interface gráfica, que será utilizada para visualizá-los de acordo com alguma fonte de dados (arrays, Cursors, etc).

Ela é utilizada por `AdapterViews`, que são Views cujo seus filhos são determinados por um Adapter, como é o caso de `ListView`, que usa um `ListAdaper`. Resumindo: todo esse mecanismo é o padrão MVC (Modelo-Visão-Controle) “por trás dos panos”, que foi fortemente utilizado na plataforma Android para podermos realizar o binding de dados (modelo) com os Views (visão) utilizando Adapters (controle), diminuindo o grau de acoplamento entre ambos.

Continuemos a análise de nosso Adapter. Declaramos dois objetos: um `Context` representando nossa `ListActivity` e um `List` (`climas`) que conterá objetos `Climas` que serão visualizados na lista. Como `BaseAdapter` é abstrata, devemos implementar os métodos abstratos (na verdade, estes métodos pertencem à interface `Adapter`) que são: `getCount()`, utilizado para retornar o número de objetos dentro da lista, que será o tamanho de nossa `List`; `getItem()`, que retorna o item (`Object`) referente a uma posição na lista; `getItemId()` que retorna o id do item de acordo com uma posição na lista; e `getView()`, que recebe três parâmetros: a sua posição, um `View` e um `ViewGroup`, que será o layout ao qual este objeto pertence. Obtemos o objeto em determinada posição da lista chamando `climas.get(position)`, que no nosso caso será um `Clima`. Depois, instanciamos `ClimaAdapterView` que será a nossa extensão de `LinearLayout`. De uma maneira bem simples: a nossa lista conterá elementos que serão objetos da classe `ClimaAdapterView`.

Agora criaremos nossa Activity (`ClimaAdapterActivity`) responsável por visualizar nosso layout customizado. Observe que ela estende `ListActivity`

pois nossa aplicação utilizará uma lista. Sobrescrevemos `onCreate()` de `ListActivity`, que é o primeiro método a ser chamado quando a `Activity` é criada.

Depois, chamamos o método `onCreate()` da superclasse e carregamos o layout atual que será utilizado por meio do método `setContentView()`. Perceba em `main.xml` no diretório `res/layout/`, (Listagem 12) que nosso layout será carregado utilizando um `<LinearLayout>`. Aninhado a este existe a tag `<ListView>` com o atributo `android:id` que possui o valor `"@+id/android:list"` informando que a lista a ser carregada será uma prédefinida pela plataforma. A tag `<TextView>` representa um texto que mostrará uma mensagem caso não exista nenhum elemento na lista. Essa mensagem é referenciada pelo atributo `android:text` com o valor `"@string/sem_itens"`, que está no arquivo `strings.xml` (Listagem 13). Agora instanciaremos um `ArrayList` que representará a fonte de dados que passaremos para `ClimaAdapter` mais à frente. Agora, instanciamos e adicionamos na lista (climas) vários objetos `Clima`, passando no seu construtor o nome da cidade, a sua temperatura, e o status do clima, respectivamente.

Aqui vem uma das partes mais interessantes: instanciamos o objeto `ClimaAdapter` e passamos em seu construtor a referência `this`, representando a `ClimaAdapterActivity`, e o `ArrayList` de climas. Finalmente, chamamos o método `setListAdapter()` e passamos um `ClimaAdapter` (`climaAdapter`) como parâmetro, responsável pelo binding com os dados e o elemento que representará a lista. Na Figura 19, podemos visualizar no emulador o estágio final de nosso novo layout em ação.

ViewGroup	Descrição
Gallery	Exibe uma lista de imagens (com rolagem para a direita e esquerda).
GridView	Exibe uma grade com rolagem que possui linhas e colunas.
ListView	Exibe uma lista de Views (apenas uma coluna).
ScrollView	Uma coluna de elementos com rolagem na vertical.
Spinner	Exibe um único item por vez, de acordo com uma lista de dados, dentro de uma caixa de texto de uma única linha. Semelhante à uma listbox que pode realizar a rolagem horizontalmente ou verticalmente.
SurfaceView	Permite acesso direto a uma superfície de desenho, possibilitando que seus filhos sejam dispostos em cima esta superfície. Porém, este componente deve ser utilizado para aplicações que desejem desenhar pixels em vez de widgets.
TabHost	Componente que permite que várias abas sejam adicionadas, selecionadas e que possam reagir a eventos de cliques do mouse.
ViewFlipper	Uma lista que exibe um componente por vez, dentro de uma caixa de texto de uma única linha. Este componente pode ser configurado para trocar (exibir outro) componente de acordo com determinado intervalo, como se fosse um slideshow.
ViewSwitcher	O mesmo que ViewFlipper.

Tabela 1. Outros Layouts disponíveis em Android.

Listagem 7. StatusClima.java

```
package android.clima.adapter;
```

```
public enum StatusClima {
    DESCONHECIDO, ENSOLARADO, NUBLADO, CHUVOSO
}
```

Listagem 8. Clima.java

```
package android.clima.adapter;
```

```
public class Clima {
    public String cidade;
    private int temperatura;
    private StatusClima statusClima = StatusClima.DESCONHECIDO;
    /* construtor e gets/sets */
    public int getStatusClimaResource() {
        switch (statusClima) {
            case ENSOLARADO:
                return R.drawable.ensolarado;
            case NUBLADO:
```

```

        return R.drawable.nublado;
    case CHUVOSO:
        return R.drawable.chuvoso;
    }
    return android.R.drawable.unknown_image;
}
}

```

Listagem 9. ClimaAdapterView.java

```

package android.clima.adapter;
/* imports */
public class ClimaAdapterView extends LinearLayout {
    public ClimaAdapterView(Context context, Clima clima) {
        super(context);
        this.setOrientation(HORIZONTAL);
        LinearLayout.LayoutParams cidadeParams =
            new LinearLayout.LayoutParams(100,
                LayoutParams.WRAP_CONTENT);
        cidadeParams.setMargins(1, 1, 1, 1);
        TextView tvCidade = new TextView(context);
        tvCidade.setText(clima.getCidade());
        tvCidade.setTextSize(14f);
        tvCidade.setTextColor(Color.WHITE);
        tvCidade.setTypeface(Typeface.DEFAULT_BOLD);
        addView(tvCidade, cidadeParams);
        LinearLayout.LayoutParams temperaturaParams =
            new LinearLayout.LayoutParams(40,
                LayoutParams.WRAP_CONTENT);
        temperaturaParams.setMargins(2, 2, 2, 2);
        TextView tvTemperatura = new TextView(context);
        tvTemperatura.setText(Integer.toString(clima.getTemperatura())+"°C");
        tvTemperatura.setTextSize(14f);
    }
}

```

```

        tvTemperatura.setTypeface(Typeface.DEFAULT_BOLD);
        tvTemperatura.setTextColor(Color.WHITE);
        addView(tvTemperatura, temperaturaParams);
        LinearLayout.LayoutParams statusClimaParams =
            new LinearLayout.LayoutParams(25,
                LayoutParams.WRAP_CONTENT);
        ImageView ivStatusClima = new ImageView(context);
        ivStatusClima.setImageResource(clima.getStatusClimaResource());
        addView(ivStatusClima, statusClimaParams);
    }
}

```

Listagem 10. ClimaAdapter.java

```

package android.clima.adapter;
/* imports */
public class ClimaAdapter extends BaseAdapter {
    private Context context;
    private List<Clima> climas;
    /* constructor */
    public int getCount() {
        return climas.size();
    }
    public Object getItem(int position) {
        return climas.get(position);
    }
    public long getItemId(int position) {
        return position;
    }
    public View getView(int position, View convertView, ViewGroup parent) {
        Clima clima = climas.get(position);
        return new ClimaAdapterView(this.context, clima );
    }
}

```

Listagem 11. ClimaAdapterActivity.java

```
package android.clima.adapter;

/* imports */

public class ClimaAdapterActivity extends ListActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icle) {
        super.onCreate(icle);
        setContentView(R.layout.main);
        ArrayList<Clima> climas = new ArrayList<Clima>();
        Clima w = new Clima("Belém", 32, StatusClima.ENSOLARADO);
        climas.add(w);
        w = new Clima("São Paulo", 17, StatusClima.NUBLADO);
        climas.add(w);
        w = new Clima("Recife", 18, StatusClima.CHUVOSO);
        climas.add(w);
        w = new Clima("Rio de Janeiro", 22, StatusClima.ENSOLARADO);
        climas.add(w);
        ClimaAdapter climaAdapter = new ClimaAdapter(this, climas);
        setListAdapter(climaAdapter);
    }
}
```

Listagem 12. main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView android:id="@+id/android:list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"/>
```

```
<TextView android:id="@+id/android:empty"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:text="@string/sem_itens"/>
</LinearLayout>
```

Listagem 13. strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ClimaAdapterView</string>
    <string name="sem_itens">Não há itens na lista</string>
</resources>
```

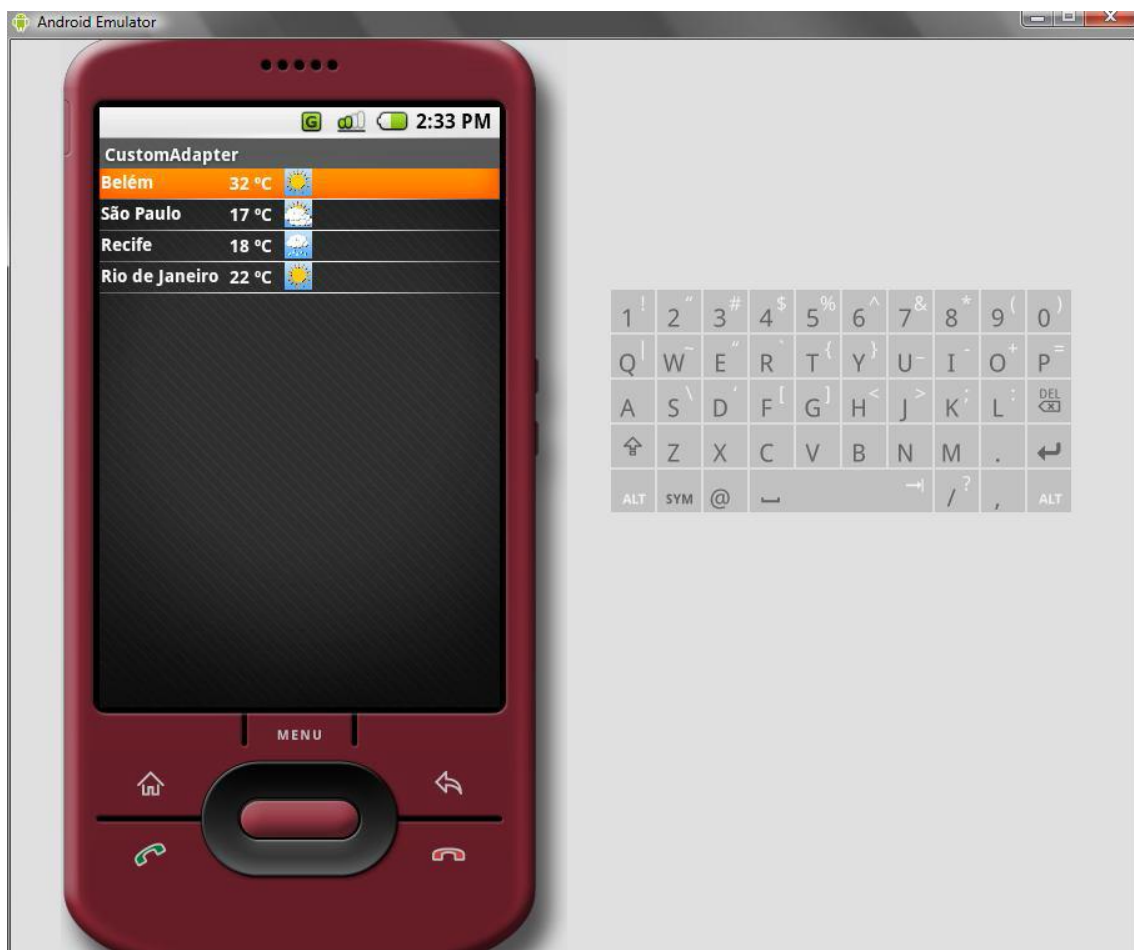


Figura 19. A aplicação carregada utilizando a nossa extensão de LinearLayout.

Button

Um botão consiste em um texto ou um ícone (ou texto e um ícone) que comunica que ação ocorre quando o usuário o pressiona.



Dependendo se você quer um botão com o texto, um ícone, ou ambos, você pode criar o botão em seu layout de três maneiras:

Com texto, usando a classe Button:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
... />
```

Com um ícone, usando a classe ImageButton:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
... />
```

Com um texto e um ícone, usando a classe Botão com o atributo `android:drawableLeft`:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
```

```
android:drawableLeft="@drawable/button_icon"  
... />
```

Respondendo a Clique em Eventos

Quando o usuário clica em um botão, o objeto Button recebe um evento on-clique.

Para definir o manipulador de eventos de clique de um botão, adicione o atributo `android:onClick` para o elemento `<Button>` em seu layout XML. O valor para este atributo deve ser o nome do método que você deseja chamar, em resposta a um evento de clique. A atividade de hospedagem do layout deve, então, implementar o método correspondente.

Por exemplo, aqui está um layout com um botão usando `android:onClick`:

```
<?xml version="1.0" encoding="utf-8"?>  
<Button xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/button_send"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/button_send"  
    android:onClick="sendMessage" />
```

Dentro da atividade que hospeda esse layout, o método a seguir manipula o evento clique:

```
/** Called when the user touches the button */  
public void sendMessage(View view) {  
    // Do something in response to button click  
}
```


O método que declara o atributo `android:onClick` deve ter uma assinatura exatamente como mostrado acima. Especificamente, o método deve:

- Ser público
- Retornar vazia
- Definir uma exibição como seu único parâmetro (esta será a vista que foi clicado)

Usando um `OnClickListener`

Você também pode declarar o manipulador de eventos de clique de forma pragmática, em vez de em um layout XML. Isto pode ser necessário se você instanciar o botão em tempo de execução ou você precisa declarar o comportamento clique em uma subclasse.

Para declarar o manipulador de eventos por meio de programação, criar um objeto `View.OnClickListener` e atribuí-lo ao botão chamando `setOnClickListener (View.OnClickListener)`. Por exemplo:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

Aplicando Styling no seu botão

A aparência do seu botão (imagem de fundo e fonte) pode variar de um dispositivo para outro, porque os dispositivos de diferentes fabricantes, muitas vezes têm estilos diferentes do padrão para controles de entrada.

Você pode controlar exatamente como os controles são decorados com um tema que você aplica a sua aplicação inteira. Por exemplo, para garantir que todos os dispositivos rodando Android 4.0 e superior usam o tema Holo em seu aplicativo, declarar `android:theme="@android:style/Theme.Holo"` em seu manifesto no elemento `<application>`.

Para personalizar botões individuais com um fundo diferente, especifique o atributo `android:background` com um recurso drawable ou cor. Alternativamente, você pode aplicar um estilo para o botão, que funciona de forma semelhante aos estilos HTML para definir propriedades de estilo múltiplas tais como o fundo, tipo de letra, tamanho e outros.

Botão sem fronteiras

Um projeto que pode ser útil é um botão de "sem fronteiras". Botões sem margens lembram botões básicos, exceto que eles não têm fronteiras ou de fundo, mas ainda mudam a aparência durante estados diferentes, como quando clicado.

Para criar um botão sem bordas, aplicar o estilo `borderlessButtonStyle` ao botão. Por exemplo:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    style="?android:attr/borderlessButtonStyle" />
```

Fundo personalizado

Se você quiser realmente redefinir a aparência do seu botão, você pode especificar um fundo personalizado. Em vez de fornecer um bitmap

simples ou cor, no entanto, o fundo deve ser um recurso de lista local que muda a aparência de acordo com o estado atual do botão.

Você pode definir a lista de estados em um arquivo XML que define três diferentes imagens ou cores a serem usadas para os diferentes estados de botão.

Para criar uma lista drawable estado para o fundo do botão:

- Crie três bitmaps para o fundo do botão que representam o padrão, pressionado, e estados de botão focados.
- Para garantir que suas imagens se encaixam botões de vários tamanhos, criar os bitmaps como Nine Patch-bitmaps.
- Colocar os bitmaps em res/drawable /directory do seu projeto. Certifique-se de cada bitmap é chamado a refletir adequadamente o estado do botão que cada um representa, como button_default.9.png, button_pressed.9.png, e button_focused.9.png.
- Crie um novo arquivo XML na res/drawable /diretório (nome dele algo como button_custom.xml). Insira o seguinte XML:

```
<?xml version="1.0" encoding="utf-8"?>
<selector
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/button_pressed"
        android:state_pressed="true" />
  <item android:drawable="@drawable/button_focused"
        android:state_focused="true" />
  <item android:drawable="@drawable/button_default" />
</selector>
```

Isto define um único recurso drawable, que vai mudar a sua imagem com base no estado atual do botão.

O primeiro <item> define o bitmap para usar quando o botão é pressionado (ativado).

O segundo <item> define o bitmap para usar quando o botão está focado (quando o botão é realçado utilizando o trackball ou direcional).

O terceiro <item> define o bitmap para usar quando o botão está no estado padrão (é nem pressionado nem focalizado).

Nota: A ordem dos elementos <item> é importante. Quando isso é referenciado no drawable, os elementos <item> são percorridos em ordem para determinar qual é apropriado para o estado do botão atual. Porque o bitmap padrão é passado, só é aplicada quando as condições do Android:state_pressed e android:state_focused têm ambos valor falso.

Este arquivo XML representa agora um único recurso drawable e quando referenciado por um botão para o seu fundo, a imagem exibida será alterada com base nesses três estados.

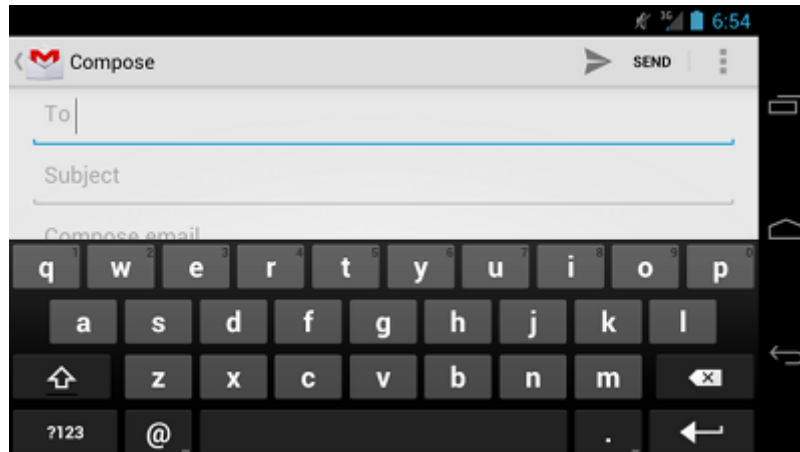
Depois, basta aplicar o arquivo XML drawable como o fundo do botão:

```
<Button
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage"
    android:background="@drawable/button_custom" />
```

TextField

Um campo de texto permite que o usuário digite um texto em sua app. Ele pode ser uma linha única qualquer ou multi-linha. Tocar em um campo de texto coloca o cursor e exibe automaticamente o teclado. Além de digitação, campos de texto permitem uma variedade de outras atividades, como a seleção de texto (cortar, copiar, colar) e olhar dados através de autopreenchimento.

Você pode adicionar um campo de texto para você layout com o objeto EditText. Normalmente devemos fazê-lo em um layout XML com um elemento <EditText>.



Especificando o tipo de teclado

Os campos de texto podem ter tipos diferentes de entrada, tais como número, data, senha ou endereço de e-mail. O tipo determina que tipo de caracteres são permitidos dentro do campo, e podem ser otimizados utilizando-se o melhor teclado virtual, aquele que possui o layout voltado para os caracteres usados com frequência.

Você pode especificar o tipo de teclado que você deseja para o objeto EditText com o atributo android:inputType. Por exemplo, se você deseja que o usuário insira um endereço de e-mail, você deve usar o tipo de entrada textEmailAddress:

```
<EditText
    android:id="@+id/email_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/email_hint"
    android:inputType="textEmailAddress" />
```

Existem vários tipos de entrada diferentes disponíveis para diferentes situações. Aqui estão alguns dos valores mais comuns para `android:inputType`:

- "text" - Teclado de texto normal.
- "textEmailAddress" - Teclado de texto normal com o caractere @.
- "textUri" - Teclado de texto normal com a tecla /.
- "number" - Teclado numérico básico.
- "phone" – Teclado estilo telefone.



Figura 1. O tipo padrão de entrada de texto.



Figura 2. O tipo de entrada `textEmailAddress`.



Figura 3. O tipo de entrada de telefone.

Controlando outros comportamentos

O `android:inputType` também permite que você especifique certos comportamentos teclado, como se deve capitalizar todas as palavras novas ou usar recursos como sugestões de auto-completar e ortografia.

O atributo `andróide:inputType` permite combinações bit a bit para que você possa especificar o layout de um teclado e um ou mais comportamentos de uma só vez.

Aqui estão alguns dos valores de entrada comuns do tipo que definem os comportamentos de teclado:

- "textCapSentences" - Teclado de texto normal que capitaliza a primeira letra de cada frase.
- "textCapWords" - Teclado de texto normal que capitaliza cada palavra. Bom para títulos ou nomes de pessoas.
- "textAutoCorrect" - Teclado de texto normal que corrige palavras comumente com erros ortográficos.
- "textPassword" - Teclado de texto normal, mas os caracteres digitados aparecem pontos.
- "textMultiLine" - Teclado de texto normal que permitem que os usuários de entrada seqüências longas de texto que incluem quebras de linha (retorno de carro).

Por exemplo, aqui está como você pode coletar um endereço postal, capitalizar cada palavra, e desativar sugestões de texto:

```
<EditText
    android:id="@+id/postal_address"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/postal_address_hint"
    android:inputType="textPostalAddress|
        textCapWords|
        textNoSuggestions" />
```

Especificando Ações Teclado

Além de mudar o tipo de teclado de entrada, o Android permite que você especifique uma ação a ser feita quando os usuários tenham concluído a sua entrada. A ação é especificada pelo botão que aparece no lugar da tecla de retorno de carro e a ação a ser feita, como "Search" ou "Enviar".



Figura 4. Se você declarar `android: imeOptions = "actionSend"`, o teclado inclui a ação de envio.

Você pode especificar a ação, definindo o atributo `android:imeOptions`. Por exemplo, aqui está como você pode especificar a ação de envio:

```
<EditText
    android:id="@+id/search"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/search_hint"
```



```
android:inputType="text"
android:imeOptions="actionSend" />
```

Se você não especificar explicitamente uma ação de entrada, o sistema tenta determinar se existem quaisquer subseqüentes `android:focusable`.

Respondendo a eventos do botão de ação

Se você tiver especificado uma ação de teclado para o método de entrada usando o atributo `Android:imeOptions` (como "actionSend"), você pode usar o evento de ação específico usando um `TextView.OnEditorActionListener`. A interface `TextView.OnEditorActionListener` fornece um método de retorno chamado `onEditorAction()` que indica o tipo de ação chamado com uma identificação de ação, como `IME_ACTION_SEND` ou `IME_ACTION_SEARCH`.

Por exemplo, aqui está como você pode fazer quando o usuário clica no botão Enviar no teclado:

```
EditText editText = (EditText) findViewById(R.id.search);
editText.setOnEditorActionListener(new OnEditorActionListener() {
    @Override
    public boolean onEditorAction(TextView v, int actionId, KeyEvent event) {
        boolean handled = false;
        if (actionId == EditorInfo.IME_ACTION_SEND) {
            sendMessage();
            handled = true;
        }
        return handled;
    }
});
```

A definição de um label do botão de ação personalizada

Se o teclado é muito grande para razoavelmente dividir o espaço com a aplicação subjacente (tais como quando um aparelho celular é na horizontal), então fullscreen é acionado. Neste modo, um botão de ação marcado é exibido ao lado da entrada. Você pode personalizar o texto do botão, definindo o atributo andróide: imeActionLabel:

```
<EditText
    android:id="@+id/launch_codes"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:hint="@string/enter_launch_codes"
    android:inputType="number"
    android:imeActionLabel="@string/launch" />
```

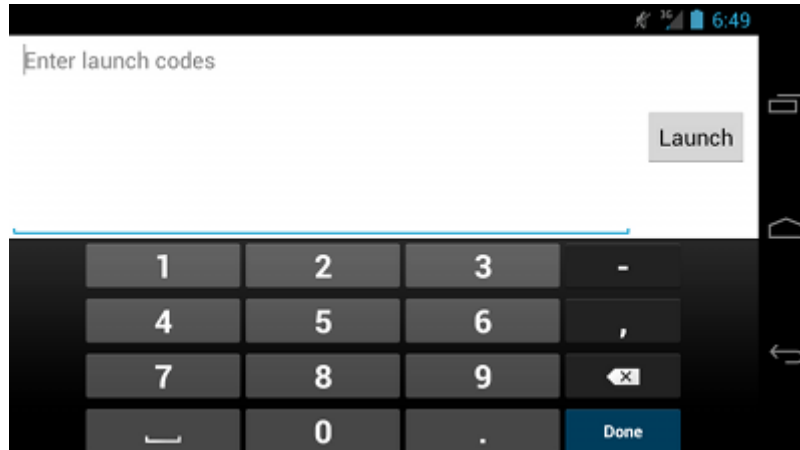


Figura 5. Uma etiqueta de ação personalizada com android: imeActionLabel.

Proporcionando Sugestões auto-completar

Se você quiser dar sugestões para os usuários, você pode usar uma subclasse de EditText chamada AutoCompleteTextView. Para implementar auto-completar, você deve especificar um (@link android.widget.Adapter) que fornece as sugestões de texto. Existem vários tipos de adaptadores

disponíveis, dependendo da proveniência dos dados, como a partir de um banco de dados ou um array.

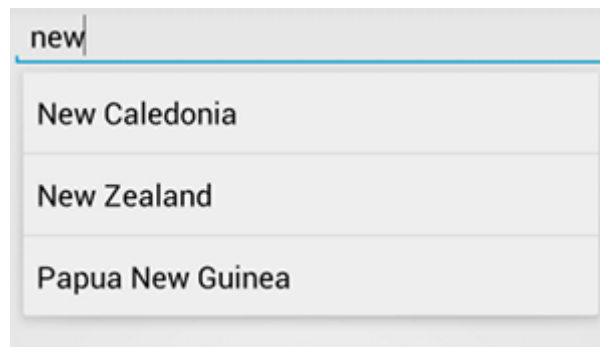


Figura 7. Exemplo de AutoCompleteTextView com sugestões de texto.

O procedimento a seguir descreve como configurar uma AutoCompleteTextView que oferece sugestões a partir de uma matriz, usando ArrayAdapter:

Adicione o AutoCompleteTextView para o seu layout. Aqui está um layout com apenas o campo de texto:

```
<?xml version="1.0" encoding="utf-8"?>
<AutoCompleteTextView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/autocomplete_country"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Definir a matriz que contém todas as sugestões de texto. Por exemplo, aqui está uma matriz de nomes de países que é definido em um arquivo de recurso (res/valores/strings.xml):

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="countries_array">
        <item>Afghanistan</item>
        <item>Albania</item>
```

```

        <item>Algeria</item>
        <item>American Samoa</item>
        <item>Andorra</item>
        <item>Angola</item>
        <item>Anguilla</item>
        <item>Antarctica</item>
        ...
    </string-array>
</resources>

```

Em sua atividade ou fragmento, use o código a seguir para especificar o adaptador que fornece as sugestões:

```

// Get a reference to the AutoCompleteTextView in the layout
AutoCompleteTextView textView = (AutoCompleteTextView)
findViewById(R.id.autocomplete_country);
// Get the string array
String[] countries =
    getResources().getStringArray(R.array.countries_array);
// Create the adapter and set it to the AutoCompleteTextView
ArrayAdapter<String> adapter =
    new ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, countries);
textView.setAdapter(adapter);

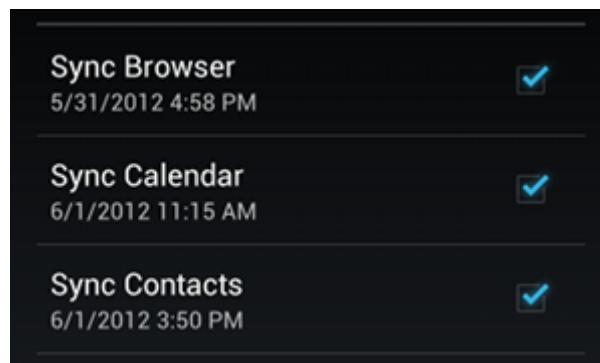
```

Aqui, um novo `ArrayAdapter` é inicializado para vincular cada item na matriz de países a um `TextView` que existe no layout `simple_list_item_1` (este é um layout fornecido pelo Android que proporciona uma aparência padrão para texto em uma lista).

Em seguida, atribuir o adaptador para o `AutoCompleteTextView` por chamando `setAdapter()`.

Checkboxes

Caixas de verificação permitem que o usuário selecione uma ou mais opções de um conjunto. Normalmente, você deve apresentar cada opção de seleção em uma lista vertical.



Para criar cada opção de seleção, crie um `CheckBox` em seu layout. Porque um conjunto de opções de caixa de seleção permite ao usuário selecionar vários itens, cada caixa é gerida separadamente e você deve registrar um verificador de clique para cada um.

Respondendo a Clique em Eventos

Quando o usuário seleciona uma caixa de seleção, o objeto `CheckBox` recebe um evento `onClick`.

Para definir o manipulador de eventos de clique para uma caixa de seleção, adicione o atributo `android:onClick` para o elemento `<CheckBox>` em seu layout XML. O valor para este atributo deve ser o nome do método que você deseja chamar em resposta a um evento de clique. A atividade de hospedagem do layout deve, então, implementar o método correspondente.

Por exemplo, aqui estão um par de objetos `CheckBox` em uma lista:

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>

```

Dentro da atividade que hospeda esse layout, o método a seguir manipula o evento clique para ambas as opções:

```

public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)

```

```
        // Cheese me
    else
        // I'm lactose intolerant
        break;
    // TODO: Veggie sandwich
}
}
```

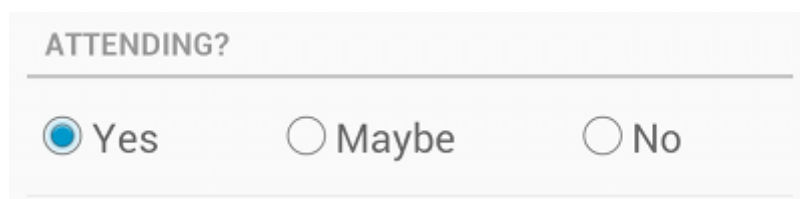
O método que declarado no atributo `android:onClick` deve ter uma assinatura exatamente como mostrado acima. Especificamente, o método deve:

- Ser público
- Volta vazia
- Definir uma exibição como seu único parâmetro (esta será a vista que foi clicado)

Dica: Se você precisa de mudar o estado de botão de rádio (tais como ao carregar um `CheckBoxPreference`), use o método `setChecked(boolean)` ou de `toggle()`.

Radio Buttons

Botões de rádio permitem que o usuário selecione uma opção de um conjunto. Você deve usar os botões de rádio para conjuntos opcionais que são mutuamente exclusivos, se você acha que o usuário precisa ver todas as opções disponíveis, lado a lado. Se não é necessário mostrar todas as opções de lado a lado, use um spinner.



ATTENDING?

☒ Yes ☐ Maybe ☐ No

Para criar cada opção de botão de rádio, deve-se criar um `RadioButton` em seu layout. No entanto, porque os botões de rádio são mutuamente exclusivos, é necessário agrupá-los dentro de um `RadioGroup`. Agrupando-os, o sistema garante que apenas um botão de opção pode ser selecionado de cada vez.

Respondendo a Clique em Eventos

Quando o usuário seleciona um dos botões de rádio, o objeto `RadioButton` correspondente recebe um evento `onClick`.

Para definir o manipulador de eventos de clique de um botão, adicione o atributo `android:onClick` para o elemento `<RadioButton>` em seu layout XML. O valor para este atributo deve ser o nome do método que você deseja chamar, em resposta a um evento de clique. A atividade de hospedagem do layout deve, então, implementar o método correspondente.

Por exemplo, aqui estão alguns objetos `RadioButton` par:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
```



```
        android:onClick="onRadioButtonClicked"/>
    </RadioGroup>
```

Nota: O RadioGroup é uma subclasse de LinearLayout que tem uma orientação vertical por padrão.

Dentro da atividade que hospeda esse layout, o método a seguir manipula o evento clique para ambos os botões de rádio:

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
                break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
                break;
    }
}
```

O método que declarado no atributo android:onClick deve ter uma assinatura exatamente como mostrado acima. Especificamente, o método deve:

- Ser público
- Volta vazia
- Definir uma exibição como seu único parâmetro (esta será a vista que foi clicado)

Dica: Se você precisa de mudar o estado de botão de rádio (tais como ao carregar um `CheckBoxPreference` salvos), use o método `setChecked(boolean)` ou `toggle()`.

Toggle Buttons

Um botão de alternância permite que o usuário altere a configuração entre dois estados.

Você pode adicionar um botão básico de alternância para o seu layout com o objeto `ToggleButton`. Android 4.0 (API nível 14) introduz um outro tipo de botão de alternância chamado um interruptor que fornece um controle deslizante, que você pode adicionar com um objeto `Switch`.



Botões de alternância



Interruptores (no Android 4.0 +)

O `ToggleButton` e controles switch são subclasses de `CompoundButton` e funcionam da mesma maneira, de modo que você pode implementar seu comportamento da mesma maneira.

Respondendo a Clique em Eventos

Quando o usuário seleciona um `ToggleButton` e `Switch`, o objeto recebe um evento `onClick`.

Para definir o manipulador de eventos, clique em adicionar o atributo `android:onClick` ao `<ToggleButton>` ou `<switch>` em seu layout XML. O valor para este atributo deve ser o nome do método que você deseja chamar,

em resposta a um evento de clique. A atividade de hospedagem do layout deve, então, implementar o método correspondente.

Por exemplo, aqui está um ToggleButton com o android: atributo onClick:

```
<ToggleButton
    android:id="@+id/togglebutton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textOn="Vibrate on"
    android:textOff="Vibrate off"
    android:onClick="onToggleClicked"/>
```

Dentro da atividade que hospeda esse layout, o método a seguir manipula o evento clique:

```
public void onToggleClicked(View view) {
    // Is the toggle on?
    boolean on = ((ToggleButton) view).isChecked();

    if (on) {
        // Enable vibrate
    } else {
        // Disable vibrate
    }
}
```

O método que declarado no atributo android:Click deve ter uma assinatura exatamente como mostrado acima. Especificamente, o método deve:

- Ser público
- Volta vazia

- Definir uma exibição como seu único parâmetro (esta será a vista que foi clicado)

Dica: Se você precisar alterar o estado mesmo, use o método `setChecked(boolean)` ou de `toggle()` para alterar o estado.

Usando um `OnCheckedChangeListener`

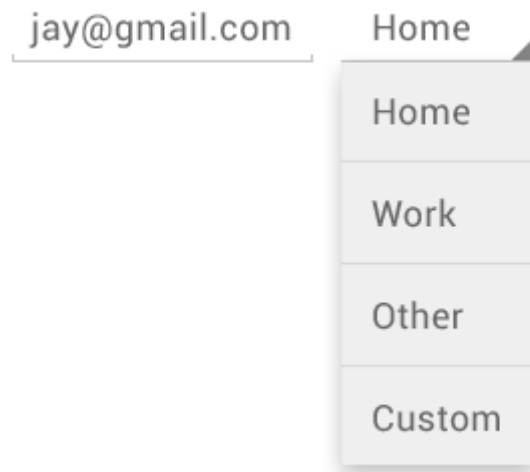
Você também pode declarar um manipulador de eventos de clique de forma pragmática, em vez de em um layout XML. Isto pode ser necessário se você instanciar o `ToggleButton` ou `Switch` em tempo de execução ou você precisa declarar o comportamento clique em uma subclasse.

Para declarar o manipulador de eventos por meio de programação, criar um objeto `CompoundButton.OnCheckedChangeListener` e atribuí-lo ao botão chamando `setOnCheckedChangeListener` (`CompoundButton.OnCheckedChangeListener`). Por exemplo:

```
ToggleButton toggle = (ToggleButton)
findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

Snippet

Um Controle Spinners fornece uma maneira rápida para selecionar um valor a partir de um conjunto. No estado padrão, um spinner mostra seu valor selecionado. Tocando o spinner exibe um menu suspenso com todos os outros valores disponíveis, a partir do qual o usuário pode selecionar um novo.



Você pode adicionar um botão rotativo para o seu layout com o objeto Spinner. Normalmente deve fazê-lo em seu layout XML com um elemento <Spinner>. Por exemplo:

```
<Spinner
    android:id="@+id/planets_spinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
```

Para preencher o spinner com uma lista de opções, então você precisa especificar um SpinnerAdapter em sua atividade ou código fonte.

Preencher o Spinner com opções de usuário

As escolhas que você fornecer para o spinner pode vir de qualquer fonte, mas deve ser fornecida através de um SpinnerAdapter, como um

ArrayAdapter se as escolhas estão disponíveis em uma matriz ou um CursorAdapter se as escolhas estão disponíveis a partir de uma consulta de banco de dados.

Por exemplo, se as opções disponíveis para o seu Spinner são pré-determinados, pode fornecê-los com uma matriz de strings definida em um arquivo de recurso de string:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="planets_array">
        <item>Mercury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Jupiter</item>
        <item>Saturn</item>
        <item>Uranus</item>
        <item>Neptune</item>
    </string-array>
</resources>
```

Com uma série como esta, você pode usar o seguinte código em sua atividade ou fragmento de fornecer o spinner com a matriz usando uma instância de ArrayAdapter:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default
// spinner layout
ArrayAdapter<CharSequence> adapter =
    ArrayAdapter.createFromResource(this,
        R.array.planets_array, android.R.layout.simple_spinner_item);
// Specify the layout to use when the list of choices appears
```

```
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
// Apply the adapter to the spinner  
spinner.setAdapter(adapter);
```

O método `createFromResource()` permite que você crie um `ArrayAdapter` da matriz de strings. O terceiro argumento para este método é um recurso de layout que define como a opção selecionada aparece no controle spinner. O layout `simple_spinner_item` é fornecida pela plataforma e é o layout padrão, você deve usar a menos que você queira definir o seu próprio layout para a aparência do controle spinner.

Você deve então chamar `setDropDownViewResource(int)` para especificar o layout do adaptador deve usar para exibir a lista de opções do spinner (`simple_spinner_dropdown_item` é outro layout padrão definido pela plataforma).

Chame `SetAdapter()` para aplicar o adaptador ao Spinner.

Respondendo a seleções do usuário

Quando o usuário seleciona um item do menu suspenso, o objeto Spinner recebe um evento `on-item-selected`.

Para definir o manipulador de eventos de seleção para um spinner, implementar a interface `AdapterView.OnItemSelectedListener` e o método correspondente `onItemSelected()` de retorno. Por exemplo, aqui está uma implementação da interface em uma Atividade:

```
public class SpinnerActivity extends Activity implements  
    OnItemSelectedListener {  
    ...  
  
    public void onItemSelected(AdapterView<?> parent, View view,
```

```

        int pos, long id) {
        // An item was selected. You can retrieve the selected item
        using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}

```

O AdapterView.OnItemSelectedListener requer métodos de retorno a onItemSelected () e onNothingSelected ().

Então você precisa especificar a implementação de interface chamando setOnItemSelectedListener ():

```

Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);

```

Se você implementar a interface AdapterView.OnItemSelectedListener com sua atividade ou fragmento (como no exemplo acima), você pode passar isso como a instância da interface.

Eventos da interface

Eventos

No Android, há mais do que uma maneira de interceptar os eventos de interação do usuário com o aplicativo. Ao considerar eventos dentro de sua interface de usuário, a abordagem é capturar os eventos do objeto View específico que o usuário interage. A classe View fornece os meios para fazê-lo.

Dentro das várias classes de exibição que você vai usar para compor o seu layout, você pode observar vários métodos de retorno que parecem úteis para eventos da UI. Esses métodos são chamados pelo quadro Android quando a respectiva ação ocorre no objeto. Por exemplo, quando a View (como um botão) é tocado, o método `onTouchEvent()` é chamado no objeto. No entanto, a fim de interceptar isso, você deve estender a classe e substituir o método. No entanto, estendendo cada objeto View, a fim de lidar com um evento como esse não seria prático. É por isso que a classe View também contém uma coleção de interfaces com callbacks que você pode muito mais facilmente definir. Estas interfaces, chamadas de listeners de eventos, são o seu meio para capturar a interação do usuário com a interface do usuário.

Enquanto você vai usar mais comumente os listeners de eventos para verificar uma interação do usuário, pode chegar um momento em que você queira estender uma classe View, a fim de construir um componente personalizado. Talvez você queira estender a classe Button para fazer algo mais chique. Neste caso, você vai ser capaz de definir os comportamentos padrão de eventos para sua classe usando os manipuladores de eventos da classe.

Listeners de eventos

Um listener de evento é uma interface na classe View que contém um método de retorno de chamada única. Estes métodos serão chamados pela estrutura Android quando a View percebe que o listener foi registrado desencadeado pela interação do usuário com o item na interface do usuário.

Incluído nas interfaces, listeners de eventos são os métodos de retorno de chamada a seguir:

`onClick()`

De `View.OnClickListener`. Isso é chamado quando o usuário toca no item (quando em modo de toque), ou incide sobre o item com as teclas de

navegação ou trackball e pressiona a tecla "Enter" ou pressiona para baixo no trackball.

`onLongClick()`

De `View.OnLongClickListener`. Isso é chamado quando o usuário toca e mantém o item pressionado, ou incide sobre o item com as teclas de navegação ou trackball, pressiona e não solta a tecla "enter" ou pressiona e mantém pressionado o trackball (durante um segundo).

`onFocusChange()`

De `View.OnFocusChangeListener`. Isto é chamado quando o usuário navega para ou deixa um item, usando a navegação chave ou trackball.

`onKey()`

De `View.OnKeyListener`. Isso é chamado quando o usuário está focada no item e pressiona ou libera uma tecla de hardware do dispositivo.

`OnTouch()`

De `View.OnTouchListener`. Isso é chamado quando o usuário executa uma ação qualificada como um evento de toque, um release, ou em qualquer gesto de movimento na tela (dentro dos limites do item).

`onCreateContextMenu()`

De `View.OnCreateContextMenuListener`. Isto é chamado quando um menu de contexto está sendo construído (como o resultado de um "clique longo").

Estes métodos são os únicos que podem ser utilizados em sua interface. Para definir um destes métodos e lidar com seus eventos, implementar a interface em sua atividade ou defini-lo como uma classe anônima. Em seguida, passe uma instância de sua implementação para o método `View.set...Listener()` respectiva. (Por exemplo, chamar `setOnClickListener()` e passá-lo a sua implementação da `OnClickListener`.)

O exemplo abaixo mostra como registrar um listener no clique de um botão.

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
    ...
}
```

Você também pode achar mais conveniente para implementar OnClickListener como parte de sua atividade. Isso irá evitar a carga horária extra e alocação de objeto. Por exemplo:

```
public class ExampleActivity extends Activity implements OnClickListener
{
    protected void onCreate(Bundle savedInstanceState) {
        ...
        Button button = (Button)findViewById(R.id.corky);
        button.setOnClickListener(this);
    }

    // Implement the OnClickListener callback
    public void onClick(View v) {
        // do something when the button is clicked
    }
}
```

```
}  
...  
}
```

Observe que o retorno `onClick()` no exemplo acima não tem valor de retorno, mas alguns métodos listeners de outros eventos devem retornar um booleano. A razão é que depende do evento. Para os poucos que o fazem, aqui está o porquê:

`onLongClick()` - Retorna um booleano para indicar se você consumiu o evento e não deve ser levado adiante. Isto é, retorna `true` para indicar que você lidou com o caso e deve parar por aqui; retorna `false` se você não tiver lidado com isso e/ou o evento deve continuar a quaisquer outros cliques em listener.

`onKey()` - Retorna um booleano para indicar se você consumiu o evento e não deve ser levado adiante. Isto é, retorna `true` para indicar que você lidou com o caso e deve parar por aqui; retorna `false` se você não tiver lidado com isso e/ou o evento deve continuar a quaisquer outros listeners `onKey`.

`OnTouch()` - Retorna um booleano para indicar se o ouvinte consome este evento. O importante é que este evento pode ter várias ações que seguem um ao outro. Então, se você retornar `false` quando o evento é recebido, indica que você não tenha consumido o evento e também não estão interessados em ações subseqüentes deste evento. Assim, você não será chamado para outras ações dentro do evento, como um toque do dedo, ou o evento eventual ação qualquer.

Lembre-se que eventos de teclado são sempre entregues à View atualmente em foco. Eles são produzidos a partir do topo da hierarquia da view, seguindo até atingir o destino apropriado. Se a sua view (ou uma herança da sua View) atualmente tem o foco, então você pode ver o evento através do método `dispatchKeyEvent()`. Como uma alternativa para a captura de eventos

de teclado através de sua view, você também pode receber todos os eventos dentro de sua atividade com `onKeyDown()` e `onKeyUp()`.

Além disso, ao pensar sobre a entrada de texto para a sua aplicação, lembre-se que muitos dispositivos só têm métodos de entrada de software. Esses métodos não são obrigatoriamente baseados em teclados, alguns podem usar entrada de voz, escrita, e assim por diante. Mesmo se um método de entrada apresentar um teclado como interface, que geralmente não aciona a família `onKeyDown()` de eventos. Você nunca deve construir uma interface de usuário que requer teclas específicas para ser controlado, a menos que você queira limitar sua aplicação para dispositivos com um teclado de hardware. Em particular, não dependem destes métodos para validar a entrada quando o usuário pressiona a tecla de retorno, em vez disso, utilizam ações como `IME_ACTION_DONE` para sinalizar o método de entrada como o aplicativo espera para reagir, por isso pode mudar sua interface do usuário de uma forma significativa. Evitar suposições sobre como um método de entrada de software deve funcionar e apenas confiar nele para suprir o texto formatado para a sua aplicação.

Nota: Android vai chamar manipuladores de eventos primeiro e depois os manipuladores padrão apropriadas a partir do segundo definição de classe. Como tal, retornando `true` desses listeners de eventos vai parar a propagação do evento para outros listeners de evento e também irá bloquear o retorno de chamada para o manipulador de eventos padrão na View.

Manipuladores de Eventos

Se você está construindo um componente personalizado de view, então você vai ser capaz de definir vários métodos de retorno usados como manipuladores de eventos padrão. Agora, você vai aprender ver alguns dos callbacks comuns utilizados para manipulação de eventos:

`onKeyDown(int, KeyEvent)` - Chamado quando um novo evento de teclado ocorre.

`onKeyUp(int, KeyEvent)` - Chamado quando um evento de pressionar uma tecla ocorre.

`onTrackballEvent(MotionEvent)` - Chamado quando um evento de movimento trackball ocorre.

`onTouchEvent(MotionEvent)` - Chamado quando um evento de toque da tela ocorre.

`onFocusChanged(boolean, int, Rect)` - Chamado quando os elementos da view ganham ou perdem o foco.

Existem alguns outros métodos que você deve estar ciente de que não fazem parte da classe `View`, mas pode impactar diretamente a maneira que você é capaz de manipular eventos. Assim, quando o gerenciamento de eventos mais complexos dentro de um layout, considerar estes outros métodos:

`Activity.dispatchTouchEvent(MotionEvent)` - Isso permite que sua atividade intercepte todos os eventos de toque antes de serem enviados para a janela.

`ViewGroup.onInterceptTouchEvent(MotionEvent)` - Isso permite que um `ViewGroup` veja os eventos que ocorrem nas view herdadas.

`ViewParent.requestDisallowInterceptTouchEvent(boolean)` - Chamar em uma view para indicar que ela não deve interceptar eventos de toque com `onInterceptTouchEvent(MotionEvent)`.

Touch Model

Quando um usuário está navegando numa interface de usuário com as teclas direcionais ou um trackball, é necessário dar atenção a itens acionáveis (como botões) para que o usuário possa ver o que vai aceitar a entrada. Se o dispositivo possui recursos de toque, no entanto, o usuário

começa a interagir com a interface por tocá-lo, então ele não é mais necessário para realçar itens, ou dar foco a uma visão particular. Assim, existe um modo para a interação com o nome "touch model".

Para um dispositivo touch, uma vez que o usuário toca a tela, o aparelho entra em modo de toque. A partir desse ponto, exibições apenas para onde `isFocusableInTouchMode()` é verdade vai ser focado, como edição de texto.

Sempre que um usuário pressiona uma tecla direcional ou direciona com um trackball, o aparelho sairá do modo de toque, e encontra o elemento a fim de tomar o foco. Agora, o usuário poderá continuar interagindo com a interface do usuário sem tocar na tela.

O estado touch mode é mantido em todo o sistema (todas as janelas e atividades). Para consultar o estado atual, você pode chamar `isInTouchMode()` para ver se o dispositivo está em modo de toque.

Manuseio Foco

O quadro vai lidar com a rotina de movimento de foco, em resposta à entrada do usuário. Isso inclui a alteração do foco quando Views são removidos ou escondidos, ou quando novas Views se tornam disponíveis. Indicar que sua View esta disponível para assumir foco através do método `isFocusable()`. Para alterar se a View pode perder o foco, chamar `setFocusable()`. Quando em modo de toque, você pode consultar se uma View permite focar com `isFocusableInTouchMode()`. Você pode mudar isso com `setFocusableInTouchMode()`.

Movimento de foco é baseado em um algoritmo que encontra o vizinho mais próximo em uma determinada direção. Em casos raros, o algoritmo padrão pode não coincidir com o comportamento desejado do desenvolvedor. Nessas situações, você pode fornecer as substituições

explícitas com os seguintes atributos XML no arquivo de layout: `nextFocusDown`, `nextFocusLeft`, `nextFocusRight`, e `nextFocusUp`. Adicionar um desses atributos para a View que o foco está saindo. Definir o valor do atributo a ser a identificação de exibição que o foco deve ser dado. Por exemplo:

```
<LinearLayout
    android:orientation="vertical"
    ... >
    <Button android:id="@+id/top"
        android:nextFocusUp="@+id/bottom"
        ... />
    <Button android:id="@+id/bottom"
        android:nextFocusDown="@+id/top"
        ... />
</LinearLayout>
```

Normalmente, neste esquema vertical, navegando a partir do primeiro botão não iria a qualquer lugar, nem navegar para baixo do segundo botão. Agora que o botão de cima definiu a um como `nextFocusUp`(e vice-versa), o foco de navegação ciclico de cima para baixo e de baixo para cima.

Para solicitar uma view especial para ter foco, chamar `requestFocus()`.

Para eventos de foco (ser notificado quando a View recebe ou perde o foco), use `onFocusChange ()`.

Menus

Os menus são um componente comuns da interface de usuário em muitos tipos de aplicações. Para oferecer uma experiência familiar e consistente ao usuário, você deve usar as APIs do Menu para apresentar as ações do usuário e outras opções em suas atividades.

Embora o design e experiência do usuário para alguns itens de menu tenham sido alterados, a semântica para definir um conjunto de ações e de opções ainda é baseada nas APIs de Menu. Este guia mostra como criar os três tipos fundamentais de menus ou apresentações de ação em todas as versões do Android.

Menus de Opção e barras de ação

O menu de opções é a coleção de itens do menu principal para uma atividade. É onde você deve colocar as ações que têm um impacto global sobre o aplicativo, como "Pesquisa", "escrever e-mail," e "Configurações".

Se você está desenvolvendo para Android 2.3 ou inferior, os usuários podem visualizar o painel do menu de opções, pressionando o botão Menu.

No Android 3.0 e superior, os itens do menu de opções são apresentadas pela barra de ação como uma combinação de elementos na tela e opções de ação. A partir do Android 3.0, o botão Menu está obsoleta (alguns dispositivos nem tem um), assim que você deve migrar para usar a barra de ação para fornecer acesso a ações e outras opções.

Menu de contexto e modo de ação contextual

Um menu de contexto é um menu flutuante que aparece quando o usuário realiza um longo clique em um elemento. Ele fornece ações que afetam o conteúdo selecionado ou quadro de contexto.

Ao desenvolver para Android 3.0 e superior, você deve usar o modo de ação contextual para ativar ações sobre o conteúdo selecionado. Este modo exibe itens de ação que afetam o conteúdo selecionado em uma barra na parte superior da tela e permite que o usuário selecionar vários itens.

Menu pop-up

Um menu pop-up exibe uma lista de itens em uma lista vertical que está ancorada na view que invocou o menu. Ações em um menu pop-up não devem afetar diretamente o conteúdo, que é o que corresponde a ações contextuais. Em vez disso, o menu pop-up é para ações prolongadas que se relacionam com regiões de conteúdo em sua atividade.

Definição de um menu em XML

Para todos os tipos de menu, Android fornece um formato padrão XML para definir os itens do menu. Em vez de construir um menu de código de sua atividade, você deve definir um menu e todos os seus itens em um menu de recursos XML. Você pode então inserir o recurso de menu (carregá-lo como um objeto de Menu) em sua atividade.

Usar um recurso de menu é uma boa prática por algumas razões:

- É mais fácil de visualizar a estrutura do menu em XML.
- Ele separa o conteúdo para o menu de código de comportamento do seu aplicativo.
- Ele permite que você crie configurações alternativas do menu para as versões de plataformas diferentes, tamanhos de tela e outras configurações, aproveitando o quadro recursos da aplicação.

Para definir o menu, criar um arquivo XML dentro de `res/menu/` do seu projeto e construir o menu com os seguintes elementos:

`<menu>`

Define um Menu, que é um recipiente para itens de menu. Um elemento `<menu>` deve ser o nó raiz para o arquivo e pode conter um ou mais elementos `<item>` ou `<group>`.

<item>

Cria um MenuItem, o que representa um único item em um menu. Este elemento pode conter um elemento <menu> aninhado, a fim de criar um submenu.

<group>

Um recipiente, opcional invisível por elementos <item>. Ele permite que você categorizar os itens de menu para que eles compartilham propriedades, tais como estado ativo e visibilidade.

Aqui está um menu de exemplo chamado game_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

O elemento <item> suporta vários atributos que você pode usar para definir a aparência de um item e seu comportamento. Os itens no menu acima incluem os seguintes atributos:

android:id

A identificação do recurso que é único para o item, o que permite que o aplicativo seja capaz de reconhecer o item quando o usuário seleciona.

android:icon

Uma referência a um drawable para usar como ícone do item.

android:título

Uma referência a uma string para usar como título do item.

android:showAsAction

Especifica quando e como este item deve aparecer como um item na barra de ação.

Você pode adicionar um submenu a um item em qualquer menu (exceto um submenu), adicionando um elemento <menu> como o filho de um <item>. Submenus são úteis quando seu aplicativo tem um monte de funções que podem ser organizados em tópicos, como os itens na barra de uma aplicação para PC do menu (Arquivo, Editar, Exibir, etc.) Por exemplo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/file"
        android:title="@string/file" >
    <!-- "file" submenu -->
    <menu>
        <item android:id="@+id/create_new"
              android:title="@string/create_new" />
        <item android:id="@+id/open"
              android:title="@string/open" />
    </menu>
  </item>
</menu>
```

Para usar o menu em sua atividade, você precisa inserir um recurso de menu (converter o recurso XML em um objeto programável) usando `MenuInflater.inflate()`.

Criando um Menu de Opções

O menu de opções é onde você deve incluir ações e outras opções que são relevantes para o contexto da atividade atual, como "Pesquisar", "Editar e-mail," e "Configurações".



Figura 1. Opções de menu no navegador, no Android 2.3.

Onde os itens em seu menu de opções aparecerá na tela depende da versão para a qual você desenvolveu sua aplicação:

Se você desenvolveu o aplicativo para Android 2.3.x (API nível 10) ou inferior, o conteúdo do seu menu de opções aparecem na parte inferior da tela quando o usuário pressiona o botão do menu, como mostrado na figura 1. Quando aberto, a primeira parte visível é o ícone do menu, que tem capacidade para até seis itens de menu. Se o seu menu inclui mais de seis itens, Android coloca o item sexto e o resto para o menu de estouro, que o usuário pode abrir selecionando Mais.

Se você desenvolveu seu aplicativo para o Android 3.0 (API nível 11) e superior, os itens do menu de opções estão disponíveis na barra de ação. Por padrão, o sistema coloca todos os itens do estouro na barra de ação, que o usuário pode ver com o ícone de estouro de ação do lado direito da barra de ação (ou pressionando o botão de menu do dispositivo, se disponível). Para permitir o acesso rápido a ações importantes, você pode promover alguns itens para aparecer na barra de ação, adicionando `android:showAsAction = "ifRoom"` para os elementos `<item>` correspondentes (ver figura 2).

Nota: Mesmo se você não está desenvolvendo para o Android 3.0 ou superior, você pode construir seu próprio layout de barra de ação para ter um efeito semelhante.



Figura 2. Barra de ação a partir da aplicação Galeria Honeycomb, mostrando abas de navegação e um item de ação da câmera (mais o botão de estouro de ação).

Você pode declarar itens para o menu de opções a partir de qualquer subclasse da sua Atividade. Se tanto sua atividade e o(s) fragmento(s) declaram itens para o menu de opções, eles são combinados na interface do usuário. Itens da atividade aparecem em primeiro lugar, seguidos de cada item de fragmento na ordem em que cada fragmento é adicionado à atividade. Se necessário, você pode re-ordenar os itens de menu com o atributo `Android:orderInCategory` em cada `<item>` que você precisa mover.

Para especificar o menu de opções para uma atividade, substituir `onCreateOptionsMenu()`. Neste método, você pode inserir seu recurso de menu (definido em XML) para o Menu fornecida no retorno. Por exemplo:

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.menu.game_menu, menu);  
    return true;  
}
```

Você também pode adicionar itens de menu usando `add()` e recuperar os itens com `FindItem()` a rever as suas propriedades com `MenuItem` APIs.

Se você desenvolveu o aplicativo para Android 2.3.x e inferior, o sistema chama `onCreateOptionsMenu()` para criar o menu de opções quando o usuário abre o menu para a primeira vez. Se você desenvolveu para Android 3.0 e superior, o sistema chama `onCreateOptionsMenu()` quando iniciar a atividade, a fim de mostrar itens para a barra de ação.

Manipulação de eventos de clique

Quando o usuário seleciona um item do menu de opções (incluindo itens de ação na barra de ação), o sistema chama o método de sua atividade `onOptionsItemSelected()`. Este método passa o `MenuItem` selecionado. Você pode identificar o item chamando `getItemId()`, que retorna o ID único para o item de menu (definido pelo atributo `Android:id` no recurso de menu ou com um inteiro dado ao método `add()`). Você pode combinar esse ID com itens do menu conhecidos para executar a ação apropriada. Por exemplo:

@Override

```
public boolean onOptionsItemSelected(MenuItem item) {  
    // Handle item selection  
    switch (item.getItemId()) {  
        case R.id.new_game:  
            newGame();  
            return true;  
    }
```

```
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Quando você tratar um item de menu, retorne true. Se você não lidar com o item de menu, você deve chamar a implementação de superclasse de `onOptionsItemSelected()` (a implementação padrão retorna false).

Se a sua actividade inclui fragmentos, o sistema chama primeiro `onOptionsItemSelected()` para a actividade até um retorno verdadeiro.

Dica: Android 3.0 adiciona a capacidade para você definir o comportamento on-clique para um item de menu em XML, usando o atributo `Android:onClick`. O valor para o atributo deve ser o nome de um método definido pela actividade usando o menu. O método deve ser público e aceitar um parâmetro `MenuItem` único, quando o sistema chama este método, ele passa o item de menu selecionado.

Alterando os itens de menu em tempo de execução

Depois que o sistema chama `onCreateOptionsMenu()`, que mantém uma instância do `Menu`, você pode preenche-lo e não chamar `onCreateOptionsMenu()` novamente, a menos que o menu seja invalidado por algum motivo. No entanto, você deve usar `onCreateOptionsMenu()` somente para criar o estado inicial do menu e não fazer alterações durante o ciclo de vida da actividade.

Se você quiser modificar o menu de opções com base em eventos que ocorrem durante o ciclo de vida de actividade, você pode fazer isso no método `onPrepareOptionsMenu()`. Este método passa o objeto de `Menu` como

ele existe atualmente para que você possa modificá-lo, tal como adicionar, remover ou desabilitar itens.

No Android 2.3.x e inferior, o sistema chama `onPrepareOptionsMenu()` cada vez que o usuário abre o menu de opções (aperta o botão Menu).

No Android 3.0 e superior, o menu de opções é considerado sempre estar aberto quando os itens de menu são apresentados na barra de ação. Quando ocorre um evento e pretende realizar uma atualização de menu, você deve chamar `invalidateOptionsMenu()` para solicitar a chamada de `onPrepareOptionsMenu()`.

Nota: Você nunca deve alterar itens no menu de opções com base na view atualmente em foco. Quando em modo de toque (quando o usuário não estiver usando um trackball ou d-pad), pontos da view não podem perder o foco, então você nunca deve usar o foco como base para a modificação de itens no menu de opções. Se você deseja fornecer itens de menu que são sensíveis ao contexto para uma view, use um menu de contexto.

Criação de menus de contexto

Um menu de contexto oferece ações que afetam um item específico ou quadro de contexto na interface do usuário. Você pode fornecer um menu de contexto para qualquer ponto da view, mas eles são mais frequentemente utilizados para itens em um `ListView`, `GridView` ou coleções de outras view em que o usuário pode executar ações diretas sobre cada item.

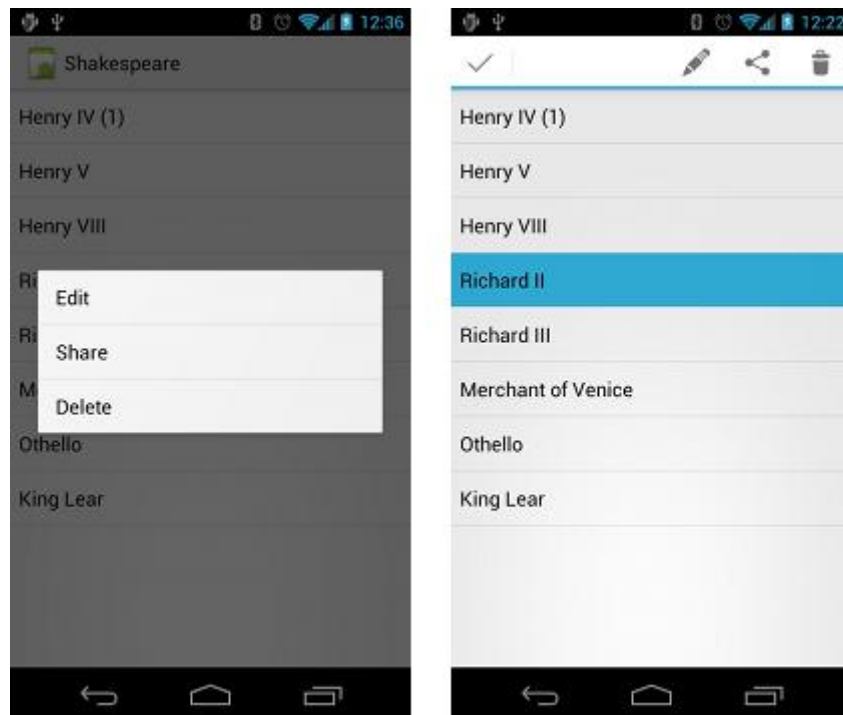


Figura 3. Imagens de um menu de contexto flutuante (à esquerda) e da barra de ação contextual (direita).

Há duas maneiras de fornecer ações contextuais:

No menu de contexto flutuante. Um menu que aparece como uma lista de itens de menu flutuante (semelhante a uma caixa de diálogo) quando o usuário executa um longo clique (pressione e segure) em uma exibição que declara apoio a um menu de contexto. Os usuários podem executar uma ação contextual sobre um item de cada vez.

No modo de ação contextual. Este modo é uma implementação do sistema de ActionMode que exibe uma barra de ação contextual no topo da tela com itens de ação que afetam o item selecionado. Quando este modo está activo, os usuários podem executar uma ação em vários itens de uma só vez (se o seu aplicativo permite que ele).

Nota: O modo de ação contextual está disponível no Android 3.0 (API nível 11) e superiores e é a técnica preferida para a exibição de ações

contextuais, quando disponível. Se o seu aplicativo suporta versões menores do que 3,0, então você deve cair de volta para um menu de contexto flutuante nesses dispositivos.

Criando um menu de contexto flutuante

Para fornecer um menu de contexto flutuante:

Registre para que view o menu de contexto deve ser associado chamando `registerForContextMenu()` e passe-o a view.

Se a sua atividade usa um `ListView` ou `GridView` e você quer que cada item forneça o mesmo menu de contexto, registre todos os itens de um menu de contexto, passando a `ListView` ou `GridView` para `registerForContextMenu()`.

Implementar o `onCreateContextMenu()` em sua atividade.

Quando a view registrada recebe um evento de longa-clique, o sistema chama o seu método `onCreateContextMenu()`. Este é onde você define os itens do menu, geralmente inserindo um recurso de menu. Por exemplo:

`@Override`

```
public void onCreateContextMenu(ContextMenu menu, View v,
                               ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

`MenuInflater` permite inserir o menu de contexto de um recurso de menu. Os parâmetros do método de retorno de chamada incluem a view selecionada pelo usuário e um objeto `ContextMenu.ContextMenuInfo` que fornece informações adicionais sobre o item selecionado. Se a sua actividade

tem várias exibições onde cada fornece um menu de contexto diferente, você pode usar esses parâmetros para determinar qual o menu de contexto para inserir.

Implementar `onContextItemSelected()`.

Quando o usuário seleciona um item de menu, o sistema chama este método para que você possa executar a ação apropriada. Por exemplo:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo)
item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

O método `getItemId()` consulta o ID do item de menu selecionado, que você deve atribuir a cada item de menu em XML usando o atributo `android:id`, como mostrado na seção sobre definição de um menu em XML.

Quando você tratar um item de menu, retorne `true`. Se você não lidar com o item de menu, você deve passar o item de menu para a implementação de superclasse. Se a sua actividade inclui fragmentos, a actividade recebe o callback primeiro. Chamando a superclasse quando tratada, o sistema passa para o evento de o método de retorno respectivo em cada fragmento, um de

cada vez (segundo a ordem de cada fragmento foi adicionado) até verdadeiro ou falso ser retornado. (A implementação padrão para `Atividade` e `android.app.Fragment` retornar falso, assim que você deve sempre chamar a superclasse quando não tratada.)

Usando o modo de ação contextual

O modo de ação contextual é uma implementação do sistema de `ActionMode` que se concentra interação do usuário para realizar ações contextuais. Quando um usuário habilita este modo, selecionando um item, uma barra de ação contextual aparece na parte superior da tela para apresentar ações que o usuário pode executar no item selecionado. Enquanto este modo é ativado, o usuário pode selecionar vários itens (se você permitir), desmarcar itens, e continuar a navegar dentro da atividade (tanto quanto você está disposto a permitir). O modo de ação é desativado e barra de ação contextual desaparece quando o usuário cancela a seleção de todos os itens, pressiona o botão de volta, ou seleciona a ação feita no lado esquerdo da barra.

Nota: A barra de ação contextual não está necessariamente associada com a barra de ação. Eles operam de forma independente, mesmo que a barra de ação contextual visualmente ultrapassa a posição da barra de ação.

Se você está desenvolvendo para o Android 3.0 (API nível 11) ou superior, você geralmente deve usar o modo de ação contextual para apresentar ações contextuais, ao invés de o menu de contexto flutuante.

Para views que oferecem ações contextuais, você geralmente deve invocar o modo de ação contextual sobre um dos dois eventos (ou ambos):

- O usuário executa um clique longo sobre a view.
- O usuário seleciona uma opção ou componente de interface semelhante dentro da view.

Como seu aplicativo chama o modo de ação contextual e define o comportamento de cada ação depende do seu design. Existem basicamente dois modelos:

- Para as ações individuais, contextuais em view arbitrários.
- Para as ações em lote contextuais em grupos de itens em um ListView ou GridView (permitindo que o usuário selecione vários itens e executar uma ação em todos eles).

Ativando o modo de ação contextual com opiniões individuais

Se você quiser chamar o modo de ação contextual apenas quando o usuário seleciona views específicas, você deve:

Implementar na interface `ActionMode.Callback`. Em seus métodos de retorno de chamada, você pode especificar as ações para a barra de ação contextual, responder a eventos click em itens de ação, e lidar com estes eventos.

Chame `startActionMode()` quando você quiser mostrar a barra (como quando o usuário de longa cliques na view).

Por exemplo:

Implementar a interface `ActionMode.Callback`:

```
private ActionMode.Callback mActionModeCallback = new
ActionMode.Callback() {

    // Called when the action mode is created; startActionMode() was called
    @Override
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Inflate a menu resource providing context menu items
        MenuInflater inflater = mode.getMenuInflater();
```

```

        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    // Called each time the action mode is shown. Always called after
    onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        return false; // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    @Override
    public boolean onOptionsItemSelected(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_share:
                shareCurrentItem();
                mode.finish(); // Action picked, so close the CAB
                return true;
            default:
                return false;
        }
    }

    // Called when the user exits the action mode
    @Override
    public void onDestroyActionMode(ActionMode mode) {
        mActionMode = null;
    }
};

```

Observe que esses retornos de eventos são quase exatamente o mesmo que os retornos de chamada para o menu de opções, exceto que cada

um destes também passa o objeto `ActionMode` associado ao evento. Você pode usar `ActionMode` APIs para fazer várias mudanças para o CAB, como revisar o título e o subtítulo com `setTitle()` e `setSubtitle()` (útil para indicar quantos itens são selecionados).

Note também que o exemplo acima define como nulo a variável `mActionMode` quando o modo de ação é destruído. Na próxima etapa, você vai ver como ele é inicializado e como salvar a variável de membro em sua atividade.

Chame `startActionMode()` para ativar o modo de ação contextual quando apropriado, como em resposta a um longo clique em uma view:

```
someView.setOnLongClickListener(new View.OnLongClickListener() {  
    // Called when the user long-clicks on someView  
    public boolean onLongClick(View view) {  
        if (mActionMode != null) {  
            return false;  
        }  
  
        // Start the CAB using the ActionMode.Callback defined above  
        mActionMode = getActivity().startActionMode(mActionModeCallback);  
        view.setSelected(true);  
        return true;  
    }  
});
```

Quando você chama `startActionMode()`, o sistema retorna o `ActionMode` criado. Ao salvar isso em uma variável de membro, você pode fazer alterações na barra de ação contextual em resposta a outros eventos. No exemplo acima, o `ActionMode` é usado para garantir que a instância `ActionMode` não será recriada se ela já está ativa, verificando se o membro é nulo antes de iniciar o modo de ação.

Permitindo ações em lote contextuais em um ListView ou GridView

Se você tem uma coleção de itens em um ListView ou GridView (ou outra extensão de AbsListView) e deseja permitir aos usuários executar ações em lote, você deve:

Implementar a interface AbsListView.MultiChoiceModeListener e configurá-lo para o groupview com setMultiChoiceModeListener(). Em métodos de retorno de chamada de listeners, você pode especificar as ações para a barra de ação contextual responder a eventos click sobre itens de ação, e lidar com callbacks.

Chame setChoiceMode() com o argumento CHOICE_MODE_MULTIPLE_MODAL.

Por exemplo:

```
ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(new MultiChoiceModeListener() {

    @Override
    public void onItemCheckedStateChanged(ActionMode mode, int position,
                                         long id, boolean checked) {
        // Here you can do something when items are selected/de-selected,
        // such as update the title in the CAB
    }

    @Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        // Respond to clicks on the actions in the CAB
        switch (item.getItemId()) {
            case R.id.menu_delete:
                deleteSelectedItems();
                return true;
            default:
                return false;
        }
    }
});
```

```

        mode.finish(); // Action picked, so close the CAB
        return true;
    default:
        return false;
    }
}

@Override
public boolean onCreateActionMode(ActionMode mode, Menu menu) {
    // Inflate the menu for the CAB
    MenuInflater inflater = mode.getMenuInflater();
    inflater.inflate(R.menu.context, menu);
    return true;
}

@Override
public void onDestroyActionMode(ActionMode mode) {
    // Here you can make any necessary updates to the activity when
    // the CAB is removed. By default, selected items are
deselected/unchecked.
}

@Override
public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
    // Here you can perform updates to the CAB due to
    // an invalidate() request
    return false;
}
});

```

É isso aí. Agora, quando o usuário seleciona um item com um clique longo, o sistema chama o método `onCreateActionMode()` e exibe a barra de ação contextual com as ações especificadas. Enquanto a barra de ação contextual é visível, os usuários podem selecionar itens adicionais.

Em alguns casos em que as ações contextuais fornecem itens de ação comuns, você pode querer adicionar uma caixa de seleção ou um elemento de interface do usuário similar que permite aos usuários selecionar itens, porque não pode descobrir o comportamento de longo-clique. Quando um usuário seleciona a caixa de seleção, você pode invocar o modo de ação contextual, definindo o respectivo item da lista para o estado verificado com `SetItemChecked()`.

Criando um menu pop-up

Um `PopupMenu` é um menu modal ancorado a uma `View`. Ele aparece na parte inferior da view se houver espaço, ou caso contrario acima da view. É útil para:

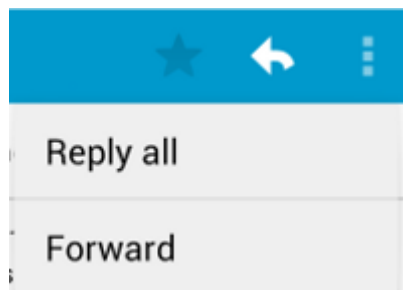


Figura 4. Um menu pop-up no app do Gmail, ancorado no botão de estouro na parte superior direita.

Fornecer um menu estouro estilo para ações que se relacionam com o conteúdo específico (como cabeçalhos de e-mail do Gmail, mostrados na figura 4).

Fornecer uma segunda parte de uma frase de comando (como um botão "Adicionar", que produz um menu com diferentes opções "Adicionar").

Fornecendo um suspenso semelhante ao `Spinner` que não retém uma seleção persistente.

Nota: `PopupMenu` está disponível com a API nível 11 e superior.

Se você definir o seu menu em XML, é aqui como você pode mostrar o menu pop-up:

Instantate um PopupMenu com seu construtor, que leva o contexto atual do aplicativo e da View onde o menu deva ser ancorado.

Use MenuInflater inserir seu recurso de menu no Menu objeto retornado por PopupMenu.getMenu(). Na API nível 14 e acima, você pode usar PopupMenu.inflate() em seu lugar.

Chame PopupMenu.show().

Por exemplo, aqui está um botão com o atributo android:onClick que mostra um menu pop-up:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

A atividade pode, então, mostrar o menu pop-up como este:

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

Na API nível 14 ou superior, você pode combinar as duas linhas que inserem o menu com PopupMenu.inflate().

O menu é fechado quando o usuário seleciona um item ou toca fora da área do menu. Você pode fechá-lo usando o evento `PopupMenu.OnDismissListener`.

Manipulação de eventos de clique

Para executar uma ação quando o usuário seleciona um item de menu, você deve implementar a interface `PopupMenu.OnMenuItemClickListener` e registrá-lo com o seu `PopupMenu` chamando `setOnMenuItemClickListener()`. Quando o usuário seleciona um item, o sistema chama `onMenuItemClick()` de retorno de chamada em sua interface.

Por exemplo:

```
public void showMenu(View v) {  
    PopupMenu popup = new PopupMenu(this, v);  
  
    // This activity implements OnMenuItemClickListener  
    popup.setOnMenuItemClickListener(this);  
    popup.inflate(R.menu.actions);  
    popup.show();  
}
```

`@Override`

```
public boolean onMenuItemClick(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.archive:  
            archive(item);  
            return true;  
        case R.id.delete:  
            delete(item);  
            return true;  
        default:
```

```
        return false;
    }
}
```

Criação de Grupos de menus

Um grupo de menus é uma coleção de itens de menu que compartilham certas características. Com um grupo, você pode:

- Mostrar ou ocultar todos os itens com `setGroupVisible()`
- Ativar ou desativar todos os itens com `setGroupEnabled()`
- Especificar se todos os itens são verificáveis com `setGroupCheckable()`

Você pode criar um grupo de elementos `<item>` alocados dentro de um elemento `<group>` em seu recurso de menu ou especificando um ID de grupo com o método `add()`.

Aqui está um recurso de menu exemplo que inclui um grupo:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/menu_save"
        android:icon="@drawable/menu_save"
        android:title="@string/menu_save" />
    <!-- menu group -->
    <group android:id="@+id/group_delete">
        <item android:id="@+id/menu_archive"
            android:title="@string/menu_archive" />
        <item android:id="@+id/menu_delete"
            android:title="@string/menu_delete" />
    </group>
</menu>
```

Os itens que estão no grupo aparecem no mesmo nível que o primeiro item de todos os três itens no menu. No entanto, você pode modificar as características dos dois itens do grupo, fazendo referência a ID de grupo e utilizando os métodos listados acima. O sistema também nunca separa itens agrupados. Por exemplo, se você declarar `android: showAsAction = "ifRoom"` para cada item, eles vão quer tanto aparecer na barra de ação ou ambos aparecem no estouro de ação.

Usando itens chacaveis em menus

Um menu pode ser útil como uma interface para ligar e desligar opções, usando uma caixa de seleção para opções autônomas, ou botões de rádio para grupos de opções mutuamente exclusivas. A Figura 5 mostra um submenu com itens que são verificáveis com botões de rádio.

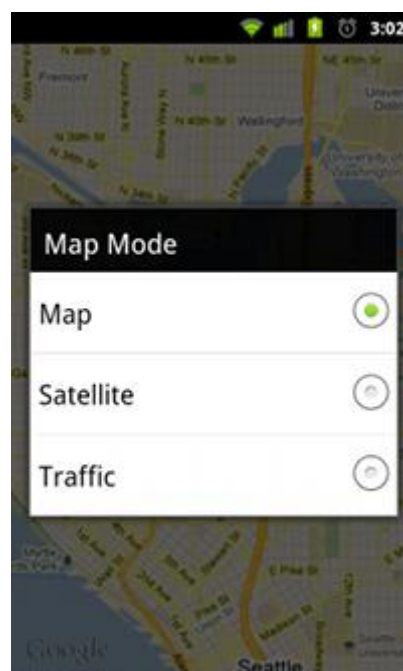


Figura 5. Screenshot de um submenu com itens checkable.

Nota: Os itens de menu no menu de ícone (no menu de opções) não pode exibir uma caixa de seleção ou botão de rádio. Se você optar por fazer itens na checkable com ícones, você deve manualmente indicar o estado

marcado para trocar o ícone e/ou o texto cada vez que houver mudanças de estado.

Você pode definir o comportamento verificável para itens de menu individuais usando o atributo `android:checkable` no elemento `<item>`, ou para um grupo inteiro com o atributo `android:checkableBehavior` no elemento `<group>`. Por exemplo, todos os itens deste grupo de menu são verificáveis com um botão de rádio:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
  <group android:checkableBehavior="single">
    <item android:id="@+id/red"
          android:title="@string/red" />
    <item android:id="@+id/blue"
          android:title="@string/blue" />
  </group>
</menu>
```

O atributo `android:checkableBehavior` aceita:

- `single` - Apenas um item do grupo pode ser verificado (botões de opção)
- `all` - Todos os itens podem ser verificados (caixas)
- `none` - Não há itens são verificáveis

Você pode aplicar um padrão verificado estado de um item usando o atributo `android:checkable` no elemento `<item>` e alterá-lo no código com o método `setChecked()`.

Quando um item verificável é selecionado, o sistema chama o método de retorno respectivo do item selecionado (como `onOptionsItemSelected()`). É aqui que você deve definir o estado da caixa de

verificação, porque um botão caixa ou rádio não muda seu estado automaticamente. Você pode consultar o estado atual do item (como era antes de o usuário selecionado) com `isChecked()` e, em seguida, definir o estado verificado com `setChecked()`. Por exemplo:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.vibrate:
        case R.id.dont_vibrate:
            if (item.isChecked()) item.setChecked(false);
            else item.setChecked(true);
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Se você não definir o estado verificado desta maneira, então o estado visível do item (a caixa de seleção ou botão de rádio) não vai mudar quando o usuário a seleciona. Quando você definir o estado, a atividade preserva o estado selecionado do item para que quando o usuário abra o menu mais tarde, o estado verificado que defido seja visível.

Nota: Os itens de menu checkable se destinam a serem usados apenas em uma base por sessão e não salva depois que o aplicativo é destruído. Se você tiver as configurações do aplicativo que você gostaria de guardar para o usuário, você deve armazenar os dados usando Preferências compartilhadas.

Adicionando itens de menu com base em uma Intent

Às vezes, você vai querer um item de menu para iniciar uma atividade usando uma intent. Quando você conhece a intent que deseja usar e ter um menu específico que deve iniciar a intent, você pode executar a intent com `startActivity()` durante o método de retorno adequado no item selecionado (como o `onOptionsItemSelected()`).

No entanto, se você não tem certeza que o dispositivo do usuário que contém um aplicativo lide com a intent, depois de adicionar uma chamada a um item de menu pode resultar em um item de menu que não funciona, porque a intent não pode resolver a informação para uma atividade. Para resolver isso, o Android permite a você adicionar dinamicamente itens de menu para o menu quando Android encontra atividades no dispositivo que precisa lidar com a sua intent.

Para adicionar itens de menu com base nas atividades disponíveis que aceitam uma intent:

- Definir uma intent com categoria `CATEGORY_ALTERNATIVE` e/ou `CATEGORY_SELECTED_ALTERNATIVE`, além de quaisquer outros requisitos.
- Chame `Menu.addIntentOptions()`. O android em seguida, procura todas as aplicações que podem ser executadas pela intent e os adiciona ao seu menu.

Se não houver aplicativos instalados que satisfaçam a intent, nenhum item de menu sera adicionado.

Nota: `CATEGORY_SELECTED_ALTERNATIVE` é usado para manipular o elemento selecionado na tela. Assim, ele só deve ser usado quando a criação de um menu em `onCreateContextMenu ()`.

Por exemplo:

@Override

```
public boolean onCreateOptionsMenu(Menu menu){
    super.onCreateOptionsMenu(menu);

    // Create an Intent that describes the requirements to fulfill, to be included
    // in our menu. The offering app must include a category value of
    Intent.CATEGORY_ALTERNATIVE.
    Intent intent = new Intent(null, dataUri);
    intent.addCategory(Intent.CATEGORY_ALTERNATIVE);

    // Search and populate the menu with acceptable offering applications.
    menu.addIntentOptions(
        R.id.intent_group, // Menu group to which new items will be added
        0, // Unique item ID (none)
        0, // Order for the items (none)
        this.getComponentName(), // The current activity name
        null, // Specific items to place first (none)
        intent, // Intent created above that describes our requirements
        0, // Additional flags to control items (none)
        null); // Array of MenuItem's that correlate to specific items (none)

    return true;
}
```

Para cada atividade que fornece um filtro de intent de correspondência a intent definida, um item de menu é adicionado, utilizando o valor no android a intent do filtro: rótulo, como o título do item de menu e o ícone do aplicativo como o ícone de item de menu. O addIntentOptions() retorna o número de itens de menu acrescentou.

Nota: Quando você chama `addIntentOptions()`, ela substitui qualquer e todos os itens de menu por um grupo de menus especificado no primeiro argumento.

Permitindo que a sua atividade possa ser adicionado a outros menus

Você também pode oferecer os serviços de sua atividade para outras aplicações, para que a sua aplicação possa ser incluído no menu de outros (inverter os papéis descritos acima).

Para ser incluído no menus de outras aplicações, você precisa definir um filtro de intent, como de costume, mas não se esqueça de incluir os valores `CATEGORY_ALTERNATIVE` e/ou `CATEGORY_SELECTED_ALTERNATIVE` para a categoria do filtro de intent. Por exemplo:

```
<intent-filter label="@string/resize_image">
    ...
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category
android:name="android.intent.category.SELECTED_ALTERNATIVE" />
    ...
</intent-filter>
```

Notificações

A notificação é uma mensagem que você pode exibir para o usuário fora da UI normal do seu aplicativo. Quando você diz que o sistema emitirá uma notificação, ele aparece pela primeira vez como um ícone na área de notificação. Para ver os detalhes da notificação, o usuário abre a gaveta de notificação. Tanto a área de notificação e na gaveta de notificação são o sistema de áreas controladas que o usuário pode ver a qualquer momento.



Figura 1. Notificações na área de notificação.



Figura 2. Notificações na gaveta notificação.

Design da notificação

Notificações, como uma parte importante da interface do usuário Android, têm as suas próprias diretrizes de design.

Exibir elementos de notificação

Notificações na gaveta notificação pode aparecer em um dos dois estilos visuais, dependendo da versão e do estado da gaveta:

Visualização normal

Uma notificação em modo normal aparece em uma área que é de até 64 dp de altura. Mesmo se você criar uma notificação com um estilo de visualização grande, ela aparecerá na view normal até que seja expandido. Este é um exemplo de uma vista normal:

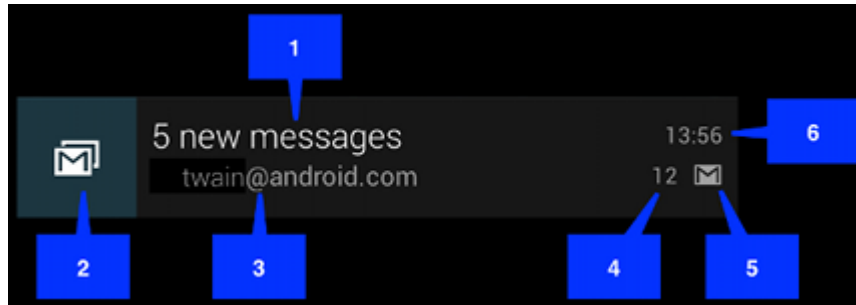


Figura 3. Notificação na view normal.

As chamadas na ilustração referem-se ao seguinte:

- 1- Título de conteúdo
- 2- Ícone grande
- 3- Conteúdo de texto
- 4- Informações conteúdo
- 5- Pequeno ícone

- 6- Tempo em que a notificação foi emitida. Você pode definir um valor explícito com `setWhen()`; se você não fizer isso o padrão é o tempo que o sistema recebeu a notificação.

Visão ampliada

Visão ampliada é uma notificação aparece apenas quando a notificação for ampliada, o que acontece quando a notificação está no topo da gaveta de notificações ou quando o usuário expande a notificação com um gesto. Notificações expandidos estão disponíveis a partir com o Android 4.1.

A imagem seguinte mostra uma notificação caixa de entrada de estilo:

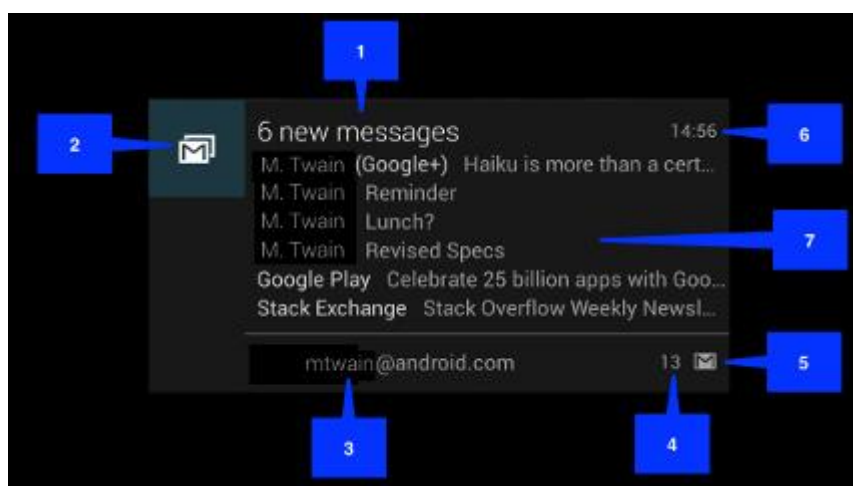


Figura 4. Notificação vista grande.

Observe que na maior parte da vista grande de seus elementos visuais são com na visão normal. A única diferença é chamada número 7, a área de detalhes. Cada estilo Visão ampliada define esta área de uma forma diferente. Os estilos disponíveis são:

Estilo de imagem grande

A área de detalhes contém um bitmap de até 256 dp em sua seção de detalhes.

Estilo de texto grande

Exibe um bloco de texto grande na seção de detalhes.

Estilo de caixa de entrada

Apresenta linhas de texto na seção de detalhes.

Todos os estilos de vista grandes também têm as seguintes opções de conteúdo que não estão disponíveis no modo normal:

Título de conteúdo grande

Permite substituir o título da vista normal de conteúdo com um título que aparece apenas na visualização expandida.

Texto de resumo

Permite adicionar uma linha de texto abaixo da área de detalhes.

Criando uma Notificação

Você especifica as informações de interface do usuário e as ações de uma notificação em um objeto `NotificationCompat.Builder`. Para criar a própria notificação, você chama `NotificationCompat.Builder.build()`, que retorna um objeto de notificação com suas especificações. Para emitir a notificação, você passa o objeto de notificação ao sistema chamando `NotificationManager.notify()`.

Conteúdo Necessários de uma notificação

Um objeto de notificação deve conter o seguinte:

- Um pequeno ícone, definida pelo `setSmallIcon()`
- Um título, definida pelo `setContentTitle()`
- Texto de detalhe, definida pelo `setContentText()`

Ações da notificação

Embora eles sejam opcionais, você deve adicionar pelo menos uma ação à sua notificação. Uma ação permite aos usuários ir diretamente para uma atividade a partir da notificação em sua aplicação, onde eles podem olhar para um ou mais eventos ou trabalhar mais.

Uma notificação pode fornecer várias ações. Você sempre deve definir a ação que é acionada quando o usuário clica a notificação, geralmente esta ação abre uma atividade em sua aplicação. Você também pode adicionar botões para a notificação para realizar ações adicionais, tais como soar um alarme ou responder imediatamente a uma mensagem de texto, este recurso está disponível como do Android 4.1. Se você usar os botões de ação adicional, você também deve fazer a sua funcionalidade disponibilizar uma atividade em seu aplicativo.

Dentro de uma notificação, a ação é definida por um PendingIntent contendo uma Intent que começa uma atividade na sua aplicação. Para associar o PendingIntent com um gesto, chamar o método apropriado de NotificationCompat.Builder. Por exemplo, se você quer começar a atividade, quando o usuário clica no texto da notificação na gaveta de notificação, você deve adicionar o PendingIntent chamando setContentIntent().

Iniciar uma Atividade quando o usuário clica a notificação é o cenário de ação mais comum. Você também pode iniciar uma atividade quando o usuário descarta uma Atividade. No Android 4.1 e posterior, você pode começar uma atividade a partir de um botão de ação.

Criando uma notificação simples

O trecho a seguir ilustra uma simples notificação que especifica uma atividade para abrir quando o usuário clica a notificação. Observe que o código cria um objeto TaskStackBuilder e usa-o para criar o PendingIntent para a ação.

```

NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");
// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack for the Intent (but not the Intent itself)
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(
        0,
        PendingIntent.FLAG_UPDATE_CURRENT
    );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
// mId allows you to update the notification later on.
mNotificationManager.notify(mId, mBuilder.build());

```

É isso aí. Seu usuário já foi notificado.

Aplicando um estilo de visualização grande para uma notificação

Para se ter uma notificação aparecem em uma visão grande quando é expandido, primeiro criar um objeto `NotificationCompat.Builder` com as opções de visualização normais que você deseja. Em seguida, chame `Builder.setStyle ()` com um objeto grande visão de estilo como seu argumento.

Lembre-se de que as notificações de expansão não estão disponíveis em plataformas anteriores ao Android 4.1.

Por exemplo, o trecho de código a seguir demonstra como alterar a notificação criado no trecho anterior para usar a caixa de entrada estilo de visualização grande:

```
NotificationCompat.Builder mBuilder = new
NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Event tracker")
    .setContentText("Events received")
NotificationCompat.InboxStyle inboxStyle =
    new NotificationCompat.InboxStyle();
String[] events = new String[6];
// Sets a title for the Inbox style big view
inboxStyle.setBigContentTitle("Event tracker details:");
...
// Moves events into the big view
for (int i=0; i < events.length; i++) {

    inboxStyle.addLine(events[i]);
}
// Moves the big view style object into the notification object.
mBuilder.setStyle(inBoxStyle);
...
// Issue the notification here.
```

Manipulação de compatibilidade

Nem todos os recursos de notificação estão disponíveis para uma versão específica, mesmo que os métodos para defini-as estejam na classe da biblioteca de apoio `NotificationCompat.Builder`. Por exemplo, os botões de ação, que dependem de notificações expandidas, só aparecem no Android 4.1 e superior, porque as notificações de tamanho grande só estão disponíveis no Android 4.1 e superior.

Para garantir a melhor compatibilidade, na criação de notificações com `NotificationCompat` e suas subclasses, particularmente `NotificationCompat.Builder`. Além disso, siga este processo quando você implementar uma notificação:

Fornecer todas as funcionalidades de uma notificação a todos os usuários, independentemente da versão que está usando. Para fazer isso, verifique se todas as funcionalidades estão disponíveis a partir de uma atividade em sua aplicação. Você pode querer adicionar uma nova atividade para fazer isso.

Por exemplo, se você quiser usar `Addaction()` para fornecer um controle que pára e começa a reprodução de mídia, primeiro implemente este controle em uma atividade em sua aplicação.

Garantir que todos os usuários podem obter a funcionalidade na atividade, por tê-lo iniciado quando os usuários clicam a notificação. Para fazer isso, crie um `PendingIntent` para a atividade. Chame `setContentIntent()` para adicionar o `PendingIntent` à notificação.

Agora adicione os recursos de notificação expandidas você deseja usar para a notificação. Lembre-se que qualquer funcionalidade que você adicionar também tem de estar disponível na Atividade que se inicia quando o usuário clicar na notificação.

Gerenciando Notificações

Quando você precisa emitir uma notificação várias vezes para o mesmo tipo de evento, você deve evitar fazer uma notificação completamente nova. Em vez disso, você deve considerar a atualização da notificação anterior, alterando alguns dos seus valores ou adicionando a ele, ou ambos.

Por exemplo, o Gmail notifica o usuário que novos e-mails chegaram ao aumentar sua contagem de mensagens não lidas e adicionando um resumo de cada e-mail com a notificação. Isso é chamado de "empilhamento" a notificação.

Nota: Esta funcionalidade do Gmail exige a "caixa de entrada" estilo de visualização grande, que é parte do recurso de notificação expandida disponível a partir de Android 4.1.

Atualizando Notificações

Para configurar uma notificação para que ele possa ser atualizada, usaremos o ID da notificação chamando `NotificationManager.notify(ID, notification)`. Para atualizar esta notificação uma vez que você a emitiu, atualizar ou criar um objeto `NotificationCompat.Builder`, construir um objeto de notificação a partir dele, e emitir a notificação com o mesmo ID que você usou anteriormente. Se a notificação anterior ainda é visível, o sistema atualiza a partir do conteúdo do objeto de notificação. Se a notificação anterior tiver sido concluída, uma nova notificação é criada em seu lugar.

O trecho a seguir demonstra uma notificação de que é atualizada para refletir o número de eventos que ocorreram. Empilha a notificação, mostrando um resumo:

```
mNotificationManager =  
    (NotificationManager)  
    getSystemService(Context.NOTIFICATION_SERVICE);
```

```

// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
mNotifyBuilder.setContentText(currentText)
    .setNumber(++numMessages);
// Because the ID remains unchanged, the existing notification is
// updated.
mNotificationManager.notify(
    notifyID,
    mNotifyBuilder.build());
...

```

Isso produz uma notificação de que se parece com isso:

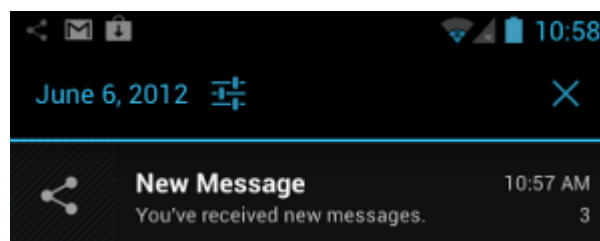


Figura 5. Notificação actualizada exibido na gaveta de notificação.

Removendo notificações

Notificações permanecem visíveis até que uma das seguintes situações:

- O usuário descarta a notificação seja individualmente ou através de "Clear All" (se a notificação pode ser desmarcado).

- O usuário clica na notificação, e chamou `setAutoCancel()` quando criou a notificação.
- Você chama `cancelar()` para um ID de notificação específico. Este método também elimina notificações em curso.
- Você chama `cancelAll()`, que remove todas as notificações que anteriormente emitidos.

Preservando a Navegação quando iniciar uma atividade

Quando você começar uma atividade de uma notificação, você deve preservar a experiência da navegação do usuário. Clicando Voltar deve levar o usuário de volta através do fluxo de trabalho da aplicação para a tela inicial, e clicando recentes deve mostrar a atividade como uma tarefa separada. Para preservar a experiência da navegação, você deve iniciar a atividade em uma nova tarefa. Como configurar o `PendingIntent` para dar uma nova tarefa depende da natureza da atividade que você está começando. Há duas situações gerais:

A atividade regular

Você está começando uma atividade que faz parte do fluxo de trabalho normal da aplicação. Nesta situação, configurar o `PendingIntent` para iniciar uma nova tarefa, e fornecer o `PendingIntent` com uma pilha de retorno, que reproduz o comportamento voltar do aplicativo normal.

Notificações do aplicativo do Gmail demonstram isso. Ao clicar em uma notificação de uma mensagem de e-mail único, você verá a mensagem em si. Tocar no voltar leva você para trás através do Gmail para a tela inicial, como se você tivesse entrado no Gmail a partir da tela inicial, em vez de digitá-la a partir de uma notificação.

Isso acontece independentemente do aplicativo que você estava quando você tocou a notificação. Por exemplo, se você estiver no Gmail escrevendo uma mensagem, e você clicar a comunicação por um único e-mail,

você vai imediatamente para o e-mail. Tocar no voltar leva você para a caixa de entrada e, em seguida, a tela inicial, ao invés de levá-lo para a mensagem que você estava escrevendo.

Atividade especial

O usuário vê apenas esta atividade se é iniciado a partir de uma notificação. Em certo sentido, a atividade se estende a notificação, fornecendo informações que seriam difíceis para exibir na própria notificação. Para esta situação, configurar o PendingIntent para começar em uma nova tarefa. Não há necessidade de criar uma pilha de volta, porém, porque a atividade começou não faz parte do fluxo da aplicação atividade. Clicando no voltar ainda vai levar o usuário para a tela inicial.

Configurando um PendingIntent atividade regular

Para configurar uma PendingIntent que começa uma atividade entrada direta, siga estes passos:

- Definir hierarquia do aplicativo com uma atividade no manifesto.
- Adicionar suporte para o Android 4.0.3 e anteriores. Para fazer isso, especifique o pai da Atividade você está começando, adicionando um elemento `<meta-data>` como o filho dos `<activity>`.
- Para este elemento, definir `android:name = "android.support.PARENT_ACTIVITY"`. Coloque em `android:value = "<parent_activity_name>"` onde `<parent_activity_name>` é o valor do `android:name` para o elemento `<activity>` pai. Veja o seguinte XML para um exemplo.

Adicionar também suporte para o Android 4.1 e posterior. Para fazer isso, adicione o atributo `android:parentActivityName` ao elemento `<activity>` da atividade que você está começando.

O XML final deve ser semelhante a este:

```
<activity
  android:name=".MainActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
<activity
  android:name=".ResultActivity"
  android:parentActivityName=".MainActivity">
  <meta-data
    android:name="android.support.PARENT_ACTIVITY"
    android:value=".MainActivity"/>
</activity>
```

Criar uma pilha de volta com base na intent que começa a Atividade:

- Crie a intent de inicio a atividade.
- Criar um construtor de pilha chamando `TaskStackBuilder.create()`.
- Adicione a pilha de volta para o construtor pilha chamando `addParentStack()`. Para cada atividade na hierarquia que você definiu no manifesto, a pilha de volta contém um objeto `Intent` que inicia a atividade. Este método também adiciona configurações que começam a pilha em uma nova tarefa.

Nota: Embora o argumento para `addParentStack()` seja uma referência à atividade iniciada, a chamada de método não adiciona a `Intent` que inicia a atividade.

Adicione a Intent que inicia a atividade a partir da notificação, chamando `addNextIntent()`. Passe a Intent que você criou na primeira etapa como o argumento para `addNextIntent()`.

Se você precisar adicionar argumentos para objetos Intent na pilha chamando `TaskStackBuilder.editIntentAt()`. Isso às vezes é necessário para garantir que a atividade de destino exiba dados significativos quando o usuário navega para ele usando Back.

Obter uma `PendingIntent` para esta pilha de retorno chamando `getPendingIntent()`. Você pode então usar esta `PendingIntent` como o argumento para `setContentIntent()`.

O trecho de código a seguir demonstra o processo:

```
...
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0,
PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new
NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Configurando um PendingIntent para uma atividade especial

A seção seguinte descreve como configurar um PendingIntent atividade especial.

A atividade especial não precisa de uma pilha de retorno, deste modo você não tem que definir sua hierarquia de Atividade no manifesto, e você não tem que chamar `addParentStack()` para construir uma pilha de retorno. Em vez disso, use o manifesto para configurar as opções de tarefa, atividade e criar o PendingIntent chamando `getActivity()`:

Em seu manifesto, adicione os seguintes atributos para o elemento `<activity>` para a Atividade

```
android:name = "activityclass"
```

O nome da atividade de classe totalmente qualificado.

```
android:taskAffinity = ""
```

Combinado com a configuração `FLAG_ACTIVITY_NEW_TASK` que definiu no código, o que garante que essa atividade não entrara em uma tarefa padrão do aplicativo. Quaisquer tarefas existentes que têm afinidade padrão do aplicativo não são afetadas.

```
android:excludeFromRecents = "true"
```

Exclui a nova tarefa a partir das recentes, de modo que o usuário não pode acidentalmente navegar de volta para ela.

Este trecho mostra o elemento:

```
<activity
  android:name=".ResultActivity"
  ...
  android:launchMode="singleTask"
  android:taskAffinity=""
```

```

        android:excludeFromRecents="true">
</activity>
...

```

Construir e emitir a notificação:

- Criar uma Intent que inicie a atividade.
- Defina a atividade para começar em uma nova tarefa vazia chamando `setFlags()` com a configuração `FLAG_ACTIVITY_NEW_TASK` e `FLAG_ACTIVITY_CLEAR_TASK`.
- Defina quaisquer outras opções que você precisa para a intent.
- Criar um `PendingIntent` da Intent chamando `getActivity()`. Você pode então usar esta `PendingIntent` como o argumento para `setContentIntent()`.

O trecho de código a seguir demonstra o processo:

```

// Instantiate a Builder object.
NotificationCompat.Builder builder = new
NotificationCompat.Builder(this);
// Creates an Intent for the Activity
Intent notifyIntent =
    new Intent(new ComponentName(this, ResultActivity.class));
// Sets the Activity to start in a new, empty task
notifyIntent.setFlags(FLAG_ACTIVITY_NEW_TASK |
FLAG_ACTIVITY_CLEAR_TASK);
// Creates the PendingIntent
PendingIntent notifyIntent =
    PendingIntent.getActivity(
        this,
        0,
        notifyIntent
        PendingIntent.FLAG_UPDATE_CURRENT

```

```
);

// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```

Exibindo Progresso em uma Notificação

As notificações podem incluir um indicador de progresso animado que mostra aos usuários o status de uma operação em curso. Se você pode estimar quanto tempo a operação demora e quanto ela é completa, em qualquer momento você pode usar a forma "determinada" do indicador (uma barra de progresso). Se não for possível estimar a duração da operação, utilize o formulário de "indeterminado" do indicador (um indicador de atividade).

Indicadores de progresso são exibidos com a implementação da plataforma da classe `ProgressBar`.

Para usar um indicador de progresso em plataformas começando com Android 4.0, chamar `setProgress()`. Para versões anteriores, você deve criar o seu layout de notificação personalizado que inclui um `ProgressBar`.

As seguintes seções descrevem como exibir o progresso em uma notificação usando `setProgress()`.

Exibindo um indicador de progresso de duração fixa

Para exibir uma barra de progresso determinada, adicionar a barra à sua notificação chamando `setProgress()`, `setProgress(max, progresso, falso)` e, em seguida, emitir a notificação. Como seus recursos de operação, o progresso de incremento, atualiza a notificação. No final da operação, o progresso deve ser igual a `max`. Uma forma comum de chamar `setProgress()` é definir o máximo para 100 e, em seguida, incrementar o progresso como uma "porcentagem concluída" para valor da operação.

Você pode deixar a barra de progresso sendo mostrada quando a operação é feita, ou removê-lo. Em qualquer caso, lembre-se de atualizar o texto da notificação para mostrar que a operação foi concluída. Para remover a barra de progresso, chamar `setProgress()`, `setProgress(0, 0, false)`. Por exemplo:

```
...
mNotifyManager =
    (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value, the
                // current completion percentage, and "determinate"
```

```

// state
mBuilder.setProgress(100, incr, false);
// Displays the progress bar for the first time.
mNotifyManager.notify(0, mBuilder.build());
// Sleeps the thread, simulating an operation
// that takes time
try {
    // Sleep for 5 seconds
    Thread.sleep(5*1000);
} catch (InterruptedException e) {
    Log.d(TAG, "sleep failure");
}
}
// When the loop is finished, updates the notification
mBuilder.setContentText("Download complete")
// Removes the progress bar
.setProgress(0,0,false);
mNotifyManager.notify(ID, mBuilder.build());
}
}
// Starts the thread by calling the run() method in its Runnable
).start();

```

As notificações resultantes são mostrados na figura 6. No lado esquerdo é uma notificação instantâneo da durante a operação; no lado direito é uma notificação instantâneo após a operação ser concluída.

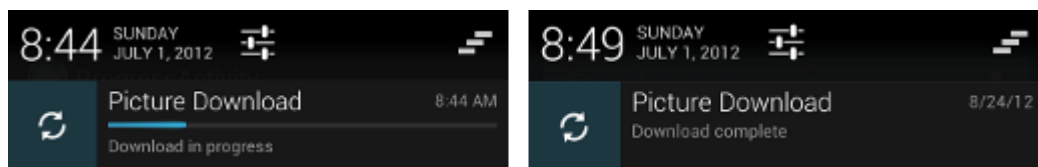


Figura 6. A barra de progresso durante e após a operação.

Exibindo um indicador de atividade contínua

Para exibir um indicador de atividade indeterminado, adicioná-lo à sua notificação com `setProgress(0, 0, true)` (os dois primeiros argumentos são ignorados), e emitidos na notificação. O resultado é um indicador que tem o mesmo estilo, como uma barra de progresso, exceto que sua animação está em curso.

Emitir a notificação, no início da operação. A animação será executado até que modifique sua notificação. Quando a operação é feita, chame `setProgress()`, `setProgress(0, 0, false)` e depois atualizar a notificação para remover o indicador de atividade. Sempre fazer isso, caso contrário, a animação será executado mesmo quando a operação estiver concluída. Lembre-se também de mudar o texto da notificação, para indicar que a operação for concluída.

Para ver como indicadores de atividade de trabalho, referem-se ao trecho anterior. Localize as seguintes linhas:

```
// Sets the progress indicator to a max value, the current completion
// percentage, and "determinate" state
mBuilder.setProgress(100, incr, false);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```

Substitua as linhas que você encontrou pelas seguintes linhas:

```
// Sets an activity indicator for an operation of indeterminate length
mBuilder.setProgress(0, 0, true);
// Issues the notification
mNotifyManager.notify(0, mBuilder.build());
```


O indicador resultante é mostrada na figura 7:

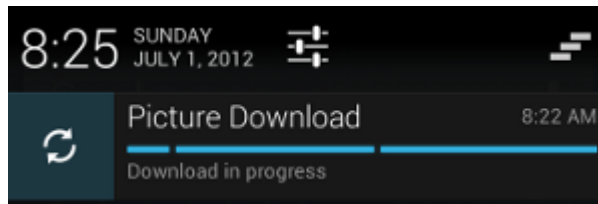


Figura 7. Um indicador de atividade em curso.

Layouts personalizados de Notificação

O quadro de notificações permite que você defina um layout notificação personalizada, que define a aparência da notificação em um objeto `RemoteViews`. Notificações de layout personalizado são semelhantes às notificações normais, mas elas são baseados em um `RemoteViews` definidas em um arquivo de layout XML.

A altura disponível para a notificação de um layout personalizado depende da view de notificação. Layouts de views normais estão limitadas a 64 DP, e layouts de visualização expandida são limitados a 256 DP.

Para definir um esquema de notificação personalizada, comece por instanciar um objeto `RemoteViews` que inclui um arquivo de layout XML. Então, em vez de chamar métodos como `setContentTitle()`, chame `setContent()`. Para definir detalhes do conteúdo da notificação personalizada, use os métodos em `RemoteViews` para definir os valores da view:

Criar um layout XML para a notificação em um arquivo separado. Você pode usar qualquer nome de arquivo que deseje, mas você deve usar a extensão `.xml`

Em seu aplicativo, use métodos `RemoteViews` para definir ícones de notificação e de texto. Coloque esse objeto em sua `RemoteViews NotificationCompat.Builder` chamando `setContent()`. Evite definir uma imagem

de fundo em seu objeto RemoteViews, porque a cor do texto pode se tornar ilegível.

A classe RemoteViews também inclui métodos que você pode usar para adicionar facilmente um cronômetro ou ProgressBar para o layout de sua notificação.

Atenção: Quando você usa uma notificação de layout personalizado, tomar cuidado especial para garantir que o seu layout personalizado trabalhe com orientações de dispositivos e resoluções diferentes. Enquanto este conselho se aplica a todos os layouts de view, é especialmente importante para as notificações porque o espaço na gaveta de notificação é muito restrito. Não faça do seu layout personalizado muito complexo, e não se esqueça de testá-lo em várias configurações.

Usando recursos de estilo para o texto notificação personalizada

Sempre use os recursos de estilo para o texto de uma notificação personalizada. A cor de fundo da notificação pode variar em diferentes dispositivos e versões, e utilizando recursos de estilo ajuda a explicar isso. A partir de Android 2.3, o sistema define um estilo para o texto padrão de layout de notificação. Se você usar o mesmo estilo em aplicativos que visam o Android 2.3 ou superior, você vai garantir que o seu texto é visível na o fundo de exibição.

Estilos e temas

Um estilo é um conjunto de propriedades que especificam a aparência e formato de uma view ou janela. Um estilo pode especificar propriedades, tais como altura, imagem de fundo, cor de fonte, tamanho de fonte, cor de fundo e muito mais. Um estilo é definido em um arquivo de recurso que é separado do XML que especifica o layout.

Estilos em Android partem de uma filosofia semelhante ao estilo em cascata no web-design que permitem a você separar o design do conteúdo.

Por exemplo, usando um estilo, você pode levar este layout XML:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

E transformá-lo para isso:

```
<TextView
    style="@style/CodeFont"
    android:text="@string/hello" />
```

Todos os atributos relacionados com o estilo foram removidos a partir do esquema XML e colocados em uma definição de estilo chamado CodeFont, que é então aplicada com o atributo de estilo. Você vai ver a definição para este modelo na seção seguinte.

Um tema é um estilo aplicado a uma Atividade inteira ou a aplicação, em vez de uma view individual (tal como no exemplo acima). Quando um estilo é aplicado como um tema, cada view na atividade ou aplicação irá aplicar cada propriedade de estilo que ele suporta. Por exemplo, você pode aplicar o mesmo estilo CodeFont como um tema para uma atividade e, em seguida, todo o texto dentro Atividade terá fonte monoespaçada verde.

Definição de estilos

Para criar um conjunto de estilos, salvar um arquivo XML em `res/values/directory` do seu projeto. O nome do arquivo XML é arbitrária, mas deve usar o `.xml` e ser salvo no `res/values/pasta`.

O nó raiz do arquivo XML deve ser `<resources>`.

Para cada estilo que deseja criar, adicionar um elemento `<style>` para o arquivo com um nome que identifica o estilo (este atributo é necessário). Em seguida, adicione um elemento `<item>` para cada propriedade desse estilo, com um nome que declara a propriedade de estilo e um valor para ir com ele (este atributo é necessário). O valor para o `<item>` pode ser uma seqüência de palavras-chave, uma cor hexadecimal, uma referência a outro tipo de recurso, ou outro valor, dependendo da propriedade de estilo. Aqui está um arquivo de exemplo com um estilo simples:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont"
parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Cada item do elemento `<resources>` é convertido em um objeto de recurso de aplicativo em tempo de compilação, que pode ser referenciado pelo valor do atributo nome do elemento `<style>`. Este estilo exemplo pode ser referenciada a partir de um layout XML como `@/style/CodeFont` (como demonstrado na introdução acima).

O atributo `parent` no elemento `<style>` é opcional e especifica o ID do recurso de outro estilo a partir do qual este estilo deve herdar propriedades. Você pode, então, substituir as propriedades de estilo herdado se você quiser.

Lembre-se, um estilo que você deseja usar como uma atividade ou tema de aplicativo é definido em XML exatamente o mesmo como um estilo para a View. Um estilo como o definido acima pode ser aplicado como um estilo para uma view única ou como um tema para uma atividade inteira ou aplicação.

Herança

O atributo `parent` no elemento `<style>` permite que você especifique um estilo a partir do qual o seu estilo deve herdar propriedades. Você pode usar isso para herdar propriedades de um estilo existente e, então, definir apenas as propriedades que você deseja alterar ou acrescentar. Você pode herdar de estilos que você mesmo criou ou de estilos que são construídos na plataforma. Por exemplo, você pode herdar a aparência da plataforma do Android padrão de texto e, em seguida, modificá-lo:

```
<style name = "GreenText" parent =
"@android:style/TextAppearance">
    <item name="android:textColor">#00FF00</item>
</style>
```

Se você quer herdar estilos que você mesmo definiu, você não tem que usar o atributo `parent`. Em vez disso, apenas o prefixo do nome do estilo que deseja herdar no nome de seu novo estilo, separados por um ponto. Por exemplo, para criar um novo estilo que herda o estilo `CodeFont` definido acima, mas fazer a cor vermelha, você pode criar o novo estilo como esta:

```
<style name="CodeFont.Red">
    <item name="android:textColor">#FF0000</item>
</style>
```

Observe que não há nenhum atributo parent na tag <style>, mas porque o atributo name começa com o nome do estilo CodeFont (que é um estilo que você tenha criado), este estilo herda todas as propriedades de estilo dele. Este estilo, então substitui a propriedade android:textColor para fazer o texto vermelho. Você pode fazer referência a esse novo estilo como @/style/CodeFont.Red.

Você pode continuar herdando quantas vezes você quiser, por nomes de encadeamento com pontos. Por exemplo, você pode estender CodeFont.Red deve ser maior, com:

```
<style name="CodeFont.Red.Big">  
    <item name="android:textSize"> 30sp </ item>  
</ Style>
```

Este herda de ambos os estilos e CodeFont CodeFont.Red, em seguida, adiciona a propriedade android:textSize.

Nota: Esta técnica de herança por encadeamento nomeada só funciona para estilos definidos por seus próprios recursos. Você não pode herdar estilos embutido do Android desta forma. Para fazer referência a um estilo de built-in, como TextAppearance, você deve usar o atributo parent.

Propriedades de estilo

Agora que você entende como um estilo definido, você precisa saber que tipo de propriedades estão disponíveis para poderem ser definidas em um estilo pelo elemento <item>. Você provavelmente está familiarizado com alguns já, como layout_width e textColor. Claro, há muito mais propriedades de estilo que você pode usar.

O melhor lugar para encontrar todas as propriedades que se aplicam a uma view específica é a referência de classe correspondente, que lista todos os atributos suportados XML. Por exemplo, todos os atributos listados na tabela

de atributos TextView XML pode ser usado em uma definição de estilo para um elemento TextView (ou uma das suas subclasses). Um dos atributos listados na referência é android:inputType, então onde você normalmente poderia colocar o atributo android:inputType em um elemento <EditText>, como esta:

```
<EditText
    android:inputType="number"
    ... />
```

Você pode, em vez disso criar um estilo para o elemento EditText que inclui esta propriedade:

```
<style name="Numbers">
    <item name="android:inputType">number</item>
    ...
</style>
```

Portanto, o seu XML para o layout pode implementar esse estilo:

```
<EditText
    style="@style/Numbers"
    ... />
```

Este exemplo simples pode parecer mais trabalho, mas quando você adicionar mais propriedades de estilo e utilizar novamente o estilo em vários lugares, o ganho pode ser enorme.

Para uma referência de todas as propriedades de estilo disponíveis, consulte a tabela de atributos. Tenha em mente que todos os objetos de visualização podem não aceitar todos os atributos de estilo, então você deve normalmente referem-se a classe View específica para propriedades de estilo suportados. No entanto, se você aplicar um estilo a uma exibição que não suporta todas as propriedades de estilo, a View serão aplicadas apenas as propriedades que são suportados e simplesmente ignora as outras.

Algumas propriedades de estilo, no entanto, não são suportados por qualquer elemento View e só podem ser aplicados como um tema. Essas propriedades de estilo se aplicam a toda a janela e não a qualquer tipo de exibição. Por exemplo, as propriedades de estilo de um tema pode ocultar o título da aplicação, ocultar a barra de status, ou mudar o fundo da janela. Este tipo de propriedades de estilo não pertencem a qualquer objeto View. Para descobrir essas propriedades-tema apenas de estilo, olhar na tabela de atributos, atributos que começam com window. Por exemplo, windowNoTitle e windowBackground são propriedades de estilo que são eficazes apenas quando o estilo é aplicado como um tema para uma atividade ou aplicação.

Nota: Não se esqueça de prefixo os nomes das propriedades de cada elemento <item> com o android:namespace. Por exemplo: <item name="android:inputType">.

Aplicando estilos e temas para a interface do usuário

Há duas maneiras de definir um estilo:

Para uma view individual, adicionando o atributo de estilo a um elemento View no XML para seu layout.

Ou, para uma atividade inteira ou aplicação, adicionando o atributo android:theme para os elementos <activity> ou <application> no manifesto do Android.

Quando você aplica um estilo a uma única exibição no layout, as propriedades definidas pelo estilo são aplicadas apenas para a view. Se um estilo é aplicado a uma ViewGroup, os elementos de Visualização não herdarão as propriedades do estilo somente o elemento ao qual você aplicar diretamente o estilo vai aplicar suas propriedades.

Para aplicar uma definição de estilo como um tema, você deve aplicar o estilo para uma atividade ou aplicação no manifesto do Android. Quando você faz isso, cada vista dentro da atividade ou aplicação será aplicada a cada propriedade que ele suportar. Por exemplo, se você aplicar o estilo CodeFont dos exemplos anteriores a uma atividade, então todos os elementos de Visualização que suportam as propriedades de estilo de texto irão aplicá-los. Qualquer elemento da view que não suporte as propriedades irão ignorá-los. Se uma view suporta apenas algumas das propriedades, ela será aplicada apenas nestas propriedades.

Aplicar um estilo a uma exibição

Veja como definir um estilo para a View no layout XML:

```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello" />
```

Agora este TextView será denominado como definido pelo estilo chamado CodeFont.

Nota: O atributo de estilo não usa o prefixo android:namespace.

Aplicar um tema para uma atividade ou aplicação

Para definir um tema para todas as atividades de sua aplicação, abra o arquivo AndroidManifest.xml e edite a tag <application> para incluir o atributo android:theme com o nome do estilo. Por exemplo:

```
<application android:theme="@style/CustomTheme">
```

Se você quer um tema aplicado a apenas uma atividade na sua aplicação, em seguida, adicionar o atributo android:theme para a tag <activity>.

Assim como o Android fornece outros recursos embutidos, há muitos temas pré-definidos que você pode usar, para evitar escrever-los sozinho. Por exemplo, você pode usar o tema de diálogo e fazer a sua Atividade aparecer como uma caixa de diálogo:

```
<activity android:theme="@android:style/Theme.Dialog">
```

Ou se você quiser o fundo para ser transparente, utilizar o tema translúcido:

```
<activity android:theme="@android:style/Theme.Translucent">
```

Se você gosta de um tema, mas quer ajustá-lo, basta adicionar o tema como o parent de seu tema personalizado. Por exemplo, você pode modificar o tema de luz tradicional de usar a sua própria cor como esta:

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
  <item
name="android:windowBackground">@color/custom_theme_color</item>
  <item
name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

(Note que a cor precisa fornecido como um recurso separado aqui porque o atributo android:windowBackground suporta apenas uma referência a outro recurso, ao contrário do Android:ColorBackground, não pode ser dada uma cor literal)

Agora use CustomTheme em vez de Theme.Light dentro do manifesto do Android:

```
<activity android:theme="@style/CustomTheme">
```

Selecione um tema baseado na versão de plataforma

As versões mais recentes do Android têm temas adicionais disponíveis para aplicações, e você pode querer usar estes durante a execução nas plataformas enquanto continuam a ser compatível com versões mais antigas. Você pode fazer isso através de um tema personalizado que usa seleção de recursos para alternar entre os temas de pais diferentes, com base na versão da plataforma.

Por exemplo, aqui é a declaração de um tema personalizado que é simplesmente o padrão plataformas tema luz padrão. Ele iria em um arquivo XML em res / valores (geralmente res / valores / styles.xml):

```
<style name="LightThemeSelector" parent="android:Theme.Light">
    ...
</ Style>
```

Para ter este tema usar o novo tema holográfico quando o aplicativo é executado no Android 3.0 (API Level 11) ou superior, você pode colocar uma declaração alternativa para o tema em um arquivo XML em res/values-v11, mas fazer o tema pai o tema holográfico:

```
<style name="LightThemeSelector"
parent="android:Theme.Holo.Light">
    ...
</ Style>
```

Agora use este tema como se fosse qualquer outro, e seu pedido será automaticamente mudar para o tema holográfico se em execução no Android 3.0 ou superior.

A lista dos atributos padrão que você pode usar em temas podem ser encontrados em `R.styleable.Theme`.

Para mais informações sobre o fornecimento de recursos alternativos, como temas e layouts, com base na versão de plataforma ou outras configurações do dispositivo, consulte o documento Fornecer Recursos.

Usando estilos Plataforma e Temas

A plataforma Android oferece uma grande coleção de estilos e temas que você pode usar em suas aplicações. Você pode encontrar uma referência de todos os estilos disponíveis na classe `R.style`. Para usar os estilos listados aqui, substituir todos os sublinhados no nome do estilo com um ponto. Por exemplo, você pode aplicar o tema `Theme_NoTitleBar` com `"@android:style/Theme.NoTitleBar"`.

A referência `R.style`, no entanto, não é bem documentado e não descrever minuciosamente os estilos, por isso vendo o código fonte para estes estilos e temas vai lhe dar uma melhor compreensão do que propriedades de estilo que cada um oferece. Para uma melhor referência aos estilos Android e temas, ver o código fonte seguinte:

- Estilos de Android (`styles.xml`) (arquivo em anexo)
- Temas Android (`themes.xml`) (arquivo em anexo)

Esses arquivos vão ajudar você a aprender através do exemplo. Por exemplo, no código-fonte temas do Android, você encontrará uma declaração para `<style name="Theme.Dialog">`. Nesta definição, você vai ver todas as propriedades que são usados para estilos de diálogos que são usados pela estrutura Android.

Mensagens em Android

Intents e intents filters

Três dos principais componentes de uma aplicação - atividades, serviços e receptores de broadcast - são ativados através de mensagens, chamadas de intents. Intents de mensagens são uma facilidade para fazer uma ligação em tempo de execução entre os componentes das aplicações iguais ou diferentes. A intent em si, um objeto Intent, é uma estrutura de dados passiva segurando uma descrição abstrata de uma operação a ser realizada - ou, muitas vezes, no caso de transmissões, a descrição de algo que já aconteceu e está sendo anunciado. Existem mecanismos separados para fornecer as intents para cada tipo de componente:

Um objeto Intent é passado para `Context.startActivity()` ou `Activity.startActivityForResult()` para iniciar uma atividade ou ter uma atividade já existente para fazer algo novo. (Também pode ser passado para `Activity.setResult()` para retornar informações para a atividade que chamou `startActivityForResult()`.)

Um objecto é passado para Intent `Context.startService()` para iniciar um serviço ou fornecer novas instruções para um serviço contínuo. Do mesmo modo, a intent pode ser passada para `Context.bindService()` para estabelecer uma ligação entre o componente de chamada e serviço de destino. Ele pode, opcionalmente iniciar o serviço se ele não estiver em execução.

Objetos Intent passados para qualquer um dos métodos de transmissão (por exemplo, `Context.sendBroadcast()`, `Context.sendOrderedBroadcast()`, ou `Context.sendStickyBroadcast()`) são entregues a todos os receptores de broadcast interessados. Muitos tipos de transmissões originários de código do sistema.

Em cada caso, o sistema localiza a atividade Android apropriada, serviço, ou um conjunto de receptores de broadcast para responder à intent, e instanciar-los se necessário. Não há sobreposição dentro destes sistemas de mensagens: as intent de emissões são entregues somente para transmitir receptores nunca, a atividades ou serviços. Uma intent passada para `startActivity()` é fornecida apenas a uma actividade, nunca a um receptor do serviço ou de transmissão, e assim por diante.

Este documento começa com uma descrição de objetos intent. Em seguida, ele descreve as regras que o Android usa para mapear as intent de componentes - como ele resolve o componente que deve receber uma mensagem de uma intent. Para intents que não explicitamente nomeam um componente de destino, este processo envolve testar o objeto Intent para cada um dos filtros de intent associados a alvos potenciais.

Objetos Intent

Um objeto Intent é um conjunto de informações. Ele contém informação de interesse para o componente que recebe a intent (tal como a ação a ser tomada, e os dados para agir) mais informação de interesse para o sistema Android (tais como a categoria de componentes que devem lidar com o propósito e instruções sobre como iniciar uma atividade de destino).

Principalmente, podem conter o seguinte:

Nome do componente

O nome do componente que deve lidar com a intent. Este campo é um objeto `ComponentName` - uma combinação do nome da classe totalmente qualificado do componente de destino (por exemplo, `"com.example.project.app.FreneticActivity"`) e o nome do pacote definido no arquivo de manifesto do aplicativo em que o componente reside (por exemplo, `"com.example.project"`). A parte do pacote do nome do componente e definir o nome do pacote no manifesto não necessariamente tem que corresponder.

O nome do componente é opcional. Se for definido, o objeto Intent é entregue a uma instância da classe designada. Se não estiver definido, o Android usa outras informações no objeto Intent para localizar um alvo adequado.

O nome do componente é configurado por `setComponent()`, `SetClass()`, ou `setClassName()` e lido por `getComponent()`.

Ação

Uma string com o nome da ação a ser realizada - ou, no caso de transmissão de intents, a ação que ocorreu e está sendo relatada. A classe Intenção define um número de constantes de ações, incluindo estes:

Constante	Alvo	Ação
<code>ACTION_CALL</code>	activity	iniciar uma chamada telefônica.
<code>ACTION_EDIT</code>	activity	exibição de dados para o usuário editar.
<code>ACTION_MAIN</code>	activity	iniciar como a atividade inicial de uma tarefa, sem entrada e sem saída de dados.
<code>ACTION_SYNC</code>	activity	Sincronizar dados em um servidor com dados no dispositivo móvel.
<code>ACTION_BATTERY_LOW</code>	broadcast receiver	aviso de que a bateria está fraca.
<code>ACTION_HEADSET_PLUG</code>	broadcast receiver	fone de ouvido foi conectado ao dispositivo, ou desconectado do mesmo.
<code>ACTION_SCREEN_ON</code>	broadcast receive	tela foi ligada.
<code>ACTION_TIMEZONE_CHANGED</code>	broadcast receiver	o fuso horário mudou

Veja a descrição da classe Intent para uma lista de pré-definidas das ações constantes genéricas. Outras ações são definidas em outras partes do API do Android. Você também pode definir suas próprias seqüências de ação para ativar os componentes em sua aplicação. Aqueles que você inventar deve incluir o pacote de aplicativos como um prefixo - por exemplo: "com.example.project.SHOW_COLOR".

A ação determina em grande parte como o resto da intent - em particular os dados e campos extras - tanto como um nome de método determina um conjunto de argumentos e um valor de retorno. Por esta razão, é uma boa idéia usar nomes de ação que são tão específico quanto possível, e para acoplá-los firmemente a outros campos da intent. Em outras palavras, em vez de definir uma ação de isolamento, definir um protocolo completo para os objectos Intent que seus componentes podem manipular.\

A ação de um objecto Intent é definido pelo `setAction()` e lido por `getAction()`.

Dados

O URI dos dados a serem executados e o tipo MIME de dados. Ações diferentes são combinados com diferentes tipos de especificações de dados. Por exemplo, se o campo de ação é `ACTION_EDIT`, o campo de dados deve conter o URI do documento a ser exibido para edição. Se a ação for `ACTION_CALL`, o campo de dados seria um telefone: URI com o número a ser chamado. Da mesma forma, se a ação for `ACTION_VIEW` e o campo de dados é um `http:URI`, a atividade de recebimento seria chamado para baixar e exibir todos os dados que a URI se refere.

Ao combinar a intent de um componente que é capaz de manipular os dados, muitas vezes é importante saber o tipo de dados (seu tipo MIME), além de sua URI. Por exemplo, um componente capaz de exibir dados de imagem não deve ser chamado a motrar um arquivo de áudio.

Em muitos casos, o tipo de dados pode ser inferida a partir da URI - particularmente pelo teor: URIs, que indicam que os dados estão localizados no dispositivo e controlado por um provedor de conteúdos. Mas o tipo também pode ser explicitamente definido no objeto Intent. O método setData() especifica os dados apenas como uma URI, setType() especifica apenas como um tipo de MIME e setDataAndType() especifica tanto como um URI e um tipo MIME. A URI é lida por getData() e o tipo de getType().

Categoria

Uma seqüência contendo informações adicionais sobre o tipo de componente que deve lidar com a intent. Qualquer número de descrições de categoria podem ser colocados em um objeto Intent. Como faz para as ações, a classe Intent define constantes diversas de categoria, incluindo estes:

Constante	Significado
CATEGORY_BROWSABLE	A atividade alvo pode ser facilmente invocado pelo navegador para exibir dados referenciados por um link - por exemplo, uma imagem ou uma mensagem de e-mail.
CATEGORY_GADGET	A atividade pode ser incorporada dentro de outra atividade que hospeda gadgets.
CATEGORY_HOME	A atividade exibe a tela inicial, a primeira tela que o usuário vê quando o dispositivo é ligado ou quando o botão Home é pressionado.
CATEGORY_LAUNCHER	A atividade pode ser a atividade inicial de uma tarefa e está listado na tela do menu de nível superior.
CATEGORY_PREFERENCE	A atividade tem como alvo um painel de preferências.

O método AddCategory() coloca uma categoria em um objeto Intent, removeCategory() exclui uma categoria adicionada anteriormente, e GetCategories() é o conjunto de todas as categorias atualmente no objeto.

Extras

Pares chave-valor para informações adicionais que devem ser entregues ao componente que manuseia a intent. Assim como algumas ações estão emparelhados com determinados tipos de dados URIs, alguns estão emparelhados com extras específicos. Por exemplo, uma intent `ACTION_TIMEZONE_CHANGED` tem um "fuso horário" extra que identifica o novo fuso horário, e `ACTION_HEADSET_PLUG` tem um "estado" extra indicando se o fone de ouvido está conectado ou desconectado, bem como um "nome" extra para o tipo de fone de ouvido. Se você tivesse que criar uma ação `SHOW_COLOR`, o valor da cor seria definido em um par chave-valor extra.

O objeto Intent tem uma série de métodos `set...()` para a inserção de vários tipos de dados adicionais e um conjunto semelhante de `get...()` para ler os dados. Estes métodos são comparáveis àquelas dos objetos Bundle. Na verdade, os extras podem ser instalados e lido como um pacote usando os métodos `putExtras()` e `getExtras()`.

Flags

Flags de vários tipos. Muitos instruem ao sistema Android como iniciar uma atividade (por exemplo, tarefa que a atividade deve pertencer) e como tratá-lo depois que ele é iniciado (por exemplo, se ele pertence na lista de atividades recentes). Todos esses sinalizadores são definidos na classe Intent.

O sistema Android e os aplicativos que vêm com a plataforma empregam objetos Intent tanto para mandar sistema originados transmissões e ativar ações definidas pelos componentes do sistema.

Resolvendo Intent

Intento podem ser divididos em dois grupos:

Intents explícitas que designam o componente de destino pelo seu nome (o campo nome do componente, mencionado anteriormente, tem um conjunto de valores). Desde os nomes dos componentes que geralmente não ser conhecidos para os desenvolvedores de outras aplicações, intents explícitas são normalmente utilizados para aplicação interna de mensagens - como uma atividade de iniciar um serviço subordinado ou início de uma atividade irmã.

Intents implícitas não nomeam um componente alvo (o campo para o nome do componente está em branco). Intents implícitas são freqüentemente usados para ativar os componentes em outras aplicações.

Android oferece uma intent explícita de uma instância da classe de destino designado. Nada no objeto Intent além dos assuntos componente de nome para determinar qual o componente que deve receber a intent.

Uma estratégia diferente é necessário para as intents implícitas. Na ausência de um alvo designado, o sistema Android deve encontrar a melhor componente (ou componentes) para lidar com a intent - uma atividade única ou serviço para executar a ação pedida ou o conjunto de receptores de broadcast para responder à mensagem publicada. Faremos isso por meio da comparação do conteúdo do objecto Intent com filtros de intents, as estruturas associadas com os componentes que podem potencialmente receber as intents. Filtros anunciam os recursos de um componente e delimitam as intents que ele pode manipular. Eles abrem o componente para a possibilidade de receber as intents implícitas do tipo anunciado. Se um componente não tem qualquer filtro de intent, pode receber apenas intent explícitas. Um componente com filtros podem receber as intents explícitas e implícitas.

Apenas três aspectos de um objeto Intent são consultados quando o objeto é testado em um filtro de intent:

Ação

Dados (tanto do tipo URI e dados)

Categoria

As extras e flags não têm qualquer papel na resolução de qual componente recebe uma intent.

Filtros de Intent

Para informar o sistema que as intents implícitas que podem segurar, atividades, serviços e receptores de broadcast podem haver um ou mais filtros de intent. Cada filtro descreve uma capacidade do componente, um conjunto de intents que o componente está disposta a receber. É, com efeito, filtros de intents de um tipo desejado, enquanto filtrando intents indesejáveis - mas só são indesejadas intent implícitas (aquelas que não nomeam uma classe de destino). Uma intent explícita é sempre entregue a seu alvo, não importa o que ela contém, o filtro não é consultado. Mas a intent implícita é entregue a um componente apenas se pode passar através de um dos filtros do componente.

Um componente tem filtros separados para cada trabalho que pode fazer, cada face pode apresentar ao usuário. Por exemplo, a atividade NoteEditor do aplicativo Pad amostra Nota tem dois filtros - um para iniciar-se com uma nota específica que o usuário pode visualizar ou editar, e outro para começar com uma nota nova, em branco que o usuário pode preencher e salvar.

Um filtro de intent é uma instância da classe `IntentFilter`. No entanto, desde que o sistema Android deve saber sobre os recursos de um componente antes que ele pode lançar esse componente, filtros de intent geralmente não são criados em código Java, mas no arquivo de manifesto do aplicativo (`AndroidManifest.xml`) com elementos `<intent-filter>`. (A única exceção seria filtros para receptores de broadcast que são registrados dinamicamente chamando `Context.registerReceiver()`; estão diretamente criados como objetos `IntentFilter`.)

Um filtro tem campos que são paralelos a ação, dados e campos de categoria de um objeto Intent. Uma intent implícita é testada pelos filtros em todas as três áreas. Para ser entregue ao componente que possui o filtro, ele deve passar por todos os três testes. Se falhar, mesmo uma delas, o sistema Android não irá entregá-lo ao componente - pelo menos não com base no filtro. No entanto, uma vez que um componente pode ter vários filtros de intent, uma intent que não passa através de um dos filtros de um componente pode fazê-lo através de um outro.

Cada um dos três testes é descrito em pormenor a seguir:

Teste de ação

Um elemento <intent-filter> no arquivo de manifesto lista ações como subelementos <Action>. Por exemplo:

```
<intent-filter . . . >
  <action android:name="com.example.project.SHOW_CURRENT" />
  <action android:name="com.example.project.SHOW_RECENT" />
  <action android:name="com.example.project.SHOW_PENDING" />
  . . .
</intent-filter>
```

Como mostra o exemplo, enquanto um objeto Intent nomeia apenas uma única ação, um filtro pode listar mais de um. A lista não pode ser vazio, um filtro deve conter pelo menos um elemento <action>, ou ele vai bloquear todas as intents.

Para passar neste teste, a ação especificada no objeto de Intent deve corresponder a uma das ações listadas no filtro. Se o objeto ou o filtro não especifica uma ação, os resultados são os seguintes:

Se o filtro não consegue listar todas as ações, não há nada para a intent de corresponder, por isso todas as intents falham no teste. Não existem intents que possam passar através do filtro.

Por outro lado, um objecto Intent que não especifica um recurso automaticamente passa o teste, desde que o filtro contém pelo menos uma ação.

Exame da categoria

Um elemento `<intent-filter>` também lista categorias como subelementos. Por exemplo:

```
<intent-filter . . . >
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  . . .
</intent-filter>
```

Nota-se que as constantes descritas anteriormente para as ações e as categorias não são usadas no arquivo de manifesto. Os valores de string completos são usados. Por exemplo, o "android.intent.category.BROWSABLE" string no exemplo acima corresponde à constante `CATEGORY_BROWSABLE` mencionado anteriormente. Da mesma forma, a string "android.intent.action.EDIT" corresponde à constante `ACTION_EDIT`.

Para a intent passar no teste de categoria, cada categoria no objeto de Intent deve corresponder a uma categoria no filtro. O filtro pode listar outras categorias, mas não pode omitir qualquer uma que estão na intent.

Em princípio, portanto, um objeto Intent sem categorias devem sempre passar neste teste, independentemente do que está no filtro. No entanto, com uma excepção, Android trata todas as intents implícitas passados para `startActivity()` como se contivessem pelo menos uma categoria:

"android.intent.category.DEFAULT" (a constante CATEGORY_DEFAULT). Portanto, as atividades que estão dispostas a receber as intents implícitas deve incluir "android.intent.category.DEFAULT" em seus filtros de intents. (Filtros como configurações "android.intent.action.MAIN" e "android.intent.category.LAUNCHER" são a exceção. Eles marcam as atividades que começam novas tarefas e que são representados na tela. Eles podem incluir "android.intent . category.DEFAULT " na lista de categorias, mas não precisa.)

Teste dos Dados

Como a ação e as categorias, a especificação de dados para um filtro de intent está contido em um subelemento. E, como nesses casos, o subelemento pode aparecer várias vezes, ou não. Por exemplo:

```
<intent-filter . . . >
  <data android:mimeType="video/mpeg" android:scheme="http" . . . />
  <data android:mimeType="audio/mpeg" android:scheme="http" . . . />
  . . .
</intent-filter>
```

Cada elemento <data> pode especificar um URI e um tipo de dados (MIME tipo de mídia). Existem atributos separados - host, schema, port e o path - para cada parte da URI:

scheme://host:port/path

Por exemplo, no seguinte URI,

Contente:./ / com.example.project: 200/folder/subfolder/etc

o esquema é "content", o host é "com.example.project", port é "200", e o caminho é "pasta/subpasta/etc". O host e port juntos constituem a autoridade URI; se um host não for especificado, a porta é ignorada.

Cada um desses atributos é opcional, mas elas não são independentes um do outro, para uma autoridade de ser significativa, um esquema também deve ser especificado. Para um caminho a ser significativo, tanto regime e uma autoridade deve ser especificado.

Quando a URI de um objeto Intent é comparada com uma especificação URI de um filtro, que é comparada apenas com as partes do URI realmente mencionados no filtro. Por exemplo, se um filtro especifica apenas um esquema, todos os URIs com esse esquema corresponde ao filtro. Se um filtro especifica um esquema e uma autoridade, mas nenhum caminho, todos os URIs com o mesmo esquema e autoridade, independentemente de seus caminhos. Se um filtro especifica um esquema, uma autoridade e um caminho, só URIs com o mesmo esquema, a autoridade e a partida caminho. No entanto, a especificação de caminho no filtro pode conter caracteres especiais para exigir apenas uma correspondência parcial do caminho.

O atributo tipo de um elemento <dados> especifica o tipo MIME dos dados. É mais comum em filtros do que um URI. Tanto o objeto intent e do filtro pode usar um coringa "*" para o campo subtipo - por exemplo, "text/*" ou "áudio/*" - indicando eventuais partidas subtipo.

O teste compara os dados, tanto o URI e o tipo de dados no objeto Intent de um tipo de URI e de dados especificados no filtro. As regras são as seguintes:

Um objeto Intent que não contém nem URI nem um tipo de dados passa no teste somente se o filtro também não especificar qualquer tipo de URIs ou dados.

Um objeto Intent que contém uma URI, mas nenhum tipo de dados (e um tipo não pode ser inferida a partir da URI) passa no teste somente se o seu URI corresponde a um URI no filtro e o filtro também não especifica um tipo. Este será o único caso de URIs como mailto: e tel: que não se referem a dados reais.

Um objeto Intent que contém um tipo de dados, mas não uma URI passa no teste somente se o filtro de lista o mesmo tipo de dados e da mesma forma não especificar um URI.

Um objeto Intent que contém tanto um URI e um tipo de dados (ou de um tipo de dados pode ser inferida a partir da URI) passa a parte tipo de dados de teste apenas se o seu tipo corresponde a um tipo listado no filtro. Ele passa a parte URI do teste ou se o sua URI corresponde a um URI no filtro ou se tem um conteúdo ou arquivo URI e o filtro não especificar um URI. Em outras palavras, um componente é presumida para suportar conteúdo e arquivo de dados se suas listas de filtros apenas um tipo de dados.

Se a intent pode passar através dos filtros de mais de uma atividade ou serviço, o usuário poderá ser solicitado que o componente para ativar. Uma exceção é levantada se nenhuma meta pode ser encontrada.

Casos comuns

A última regra acima para o teste de dados, a regra, reflete a expectativa de que os componentes são capazes de obter dados locais de um arquivo ou provedor de conteúdo. Portanto, seus filtros podem listar apenas um tipo de dados e não precisa nomear explicitamente o conteúdo e arquivo de esquemas. Este é um caso típico. Um elemento <dados> como o seguinte, por exemplo, diz ao android que o componente pode obter os dados de imagem a partir de um provedor de conteúdo e exibi-lo:

```
<data android:mimeType="image/*" />
```

Como a maioria dos dados disponíveis é dispensado por provedores de conteúdo, filtros que especificam um tipo de dados, mas não uma URI são, talvez, o mais comum.

Outra configuração comum são filtros com um esquema e um tipo de dados. Por exemplo, um elemento `<data>` como o android diz que o componente pode obter dados de vídeo a partir da rede e exibi-lo:

```
<data android:scheme="http" android:type="video/*" />
```

Considere, por exemplo, o que faz o aplicativo do navegador quando o usuário segue um link em uma página web. Ele primeiro tenta exibir os dados (como poderia se a ligação foi para uma página HTML). Se ele não pode exibir os dados, ele reúne uma intent implícita com o tipo de esquema e de dados e tenta iniciar uma atividade que pode fazer o trabalho. Se não houver compradores, ele pede ao gerenciador de download para baixar os dados. Isso coloca-o sob o controle de um fornecedor de conteúdos, para uma piscina potencialmente maior de atividades (aqueles com filtros que apenas um nome do tipo de dados) podem responder.

A maioria dos aplicativos também têm uma maneira de começar de novo, sem uma referência a todos os dados particulares. Atividades que podem iniciar as aplicações têm filtros com "android.intent.action.MAIN" especificado como a ação. Se eles estão sendo representados no lançador de aplicativos, eles também especificam a categoria "android.intent.category.LAUNCHER":

```
<intent-filter . . . >
  <action android:name="android.intent.action.MAIN" />
  <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

Usando a intent de correspondência

Intents são correspondentes aos filtros de intent não só de descobrir um componente de destino para ativar, mas também para descobrir algo sobre o conjunto de componentes do dispositivo. Por exemplo, o sistema Android preenche o lançador de aplicativos, a tela de nível superior que mostra os aplicativos que estão disponíveis para o usuário, por encontrar todas as

atividades com filtros de intent que especificam a ação "android.intent.action.MAIN" a categoria "android.intent.category.LAUNCHER" (como ilustrado na seção anterior). Em seguida, exibe os ícones e rótulos de tais atividades na tela. Da mesma forma, ele descobre a tela inicial, olhando para a atividade de "android.intent.category.HOME" em seu filtro.

Armazenamento de dados

SQLite

SQLite é uma biblioteca C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo RDBMS separado.

SQLite não é uma biblioteca de cliente usada para conectar com um grande servidor de banco de dados. SQLite é o servidor. A biblioteca SQLite lê e escreve diretamente para e do arquivo do banco de dados no disco.

O uso do SQLite é recomendado onde a simplicidade da administração, implementação e manutenção são mais importantes que incontáveis recursos que SGBDs mais voltados para aplicações complexas possivelmente implementam. Entretanto situações onde a simplicidade é a melhor escolha são muito mais freqüentes do que pode-se imaginar.

Exemplos de uso do SQLite não restrito a :

- sites com menos de cem mil requisições por dia,
- dispositivos e sistemas embarcados,
- aplicações desktop,
- ferramentas estatísticas e de análise,
- aprendizado de banco de dados,
- implementação de novas extensões à SQL.

Não se recomenda o uso do SQLite para sites com :

- muitos acessos,
- grande quantidades de dados (talvez maior que algumas duzias de gigabytes),
- sistemas com grande concorrência,

- aplicações cliente/servidor.

Algumas Características do SQLite:

- Software Livre /domínio público e Multiplataforma
- Mecanismo de armazenamento seguro com transações ACID
- Não necessita de instalação, configuração ou administração
- Implementa a maioria do SQL92
- O Banco de Dados é guardado em um único arquivo
- Suporta bases de dados acima de 2 terabytes
- Sem dependências externas

Utilizando bancos de dados SQL no Android

O que pretendemos aqui é saber como utilizar a API de acesso ao SQLite, o gerenciador de banco de dados nativo do Android.

Antes de falarmos do acesso ao banco, vamos ver o que é uma tabela.

Uma tabela é uma entidade que armazena uma lista de um dado modelo, assim como uma classe Java também armazena um modelo de alguma informação, cada linha da tabela armazena a informação de um modelo, seja essa informação a modelagem de um Cliente, uma Cadeira ou um Item emprestado para um amigo. É claro que existem muitas diferenças entre tabelas e classes Java, mas no sentido geral, ambas servem para armazenar informações de acordo com atributos comuns.

Uma das diferenças entre tabelas e objetos é que um objeto Java não é persistente, ou seja, as informações dentro dele são perdidas ao fechar o programa.

Vamos ver como criar um banco de dados simples, mas nem por isso rudimentar. Utilizaremos a modelagem de banco de um projeto

Empréstimos, então criaremos um projeto no Eclipse de nome Empreste. O nosso banco de dados contém somente duas tabelas. A tabela Empréstimos e a tabela Categorias, assim como modeladas abaixo:



Tabelas

A tabela Categoria nada mais é do que um INTEGER de nome `_ID` e um TEXT de nome `DESCRICAO`, que na verdade é o nome da categoria.

A tabela Empréstimos possui mais campos:

- `_ID`: Identificador único;
- `ITEM`: Nome do item a ser emprestado;
- `DESCRICAO`: Uma pequena descrição do item ou comentário;
- `STATUS`: Diz se o item está sendo emprestado para você ou se você está pegando o item emprestado, o SQLite não possui boolean e por isso utilizamos constantes inteiras para definir o estado.
- `STATUS_ALARME`: Também deveria ser um boolean para indicar se o alarme para lembrar de devolver o item está ativado ou não. Também são utilizadas constantes inteiras para representar o estado.

- DATA_DEVOLUCAO: O SQLite até tem um jeito de guardar DATE no banco, mas na verdade o que ele faz é uma conversão automática entre o DATE e um Long (que na verdade é um INTEGER no banco), para simplificar as coisas eu guardo o número de milissegundos do Date (o do Java) no banco declarando esse campo como INTEGER; (Fica mais claro ao ver o código)
- ID_CATEGORIA: É uma chave estrangeira da tabela Categoria para sabermos a qual categoria o item pertence, algo como “CDs”, “Livros” seriam boas categorias.
- ID_CONTATO: Guarda o ID de um contato da agenda telefônica do aparelho, como não sei dizer ao sistema que é uma chave estrangeira de lá, é sempre necessário ter uma atenção maior a esse campo para manter a integridade do sistema;
- NOME_CONTATO: Quando a pessoa que está pegando um item emprestado não consta na lista de contatos, utilizamos esse campo para entrar com o nome dela, caso contrário, ele permanecerá em branco.

Bem, são somente duas tabelas, mas acredito que vamos poder abordar vários aspectos do banco de dados Android utilizando os atributos das tabelas acima, o manuseio de “booleans”, “datas” e chaves estrangeiras são coisas que costumam gerar dúvidas a qualquer iniciante no desenvolvimento Android, isso será explicado posteriormente, vamos nos conter ao mais básico possível aqui.

Além dos tipos utilizados nas tabelas acima, INTEGER e TEXT, existem NULL, REAL e BLOB, mas basicamente utilizamos somente os três abaixo na maior parte do tempo:

- INTEGER: Serve para guardar int, Integer e Long;
- TEXT: Guarda String de qualquer tamanho;

- REAL: Guarda float e double;

Agora que sabemos o que queremos guardar, vamos começar a ver como persistir esses dados no banco.

Vamos criar uma classe chamada DbAdapter.java e dentro dela declarar o seguinte:

```
public static final String TABELA_CATEGORIA = "CATEGORIA";

public static final String COLUNA_ID_CATEGORIA = "_id";
public static final String COLUNA_DESCRICAO_CATEGORIA =
"DESCRICAO";

public static final String TABELA_EMPRESTIMOS = "EMPRESTIMOS";

public static final String COLUNA_ID_EMPRESTIMOS = "_id";
public static final String COLUNA_ITEM_EMPRESTIMOS = "ITEM";
public static final String COLUNA_DESCRICAO_EMPRESTIMOS =
"DESCRICAO";
public static final String COLUNA_STATUS_EMPRESTIMOS = "STATUS";
public static final String COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS =
"DATA_DEVOLUCAO";
public static final String COLUNA_STATUS_ALARME_EMPRESTIMOS =
"STATUS_ALARME";
public static final String COLUNA_ID_CATEGORIA_EMPRESTIMOS =
"ID_CATEGORIA";
public static final String COLUNA_ID_CONTATO_EMPRESTIMOS =
"ID_CONTATO";
public static final String COLUNA_CONTATO_EMPRESTIMOS = "CONTATO";

private static final String CATEGORIA_CREATE_TABLE = "CREATE TABLE "
+ TABELA_CATEGORIA + " (" + COLUNA_ID_CATEGORIA
```



```

+ " INTEGER PRIMARY KEY AUTOINCREMENT, "
+ COLUNA_DESCRICAO_CATEGORIA + " text not null);";

```

```

private static final String EMPRESTIMOS_CREATE_TABLE = "CREATE
TABLE "

```

```

+ TABELA_EMPRESTIMOS + " (" + COLUNA_ID_EMPRESTIMOS
+ " INTEGER PRIMARY KEY AUTOINCREMENT, " +
COLUNA_ITEM_EMPRESTIMOS
+ " TEXT NOT NULL," + COLUNA_DESCRICAO_EMPRESTIMOS
+ " TEXT NOT NULL," + COLUNA_STATUS_EMPRESTIMOS
+ " INTEGER NOT NULL," +
COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS
+ " INTEGER NOT NULL," +
COLUNA_STATUS_ALARME_EMPRESTIMOS
+ " INTEGER NOT NULL," +
COLUNA_ID_CATEGORIA_EMPRESTIMOS
+ " INTEGER NOT NULL," +
COLUNA_ID_CONTATO_EMPRESTIMOS
+ " INTEGER," + COLUNA_CONTATO_EMPRESTIMOS + " TEXT, "
+ " FOREIGN KEY ( " +
COLUNA_ID_CATEGORIA_EMPRESTIMOS
+ " ) REFERENCES " + TABELA_CATEGORIA + " (" +
COLUNA_ID_CATEGORIA
+ " ) ON DELETE RESTRICT ON UPDATE CASCADE);";

```

```

private static final String TAG = "DbAdapter";

```

```

private DatabaseHelper mDbHelper;

```

```

private SQLiteDatabase mDb;

```

```

private static final String DB_NAME = "ABP";

```

```

private static final int DATABASE_VERSION = 1;

```

```

private final Context mCtx;

```

O código acima é a declaração do nome das tabelas e suas colunas. Depois essas Strings constantes e estáticas são utilizadas para definir as Strings de criação das tabelas do banco. Existe um certo trabalho fazer isso, mas há um ganho enorme em vários sentidos, antes de seguir em frente vamos analisar o que fizemos. Vamos pensar nos problemas que teríamos ao tentar criar uma tabela desta maneira:

//Trecho de código retirado do tutorial do NotePad em
<http://developer.android.com/resources/tutorials/notepad/index.html>

```
public static final String KEY_TITLE = "title";
public static final String KEY_BODY = "body";
public static final String KEY_ROWID = "_id";
/**
 * Database creation sql statement
 */
private static final String DATABASE_CREATE =
    "create table notes (_id integer primary key autoincrement, "
    + "title text not null, body text not null);";

private static final String DATABASE_NAME = "data";
private static final String DATABASE_TABLE = "notes";
```

O trecho acima foi retirado de um dos tutoriais oficiais do Developer Android.

A abordagem acima sugere que sejam declaradas Strings para definir constantes para cada coluna da tabela “notes”, mas na hora de definir a String de criação da tabela (DATABASE_CREATE) o texto foi integralmente escrito no código fonte e não utilizaram as constantes previamente criadas.

Essa atitude é aceitável em um tutorial, pois facilita a leitura do código e não existe manutenção do mesmo, mas em um projeto real seria melhor tomar algumas providências para facilitar o trabalho.

Em um projeto real temos várias tabelas, cada tabela com várias colunas. Você pode errar ao escrever o nome de pelo menos uma coluna e se isso ocorrer perderia horas debugando código até ver que trocou uma letra em um nome de coluna em algum lugar. Então, basicamente, a definição dos nomes de tabelas e colunas com a utilização de Strings constantes pode consumir algum tempo a principio, mas pode evitar que erros corriqueiros aconteçam.

O mencionado acima é uma das boas praticas de engenharia de software relacionadas ao gerenciamento de banco no Android, porém há muito o que melhorar ainda. Neste tutorial o gerenciamento do banco, a definição das tabelas e as consultas ficarão no mesmo arquivo, porem se projeto possuir umas 5 ou 6 tabelas, esse arquivo poderia alcaçar umas mil linhas de código que incluiriam consultas e constantes, o que acabaria por dificultar seu manuseio, falaremos sobre isso mais tarde.

O próximo passo é estender a classe SQLiteOpenHelper e sobrescrever alguns métodos, como o `onOpen(SQLiteDatabase db)`, `onCreate(SQLiteDatabase db)` e o `onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)`.

O código abaixo faz isso:

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    @Override
    public void onOpen(SQLiteDatabase db)
    {
        super.onOpen(db);
        if (!db.isReadOnly())
        {
            db.execSQL("PRAGMA foreign_keys=ON;");
        }
    }
}
```

```
}
```

```
DatabaseHelper(Context context) {  
    super(context, DB_NAME, null, DATABASE_VERSION);  
}
```

```
@Override
```

```
public void onCreate(SQLiteDatabase db) {
```

```
    db.execSQL(CATEGORIA_CREATE_TABLE);  
    db.execSQL(EMPRESTIMOS_CREATE_TABLE);
```

```
    //Insere valores iniciais na tabela CATEGORIA
```

```
    ContentValues values = new ContentValues();  
    values.put(COLUNA_DESCRICAO_CATEGORIA, "Todos");  
    db.insert(TABELA_CATEGORIA, null, values);  
    values.clear();  
    values.put(COLUNA_DESCRICAO_CATEGORIA, "CDs");  
    db.insert(TABELA_CATEGORIA, null, values);  
    values.clear();  
    values.put(COLUNA_DESCRICAO_CATEGORIA, "Livros");  
    db.insert(TABELA_CATEGORIA, null, values);
```

```
    Log.w("DbAdapter", "DB criado com sucesso!");
```

```
}
```

```
@Override
```

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {
```

```
    Log.w(TAG, "Atualizando o banco de dados da versão " +  
oldVersion
```

```
        + " para " + newVersion
```

```
        + ", todos os dados serão perdidos!");
```

```

        db.execSQL("DROP TABLE IF EXISTS " +
TABELA_CATEGORIA);
        db.execSQL("DROP TABLE IF EXISTS " +
TABELA_EMPRESTIMOS);
        onCreate(db);
    }
}

```

Note que esta é uma classe private de nome DatabaseHelper.

onOpen(SQLiteDatabase db)

É executado toda vez que se cria um objeto dessa classe e o nosso faz com que a utilização de chaves estrangeiras seja ativada para o banco. Até o Android 2.1 a utilização de chaves estrangeiras não era possível, mas mesmo se você executar isso em uma versão sem suporte não haverá problemas.

DatabaseHelper.onCreate

É executado somente na primeira vez que a classe DbAdapter é instanciada. Esse método recebe como parâmetro um objeto do tipo SQLiteDatabase que é o verdadeiro objeto de acesso ao banco. Este objeto é utilizado para executar a criação das tabelas TABELA_CATEGORIA e TABELA_EMPRESTIMOS, depois disso é utilizado para inserir alguns valores padrão na tabela TABELA_CATEGORIA.

DatabaseHelper.onUpgrade

Também recebe um SQLiteDatabase como parâmetro, mas muitas vezes o que nos interessa nesse método são os outros dois parâmetros oldVersion e newVersion.

Eles nos informam o valor da versão antiga e da nova versão do banco. Quando estamos atualizando um aplicativo de versão, caso haja uma nova versão do banco definida, o método `DatabaseHelper.onUpgrade` será executado e podemos nos basear no número da versão para escolher quais alterações aplicar nas tabelas do banco, por exemplo um `ALTER TABLE` de uma versão para outra.

Falando em versão de banco, você reparou que o construtor da classe também foi sobrescrito?

```
private static class DatabaseHelper extends SQLiteOpenHelper {

    DatabaseHelper(Context context) {
        super(context, DB_NAME, null, DATABASE_VERSION);
    }

    ...
}
```

Esse construtor é obrigado a executar o construtor da classe mãe através do `super` e passar alguns parâmetros, o `DB_NAME` e o `DATABASE_VERSION`, respectivamente, o nome do banco e sua versão atual. Este último valor deve ser mudado a cada vez que o banco for editado, fazendo assim que o método `DatabaseHelper.onUpgrade` seja executado. O valor dessas constantes foram declaradas junto com as que definem as tabelas, colunas e uma instância da classe que acabamos de definir.

```
private static final String TAG = "DbAdapter";
private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;

private static final String DB_NAME = "ABP";
private static final int DATABASE_VERSION = 1;

private final Context mContext;
```

Temos a classe DbAdapter com as definições das tabelas e colunas, dentro dela temos a classe private DatabaseHelper que é a responsável por criar, atualizar e nos dar um meio de acesso ao mesmo.

Agora temos que definir os métodos de DbAdapter que encapsulam o acesso ao banco.

Abaixo temos o construtor de DbAdapter que recebe como parâmetro um Context, o método DbAdapter.open e o DbAdapter.close. É de vital importância para seu programa que para cada open executado seja também executado um close, assim como em qualquer acesso a banco de dados relacionado de qualquer outra plataforma:

```
public DbAdapter(Context ctx) {
    this.mCtx = ctx;
}

public DbAdapter open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
    return this;
}

public void close() {
    mDbHelper.close();
    mDb.close();
}
```

O método DbAdapter.open é o responsável por instanciar a mDbHelper e por adquirir acesso com permissão de escrita ao banco (leitura também). Veja que mDb também é instanciado e fechado junto de mDbHelper.

Pronto, de agora em diante não precisamos mais de utilizar o objeto `mDbHelper`, somente o `mDb` é o suficiente para inserir, atualizar, remover e buscar informações nas tabelas através dos métodos `insert`, `update`, `delete` e `query`. Vamos a alguns exemplos e explicações de cada método.

```
public long criarCategoria(String descricao) {  
    ContentValues values = new ContentValues();  
    values.put(COLUNA_DESCRICAO_CATEGORIA, descricao);  
  
    return mDb.insert(TABELA_CATEGORIA, null, values);  
}
```

Método chamado quando se deseja inserir uma categoria no banco, como `ID_CATEGORIA` possui auto incremento somente precisamos passar sua descrição como parâmetro do método. Como regra padrão do Android sempre inserimos um `ContentValues` no banco, esse objeto funciona como um `HashMap`, com chaves únicas e valores, so que há uma diferença. As Chaves são sempre `Strings`, no caso, as colunas da tabela.

E além disso, os valores podem ser de qualquer tipo primitivo “envelopado” do Java, no mesmo `ContentValues` podemos inserir tanto `Strings` quanto `Integers`, `Longs` ou `Doubles`. Desse jeito, no método acima estamos dizendo: “Insira na tabela `TABELA_CATEGORIA` uma linha, sendo que na coluna `COLUNA_DESCRICAO_CATEGORIA` haverá o valor da `String` `descricao`”.

```
public boolean removerCategoria(long idCategoria) {  
  
    return mDb.delete(TABELA_CATEGORIA,  
COLUNA_ID_CATEGORIA + "=?",  
        new String[] { String.valueOf(idCategoria) }) > 0;  
}
```


O método delete recebe como parâmetros o nome da tabela, uma String com o nome da coluna que será utilizada no WHERE concatenada com a condição, no caso o “=” e uma interrogação. A interrogação vai ser substituída pelo primeiro valor do array de String que é passado como último parâmetro para o método.

Estamos dizendo: “Remova uma linha da tabela TABELA_CATEGORIA onde COLUNA_ID_CATEGORIA for igual a idCategoria”.

Caso fosse preciso passar dois parâmetros para o WHERE ao invés de somente um, poderíamos ter feito como no exemplo abaixo:

```
public boolean removerCategoria(long idCategoria, String descricao) {  
  
    return                                mDb.delete(TABELA_CATEGORIA,  
COLUNA_ID_CATEGORIA + "=?" AND "  
    + COLUNA_DESCRICAO_CATEGORIA + "=?",  
    new String[] { String.valueOf(idCategoria), descricao }) > 0;  
}
```

Estamos dizendo: “Remova as linhas da tabela TABELA_CATEGORIA onde coluna COLUNA_ID_CATEGORIA for igual a idCategoria e coluna COLUNA_DESCRICAO_CATEGORIA seja igual a descricao”.

Se for preciso ter acesso a todas as linhas guardadas na tabela TABELA_CATEGORIA, o método consultarTodasCategorias é o que você precisa.

```
public Cursor consultarTodasCategorias() {  
  
    return    mDb.query(TABELA_CATEGORIA,    new    String[]    {  
COLUNA_ID_CATEGORIA,
```

```
COLUNA_DESCRICAO_CATEGORIA }, null, null, null, null,  
null);  
}
```

O método query é o responsável por fazer SELECTs no banco.

A consulta acima seria algo como :

```
SELECT  
COLUNA_ID_CATEGORIA,COLUNA_DESCRICAO_CATEGORIA      FROM  
TABELA_CATEGORIA;
```

Não se preocupe com o objeto Cursor retornado por esse método por enquanto, ele será explicado em breve.

Quando for necessário retornar somente uma categoria, utilizamos o método consultarCategoria:

```
public Cursor consultarCategoria(long idCategoria) throws SQLException {  
  
    Cursor mCursor = mDb.query(true, TABELA_CATEGORIA, new  
String[] { COLUNA_ID_CATEGORIA, COLUNA_DESCRICAO_CATEGORIA },  
COLUNA_ID_CATEGORIA + "=?",  
    new String[] { String.valueOf(idCategoria) }, null, null, null,  
    null);  
    if (mCursor != null) {  
        mCursor.moveToFirst();  
    }  
    return mCursor;  
}
```

Esse método também utiliza o método query, mas dessa vez passamos mais dois parâmetros para esse método. Sendo esses campos os mesmos que utilizamos no método delete, eles foram o WHERE da consulta.

A consulta acima seria algo como :

```
SELECT
COLUNA_ID_CATEGORIA,COLUNA_DESCRICAO_CATEGORIA      FROM
TABELA_CATEGORIA WHERE COLUNA_ID_CATEGORIA=idCategoria;
```

Para atualizar o conteúdo de uma linha na tabela TABELA_CATEGORIA utilizamos um ContentValues como parâmetro para o método update, bem como cláusulas para o WHERE.

```
public boolean atualizarCategoria(long idCategoria, String descricao) {
    ContentValues values = new ContentValues();
    values.put(COLUNA_DESCRICAO_CATEGORIA, descricao);

    return      mDb.update(TABELA_CATEGORIA,      values,
COLUNA_ID_CATEGORIA + "=?",
        new String[] { String.valueOf(idCategoria) }) > 0;
}
```

A leitura desse trecho fica assim:

“Atualize a linha idCategoria da coluna COLUNA_ID_CATEGORIA com os valores values na tabela TABELA_CATEGORIA”.

Agora um exemplo simples de acesso a uma categoria através de um Cursor.

```
public class EmprestaAiActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

setContentView(R.layout.main);
DbAdapter db = new DbAdapter(getApplicationContext());
db.open();

Long idCategoria = 1L;
String descricao = "";
Cursor c = db.consultarCategoria(idCategoria);
if(c != null){
    descricao =
c.getString(c.getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA));
}

db.close();
c.close();

Log.w("EmprestaAiActivity","Mostrando categoria: "+descricao);
}
}

```

Em uma Activity, instanciamos um DbAdapter, damos um open nele, executamos uma busca por um dado id de categoria, utilizamos o Cursor retornado para, dentro dele, recuperar o valor de uma coluna.

O método `Cursor.getString` recebe como parâmetro um inteiro que representa o index de uma coluna, o método `Cursor.getColumnIndex` nos retorna isso, mas pede como parâmetro o nome da coluna, que é exatamente o que está guardado em `DbAdapter.COLUNA_DESCRICAO_CATEGORIA`.

Depois de executar a consulta e recuperar a String de descrição, fechamos o banco e o Cursor (sim, o cursor também tem que ser fechado.), por fim imprimimos no LogCat a descrição da categoria.

Vamos aos métodos de gerencia da tabela TABELA_EMPRESTIMOS.

Mas primeiro adicione as seguintes constantes no início do arquivo DbAdapter.java

```
public static final int STATUS_EMPRESTAR = 0;
public static final int STATUS_PEGAR_EMPRESTADO = 1;

public static final int STATUS_ALARME_LIGADO = 0;
public static final int STATUS_ALARME_DESLIGADO = 1;
```

Estas constantes irão ajudar a mapear variáveis boolean no banco.

Vamos dar conferlr o código completo e depois analisar cada método separadamente:

```
public long criarEmprestimo(String item, String descricao,
    boolean statusEmprestimo, Date devolucao, boolean statusAlarme,
    long idCategoria, long idContato, String contato) {

    ContentValues values = new ContentValues();

    values.put(COLUNA_ITEM_EMPRESTIMOS, item);
    values.put(COLUNA_DESCRICAO_EMPRESTIMOS, descricao);
    values.put(COLUNA_STATUS_EMPRESTIMOS,
        statusEmprestimo ? STATUS_EMPRESTAR :
STATUS_PEGAR_EMPRESTADO);
    values.put(COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
devolucao.getTime());
    values.put(COLUNA_STATUS_ALARME_EMPRESTIMOS,
        statusAlarme ? STATUS_ALARME_LIGADO :
STATUS_ALARME_DESLIGADO);
```

```

        values.put(COLUNA_ID_CATEGORIA_EMPRESTIMOS,
idCategoria);
        values.put(COLUNA_ID_CONTATO_EMPRESTIMOS, idContato);
        values.put(COLUNA_CONTATO_EMPRESTIMOS, contato);

        return mDb.insert(TABELA_EMPRESTIMOS, null, values);
    }

    public Date getDataEmprestimo(long idEmprestimo){

        Cursor c = consultarEmprestimo(idEmprestimo);

        Long mili = 0L;
        mili =
c.getLong(c.getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMP
RESTIMOS));
        c.close();

        return new Date(mili);
    }

    public Cursor consultarEmprestimosBetween(Date inicio, Date fim) {
        Cursor mCursor =

        mDb.query(
            true,
            TABELA_EMPRESTIMOS,
            null,
            COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS + ">=? AND "
            + COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS + "<=?",
            new String[] { String.valueOf(inicio.getTime()),
                String.valueOf(fim.getTime()) }, null, null, null, null);
        if (mCursor != null) {
            mCursor.moveToFirst();

```

```

    }
    return mCursor;

}

public boolean removerEmprestimo(long idEmprestimo) {

    return mDb.delete(TABELA_EMPRESTIMOS,
        COLUNA_ID_EMPRESTIMOS + "=?",
        new String[] { String.valueOf(idEmprestimo) }) > 0;
}

// Retorna TUDO que estiver na tabela Emprestimos. Funciona, mas precisa
// mesmo passar todas as colunas ?
public Cursor consultarTodosEmprestimosV1() {

    return mDb.query(TABELA_EMPRESTIMOS, new String[] {
        COLUNA_ID_EMPRESTIMOS,
        COLUNA_ITEM_EMPRESTIMOS,
        COLUNA_DESCRICAO_EMPRESTIMOS,
        COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
        COLUNA_STATUS_EMPRESTIMOS,
        COLUNA_STATUS_ALARME_EMPRESTIMOS,
        COLUNA_ID_CATEGORIA_EMPRESTIMOS,
        COLUNA_ID_CONTATO_EMPRESTIMOS,
        COLUNA_CONTATO_EMPRESTIMOS }, null, null, null, null, null);
}

// Retorna TUDO que estiver na tabela Emprestimos, assim como o método V1,
// mas é bem mais simples.
public Cursor consultarTodosEmprestimosV2() {

    return mDb
        .query(TABELA_EMPRESTIMOS, null, null, null, null, null, null);
}

```

```
}
```

```
public Cursor consultarEmprestimo(long idEmprestimo) throws SQLException {
```

```
    Cursor mCursor =
```

```
        mDb.query(true, TABELA_EMPRESTIMOS, null, COLUNA_ID_EMPRESTIMOS + "=?",
```

```
            new String[] { String.valueOf(idEmprestimo) }, null, null, null, null);
```

```
        if (mCursor != null) {  
            mCursor.moveToFirst();  
        }
```

```
        return mCursor;
```

```
}
```

```
public Cursor consultarEmprestimosPorCategoria(long idCategoria)  
    throws SQLException {
```

```
    Cursor mCursor =
```

```
        mDb.query(true, TABELA_EMPRESTIMOS, null, COLUNA_ID_CATEGORIA_EMPRESTIMOS + "=?",  
            new String[] { String.valueOf(idCategoria) }, null, null, null, null);
```

```
        if (mCursor != null) {  
            mCursor.moveToFirst();  
        }
```

```
        return mCursor;
```

```
}
```

```
public List<Date> consultarQuantosEmprestimosBetween(Date inicio, Date fim)  
{
```

```
    List<Date> datas = new ArrayList<Date>();
```



```

        Cursor mCursor = consultarEmprestimosBetween(inicio, fim);
        while (!mCursor.isAfterLast()) {
            Long mili = mCursor
                .getLong(mCursor

.getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS
));

            datas.add(new Date(mili));
            mCursor.moveToNext();
        }
        mCursor.close();

        return datas;
    }

```

```

public boolean atualizarEmprestimo(long idEmprestimo, String item,
    String descricao, boolean statusEmprestimo, Date devolucao,
    boolean statusAlarme, long idCategoria, long idContato,
    String contato) {

    ContentValues values = new ContentValues();

    values.put(COLUNA_ITEM_EMPRESTIMOS, item);
    values.put(COLUNA_DESCRICAO_EMPRESTIMOS, descricao);
    values.put(COLUNA_STATUS_EMPRESTIMOS,
        statusEmprestimo ? STATUS_EMPRESTAR
        : STATUS_PEGAR_EMPRESTADO);
    values.put(COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
        devolucao.getTime());
    values.put(COLUNA_STATUS_ALARME_EMPRESTIMOS,
        statusAlarme ? STATUS_ALARME_LIGADO
        : STATUS_ALARME_DESLIGADO);
}

```

```

        values.put(COLUNA_ID_CATEGORIA_EMPRESTIMOS,
idCategoria);
        values.put(COLUNA_ID_CONTATO_EMPRESTIMOS, idContato);
        values.put(COLUNA_CONTATO_EMPRESTIMOS, contato);

        return      mDb.update(TABELA_EMPRESTIMOS,      values,
COLUNA_ID_EMPRESTIMOS
        + "=?", new String[] { String.valueOf(idEmprestimo) }) > 0;
    }

    public long getNumeroDeEmprestimos() {
        return      DatabaseUtils.queryNumEntries(mDb,
TABELA_EMPRESTIMOS);
    }

```

Vamos aos métodos da tabela TABELA_EMPRESTIMOS e se você pulou o código acima e acha que simplesmente repeti o que fizemos com TABELA_CATEGORIA, em parte você está certo, mas algumas novas idéias e explicações serão mostradas aqui:

Por exemplo:

```

public long criarEmprestimo(String item, String descricao,
        boolean statusEmprestimo, Date devolucao, boolean statusAlarme,
        long idCategoria, long idContato, String contato) {

    ContentValues values = new ContentValues();

    values.put(COLUNA_ITEM_EMPRESTIMOS, item);
    values.put(COLUNA_DESCRICAO_EMPRESTIMOS, descricao);
    values.put(COLUNA_STATUS_EMPRESTIMOS,
        statusEmprestimo      ?      STATUS_EMPRESTAR      :
STATUS_PEGAR_EMPRESTADO);

```

```

        values.put(COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
devolucao.getTime());
        values.put(COLUNA_STATUS_ALARME_EMPRESTIMOS,
            statusAlarme ? STATUS_ALARME_LIGADO :
STATUS_ALARME_DESLIGADO);
        values.put(COLUNA_ID_CATEGORIA_EMPRESTIMOS,
idCategoria);
        values.put(COLUNA_ID_CONTATO_EMPRESTIMOS, idContato);
        values.put(COLUNA_CONTATO_EMPRESTIMOS, contato);

        return mDb.insert(TABELA_EMPRESTIMOS, null, values);
    }

```

Este método insere uma linha na tabela TABELA_EMPRESTIMOS, veja que o método ContentValues.put é sobrecarregado para todo tipo nativo do Java (int, boolean, float, String, etc), mas o SQLite não suporta booleanos e por isso fazemos um “casting” (conversão) para Integer e assim salvar as colunas COLUNA_STATUS_EMPRESTIMOS e COLUNA_STATUS_ALARME_EMPRESTIMOS.

```

        values.put(COLUNA_STATUS_EMPRESTIMOS,
            statusEmprestimo ? STATUS_EMPRESTAR :
STATUS_PEGAR_EMPRESTADO);
        values.put(COLUNA_STATUS_ALARME_EMPRESTIMOS,
            statusAlarme ? STATUS_ALARME_LIGADO :
STATUS_ALARME_DESLIGADO);

```

Se o valor contido em statusEmprestimo for true o valor de STATUS_EMPRESTAR será utilizado, se for false o valor de STATUS_PEGAR_EMPRESTADO. O mesmo acontece com statusAlarme, se seu valor for verdadeiro o valor de STATUS_ALARME_LIGADO será inserido no banco, caso contrário será utilizado o valor de STATUS_ALARME_DESLIGADO;

Ainda no método `DbAdapter.criarEmprestimo`, você reparou que ele recebe um `Date` como parâmetro?

O `SQLite` também não possui o tipo `Date` e por isso guardarei como `INTEGER` (poderia ter guardado como `TEXT`), logo logo você verá as vantagens dessa abordagem.

```
values.put(COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
devolucao.getTime());
```

Para guardar um `Date` como `INTEGER` utilizamos o método `Date.getTime` que nós retorna o número de milissegundos desde primeiro de Janeiro de 1970 as 00:00 GMT. Veja que um `INTEGER` (sempre que escrevo `INTEGER` com todas as letras em maiúsculo estou me referindo a um tipo do `SQLite`) do `SQLite` pode armazenar números bem grandes que no Java são representados por `Long` e isso vai ser importante na hora de recuperar esse campo.

```
public Date getDataEmprestimo(long idEmprestimo){

    Cursor c = consultarEmprestimo(idEmprestimo);
    Long mili = mCursor
        .getLong(mCursor
            .getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS
        ));
    c.close();
    return new Date(mili);
}
```

Utilizamos o método `DbAdapter.consultarEmprestimo` para recuperar um `Cursor` com base no id de um Empréstimo e desse cursor recuperamos um `Long` usando o método `Cursor.getLong`. Todo `Cursor` possui os métodos para recuperar cada um dos tipos nativos do Java que o `SQLite` tem suporte. Entenda, não guardamos um `Long` no `SQLite` e sim um `INTEGER`

que a API do Google para o objeto Cursor junto com o SQLite trata de nós retornar um Long no Java. É uma conversão implícita.

Resumo: Você guarda um Long (número de milissegundos) dentro de um INTEGER e recupera um Long.

O método `Cursor.getLong` pede como parâmetro o índice de uma coluna da tabela, para saber o índice de uma coluna utilizamos o valor de retorno de `Cursor.getColumnIndex` que por sua vez pede como parâmetro o nome de uma coluna, sendo o nome da coluna que requeremos retornar `COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS`. Utilizamos o número de milissegundos retornado para, no construtor de um `Date`, retornar informações de data que foram guardadas.

Uma das vantagens de guardar um `Date` como `INTEGER` é permitir que você possa emular um `BETWEEN` no banco, como o número de milissegundos desde Janeiro de 1970 nunca parou de aumentar (e espero que nunca pare apesar do fim do mundo ano em 2012) você pode fazer algo assim:

```
public Cursor consultarEmprestimosBetween(Date inicio, Date fim) {
    Cursor mCursor =

    mDb.query(
        true,
        TABELA_EMPRESTIMOS,
        null,
        COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS + ">=? AND "
            + COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS +
"<=?",
        new String[] { String.valueOf(inicio.getTime()),
            String.valueOf(fim.getTime()) }, null, null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
}
```

```
        return mCursor;
    }
}
```

Se tivéssemos guardado um TEXT que representa o Date, isso seria bem mais complicado, não acha?

Se você está estranhando o primeiro parâmetro para o método SQLiteDatabase.query ser um boolean, não fique. Este método é sobrecarregado e existem algumas chamadas possíveis. Este boolean está aí para dizer que queremos somente respostas não repetidas (DISTINCT). O segundo parâmetro é o nome da tabela, no caso TABELA_EMPRESTIMOS. O terceiro parâmetro deveria ser um Array de String com os nomes das colunas que você quer que sejam retornadas (PROJECTION), como estamos passando “null” estamos dizendo que queremos todas as colunas. O quarto parâmetro é uma String que represente as condições do WHERE, as “?” serão substituídas pelo conteúdo do quinto parâmetro que também deve ser um Array de String cujo tamanho deve ser o mesmo número “?” durante a consulta. Note ainda que os itens dentro do Array do quinto parâmetro devem estar na mesma ordem que devem ser substituídos no quarto parâmetro.

O SQL disso seria algo como:

```
SELECT  DISTINCT  *  FROM  TABELA_EMPRESTIMOS  WHERE
COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS>=inicio          AND
COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS<=fim;
```

```
public boolean removerEmprestimo(long idEmprestimo) {
    return      mDb.delete(TABELA_EMPRESTIMOS,
COLUNA_ID_EMPRESTIMOS + "=?",
        new String[] { String.valueOf(idEmprestimo) }) > 0;
}
```

Sem muita novidade por aqui, utilizamos o id de um Empréstimo para remover uma linha da tabela.

```

// Retorna TUDO que estiver na tabela Emprestimos. Funciona, mas precisa
// mesmo passar todas as colunas? Não.
public Cursor consultarTodosEmprestimosV1() {

    return mDb.query(TABELA_EMPRESTIMOS, new String[] {
        COLUNA_ID_EMPRESTIMOS,
COLUNA_ITEM_EMPRESTIMOS,
        COLUNA_DESCRICAO_EMPRESTIMOS,
        COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
COLUNA_STATUS_EMPRESTIMOS,
        COLUNA_STATUS_ALARME_EMPRESTIMOS,
        COLUNA_ID_CATEGORIA_EMPRESTIMOS,
COLUNA_ID_CONTATO_EMPRESTIMOS,
        COLUNA_CONTATO_EMPRESTIMOS }, null, null, null, null, null);
}

// Retorna TUDO que estiver na tabela Emprestimos, assim como o método V1,
// mas é bem mais simples.
public Cursor consultarTodosEmprestimosV2() {

    return mDb
        .query(TABELA_EMPRESTIMOS, null, null, null, null, null, null);
}

```

Os dois métodos acima fazem exatamente a mesma coisa, mas vamos enfatizar aqui. Se você passar o segundo parâmetro como “null” todas as colunas da tabela virão no SELECT, no caso dos métodos acima isso acontece.

Os métodos acima se assemelham com as consultas SQL abaixo:

```

SELECT COLUNA_ID_EMPRESTIMOS, COLUNA_ITEM_EMPRESTIMOS,

```

```
        COLUNA_DESCRICAO_EMPRESTIMOS,  
        COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,  
COLUNA_STATUS_EMPRESTIMOS,  
        COLUNA_STATUS_ALARME_EMPRESTIMOS,  
        COLUNA_ID_CATEGORIA_EMPRESTIMOS,  
COLUNA_ID_CONTATO_EMPRESTIMOS,  
        COLUNA_CONTATO_EMPRESTIMOS           FROM  
TABELA_EMPRESTIMOS;
```

e

```
SELECT * FROM TABELA_EMPRESTIMOS;
```

O problema dessa abordagem é que se você tiver uma tabela com muitas colunas e linhas o consumo de memória por parte do Cursor será bem grande.

Exemplo mais ou menos numérico:

Seja “campo” qualquer item de uma linha em qualquer coluna retornado por um SELECT.

Em uma tabela com 10 coluna e 100 entradas (linhas).

Se você só vai precisar de duas colunas e fizer uma consulta que retorne somente as duas colunas, você vai ter um cursor com 200 (2*100) “campos”.

Se você utilizar uma consulta genérica como as acima, você vai ter um Cursor com 1 mil (10*100) “campos”.

Ou seja, 5 vezes mais memória do que o necessário, logo se você quer que seu programa seja mais otimizado possível deve trabalhar essas consultas. Se seu programa não vai lidar com grandes quantidades de dados

não haverá muita diferença no desempenho tanto por parte de consumo de RAM quanto por parte de processamento, mas isso é uma análise que deve ser feita durante o desenvolvimento do projeto.

```
public Cursor consultarEmprestimo(long idEmprestimo) throws SQLException {
```

```
    Cursor mCursor =  
        mDb.query(true,          TABELA_EMPRESTIMOS,          null,  
COLUNA_ID_EMPRESTIMOS + "=?",  
        new String[] { String.valueOf(idEmprestimo) }, null, null,  
        null, null);  
    if (mCursor != null) {  
        mCursor.moveToFirst();  
    }  
    return mCursor;  
}
```

Utilizamos id de um Empréstimo para retornar um Cursor com uma linha do banco.

```
public Cursor consultarEmprestimosPorCategoria(long idCategoria)  
    throws SQLException {
```

```
    Cursor mCursor =  
  
    mDb.query(true, TABELA_EMPRESTIMOS, null,  
        COLUNA_ID_CATEGORIA_EMPRESTIMOS + "=?",  
        new String[] { String.valueOf(idCategoria) }, null, null, null,  
        null);  
    if (mCursor != null) {  
        mCursor.moveToFirst();  
    }  
    return mCursor;  
}
```

Retorna um Cursor com todas as linhas que estão na categoria de valor idCategoria. Veja que nesse caso movemos o Cursor para a primeira posição supondo que podemos ter mais de um Empréstimo por Categoria. Suponha que você queira saber quais as datas dos Empréstimos que foram feitos entre duas datas, poderíamos fazer assim:

```
public List<Date> consultarQuantosEmprestimosBetween(Date inicio, Date fim)
{
    List<Date> datas = new ArrayList<Date>();

    Cursor mCursor = consultarEmprestimosBetween(inicio, fim);
    while (!mCursor.isAfterLast()) {
        Long mili = mCursor
            .getLong(mCursor

                .getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMP
RESTIMOS));

        datas.add(new Date(mili));
        mCursor.moveToNext();
    }
    mCursor.close();

    return datas;
}
```

Ao invés de retornar um Cursor, estamos percorrendo o Cursor por meio do while adicionando um novo Date a lista de Datas a cada iteração e movendo o Cursor por meio do Cursor.moveToNext. Por fim, fechamos o Cursor quando ele não é mais necessário (!) e retornamos a lista de datas.

```
Cursor mCursor = consultarEmprestimosBetween(inicio, fim);
```

Estamos utilizando um método previamente definido para fazer o BETWEEN para nós. Você acaba de cair em uma armadilha, o método acima busca no banco todas as colunas e você pode cair no caso desse Cursor gastar mais memória que o necessário. Como já disse, cabe a você programador decidir se vai fazer um segundo método que faça a busca por somente a coluna que guarda a coluna desejada ou não.

```
public boolean atualizarEmprestimo(long idEmprestimo, String item,
    String descricao, boolean statusEmprestimo, Date devolucao,
    boolean statusAlarme, long idCategoria, long idContato,
    String contato) {

    ContentValues values = new ContentValues();

    values.put(COLUNA_ITEM_EMPRESTIMOS, item);
    values.put(COLUNA_DESCRICAO_EMPRESTIMOS, descricao);
    values.put(COLUNA_STATUS_EMPRESTIMOS,
        statusEmprestimo ? STATUS_EMPRESTAR :
STATUS_PEGAR_EMPRESTADO);
    values.put(COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS,
devolucao.getTime());
    values.put(COLUNA_STATUS_ALARME_EMPRESTIMOS,
        statusAlarme ? STATUS_ALARME_LIGADO :
STATUS_ALARME_DESLIGADO);
    values.put(COLUNA_ID_CATEGORIA_EMPRESTIMOS,
idCategoria);
    values.put(COLUNA_ID_CONTATO_EMPRESTIMOS, idContato);
    values.put(COLUNA_CONTATO_EMPRESTIMOS, contato);

    return mDb.update(TABELA_EMPRESTIMOS, values,
COLUNA_ID_EMPRESTIMOS
    + "=?", new String[] { String.valueOf(idEmprestimo) }) > 0;
}
```

O método `DbAdapter.atualizarEmprestimo` é muito parecido com o `DbAdapter.criarEmprestimo`, na verdade somente a última linha é diferente, ao invés do insert temos o update, este método já foi explicado.

```
public long getNumeroDeEmprestimos() {  
    return DatabaseUtils.queryNumEntries(mDb,  
TABELA_EMPRESTIMOS);  
}
```

O último método que mostraremos aqui é o `DbAdapter.getNumeroDeEmprestimos` que utiliza um método utilitário de `DatabaseUtils` para retornar o número de linhas em `TABELA_EMPRESTIMOS`.



Temos acima um item de lista é o conteúdo a ser exibido como um item de um ListView ou um Spinner, sendo que um item de lista pode ser um texto simples (um TextView) ou , por exemplo, dois TextView, um contendo o nome de item emprestado e outro uma descrição a respeito daquele item. Repare na figura acima, o Spinner utiliza como item de lista somente um TextView, mas no ListView cada item de lista é composto por dois TextView.

Agora explicar como utilizar um Cursor para popular ListViews e Spinners por meio do SimpleCursorAdapter e como gerenciar o ciclo de vida de Cursores dentro do ciclo de vida de uma Activity.

A classe EmpresteActivity é mostrada abaixo (abaixo dela temos uma análise por partes), ela acessa as tabelas TABELA_EMPRESTIMOS e TABELA_CATEGORIAS através de Cursores para preencher um Spinner e um ListView.

```
package org.android.brasil.projetos.emprestaai;

import java.util.Date;
import java.util.List;

import android.app.ListActivity;
import android.database.Cursor;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.AdapterView.OnItemSelectedListener;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.Spinner;
import android.widget.Toast;
```

```

public class EmprestaAiActivity extends ListActivity {
    private Spinner sp;
    private Cursor cursorCategoria;
    private Cursor cursorEmprestimos;
    private DbAdapter db;

    private OnItemClickListener lvListener = new OnItemClickListener() {

        @Override
        public void onItemClick(AdapterView<?> adapter, View view,
            int position, long idEmprestimo) {

            //Cursor aux = db.consultarEmprestimo(idEmprestimo);
            Cursor aux = (Cursor) adapter.getItemAtPosition(position);

            // Permite que o Android gerencie o Cursor para melhor
            desempenho.
            //startManagingCursor(aux);

            String text = "";
            String item = aux.getString(aux

            .getColumnIndex(DbAdapter.COLUNA_ITEM_EMPRESTIMOS));
            String descri = aux.getString(aux

            .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_EMPRESTIM
OS));

            Long milis = aux

            .getLong(aux

            .getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMP
RESTIMOS));
            Long idContato = aux.getLong(aux

```

```

        .getColumnIndex(DbAdapter.COLUNA_ID_CONTATO_EMPRESTI
MOS));

        Long idCategoria = aux.getLong(aux

        .getColumnIndex(DbAdapter.COLUNA_ID_CATEGORIA_EMPREST
IMOS));

        int status = aux.getInt(aux

        .getColumnIndex(DbAdapter.COLUNA_STATUS_EMPRESTIMOS));

        //stopCursor(aux);

        // Busca o nome da categoria no banco.
        aux = db.consultarCategoria(idCategoria);

        startManagingCursor(aux);

        String nomeCategoria = aux.getString(aux

        .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)
);

        text = "O item: " + item + " possui a descrição: " + descri
            + ".\n Ele foi adicionado na data "
            + (new Date(milis)).toString()
            + ", pertence a categoria de id: " + idCategoria + "("
            + nomeCategoria + ")";

        if (status == DbAdapter.STATUS_EMPRESTAR) {
            text = text + ", \n foi emprestado pelo contato de id: "
                + idContato + " ";
        } else {

```

```

        text = text + ", \n foi emprestado para o contato de id: "
            + idContato + " ";
    }

    Toast.makeText(getApplicationContext(),                text,
    Toast.LENGTH_LONG)
        .show();
    Log.w("EmprestaAiActivity", text);

    stopCursor(aux);

    }
};

private OnItemSelectedListener spListener = new OnItemSelectedListener() {

    @Override
    public void onItemSelected(AdapterView<?> adapter, View view,
        int position, long idCategoria) {

        String descricao = null;

        // Verifica qual o item selecionado usando baseando-se nas
        // informações do Cursor.
        //Cursor aux = db.consultarCategoria(idCategoria);

        Cursor aux = (Cursor) adapter.getItemAtPosition(position);

        //startManagingCursor(aux);

        if (aux != null) {
            descricao = aux.getString(aux

```



```

        .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)
    );
    }

    //stopCursor(aux);

    stopCursor(cursorEmprestimos);

    if ((descricao != null) && (descricao.equalsIgnoreCase("Todos"))) {

        // Busca todos!
        cursorEmprestimos = db.consultarTodosEmprestimosV2();

        // Busca todos empréstimos entre Janeiro de 1970 e a data
        // atual!
        //
        // cursorEmprestimos
        // =
        db.consultarEmprestimosBetween(new
            // Date(0), new Date());

        // Busca somente os empréstimos entre essas datas.
        // cursorEmprestimos = db.consultarEmprestimosBetween(
        // new Date(10000), new Date());

        // Busca todas datas de empréstimos entre Janeiro de 1970
        // e
        // a data atual!
        List<Date>
        // datas
        // =
        db.consultarQuantosEmprestimosBetween(
            new Date(0), new Date());

        // Imprime no LogCat todas as datas de empréstimos.
        for (Date d : datas) {
            Log.w("EmprestaAiActivity", d.toString());
        }
    }
}

```

```

        }
    } else {
        cursorEmprestimos = db
            .consultarEmprestimosPorCategoria(idCategoria);
    }

    startManagingCursor(cursorEmprestimos);

    SimpleCursorAdapter lvAdapter = new SimpleCursorAdapter(
        getApplicationContext(), R.layout.linha, cursorEmprestimos,
        new String[] { DbAdapter.COLUNA_ITEM_EMPRESTIMOS,
            DbAdapter.COLUNA_CONTATO_EMPRESTIMOS }, new
int[] {

        R.id.linha1, R.id.linha2 });

    setListAdapter(lvAdapter);

}

@Override
public void onNothingSelected(AdapterView<?> arg0) {
    // TODO Auto-generated method stub

}

};

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    sp = (Spinner) findViewById(R.id.sp_categorias);
    sp.setOnItemSelectedListener(spListener);
}

```

```

        ListView lv = getListView();
        lv.setOnItemClickListener(lvListener);
    }

    private void fillData() {

        db = new DbAdapter(getApplicationContext());
        db.open();

        stopCursor(cursorCategoria);

        cursorCategoria = db.consultarTodasCategorias();

        startManagingCursor(cursorCategoria);

        SimpleCursorAdapter spAdapter = new SimpleCursorAdapter(
            getApplicationContext(), R.layout.linha, cursorCategoria,
            new String[] { DbAdapter.COLUNA_DESCRICAO_CATEGORIA },
            new int[] { R.id.linha1 });

        if (sp != null) {
            sp.setAdapter(spAdapter);
        }

    }

    protected void onResume() {
        super.onResume();
        fillData();

        // Verificação para evitar ficar inserindo as linhas abaixo a cada
        // onCreate
        if (db.getNumeroDeEmprestimos() == 0) {

```

```
db.criarEmprestimo("Guia do Mochileiro das Galaxias", "Livro 1",
    true, new Date(100), true, 3, 0, "Julio");
```

```
db.criarEmprestimo("Guia do Mochileiro das Galaxias", "Livro 2",
    true, new Date(1000), true, 3, 0, "Gu");
```

```
db.criarEmprestimo("Crime e Castigo", "Um livro lá", true,
    new Date(100000), true, 3, 0, "Zê");
```

```
db.criarEmprestimo("Linkin Park", " Album novo", true, new Date(
    1000000), true, 2, 0, "Gilberto");
```

```
}
```

```
}
```

```
@Override
```

```
protected void onPause() {
    super.onPause();
```

```
    stopCursor(cursorCategoria);
    stopCursor(cursorEmprestimos);
```

```
    db.close();
```

```
}
```

```
private void stopCursor(Cursor c) {
    if (c != null && !c.isClosed()) {
        stopManagingCursor(c);
        c.close();
```

```
    }
```

```
}
```

```
}
```

No método onCreate (logo abaixo) instanciamos um Spinner e um ListView, também adicionamos Listeners para cada um, por enquanto nada relativo a DB.

@Override

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
  
    sp = (Spinner) findViewById(R.id.sp_categorias);  
    sp.setOnItemSelectedListener(spListener);  
  
    ListView lv = getListView();  
    lv.setOnItemClickListener(lvListener);  
}
```

O método stopCursor mostrado abaixo é uma mera conveniência para facilitar esta operação que deve ser executada algumas vezes durante a execução de uma aplicação.

```
private void stopCursor(Cursor c) {  
    if (c != null && !c.isClosed()) {  
        stopManagingCursor(c);  
        c.close();  
    }  
}
```

Por enquanto atenha-se a regra: Antes de abrir um Cursor, execute o método stopCursor e seu aplicativo vai funcionar em qualquer versão do Android.

Um exemplo de quando chamar stopCursor, quando a Activity atual está perdendo o foco, ou seja, está sendo pausada, destruída ou estamos iniciando uma nova Activity, temos que fechar os Cursores e o banco.

```
@Override
protected void onPause() {
    super.onPause();

    stopCursor(cursorCategoria);
    stopCursor(cursorEmprestimos);

    db.close();
}
```

Isso é natural, recursos de banco sempre tem que ser fechados quando não são mais necessários.

Como temos duas instâncias de Cursor, temos que fechar ambos e o banco também.

O método fillData, logo abaixo, só é chamado no método onResume e preste atenção neste ponto. O método onResume só é chamado quando a Activity é iniciada e depois quando a Activity já passou pelo onPause.

Na primeira vez que é executado o método onResume o objeto db é null e está tudo bem em instância-lo, quando a Activity é pausada o banco é fechado então quando a Activity volta a ser executada tudo bem chamar o fillData mais uma vez para pegar uma nova instância do banco. Esse tipo de comportamento, ter somente uma instância do banco e abrir e fechá-lo apropriadamente durante o ciclo de vida da sua aplicação, faz toda diferença com relação a estabilidade da sua aplicação.

```
private void fillData() {
```

```

db = new DbAdapter(getApplicationContext());
db.open();

stopCursor(cursorCategoria);

cursorCategoria = db.consultarTodasCategorias();

startManagingCursor(cursorCategoria);

SimpleCursorAdapter spAdapter = new SimpleCursorAdapter(
    getApplicationContext(), R.layout.linha, cursorCategoria,
    new String[] { DbAdapter.COLUNA_DESCRICAO_CATEGORIA },
    new int[] { R.id.linha1 });

if (sp != null) {
    sp.setAdapter(spAdapter);
}
}

```

Depois do banco aberto, consultamos um Cursor que contem todas as linhas da tabela TABELA_CATEGORIA.

O cursorCategoria é utilizado para montar um SimpleCursorAdapter que é um Adapter para o tipo Cursor que a própria API do Android fornece para o desenvolvedor. Para instanciar um SimpleCursorAdapter precisamos passar alguns parâmetros para ele.

O primeiro é um Context, fornecido pelo método Activity(getApplicationContext), O segundo parâmetro é uma referência para um layout, este layout será utilizado para montar uma linha do nosso item de lista, o terceiro parâmetro é o nosso Cursor com os dados. Os últimos dois

parâmetros fazem a ponte entre o conteúdo do Cursor e em qual View do layout será mostrado o conteúdo, no caso R.id.linha1 é um TextView.

Em outras palavras: O conteúdo da coluna COLUNA_DESCRICAO_CATEGORIA dentro de cursorCategoria será mostrado, linha por linha (da tabela e do ListView ou Spinner), na View R.id.linha1.

O SimpleCursorAdapter faz com que cursorCategoria funcione como se fosse uma lista e cada item dessa lista será mostrado em um item de lista.

```
protected void onResume() {
    super.onResume();

    fillData();

    // Verificação para evitar ficar inserindo as linhas abaixo a cada
    // onCreate
    if (db.getNumeroDeEmprestimos() == 0) {
        db.criarEmprestimo("Guia do Mochileiro das Galaxias", "Livro 1",
            true, new Date(100), true, 3, 0, "Julio");

        db.criarEmprestimo("Guia do Mochileiro das Galaxias", "Livro 2",
            true, new Date(1000), true, 3, 0, "Gu");

        db.criarEmprestimo("Crime e Castigo", "Um livro lá", true,
            new Date(100000), true, 3, 0, "Zê");

        db.criarEmprestimo("Linkin Park", " Album novo", true, new Date(
            1000000), true, 2, 0, "Gilberto");
    }
}
```


Com o banco ainda aberto (isso foi pelo pela chamada de fillData) consultamos quantas linhas existem na tabela TABELA_EMPRESTIMOS e caso ainda não existam nenhuma, inserimos algumas.

No onCreate temos as seguintes linhas:

```
sp.setOnItemSelectedListener(spListener);  
lv.setOnItemClickListener(lvListener);
```

O objeto lvListener é declarado e instanciado no início da classe e ele é o responsável por mostrar alguns dados do item que foi clicado na tela e no LogCat.

```
private OnItemClickListener lvListener = new OnItemClickListener() {
```

```
@Override
```

```
public void onItemClick(AdapterView<?> adapter, View view,  
    int position, long idEmprestimo) {
```

```
    //Cursor aux = db.consultarEmprestimo(idEmprestimo);  
    Cursor aux = (Cursor) adapter.getItemAtPosition(position);
```

```
    // Permite que o Android gerencie o Cursor para melhor  
    desempenho.
```

```
    //startManagingCursor(aux);
```

```
    String text = "";
```

```
    String item = aux.getString(aux  
        .getColumnIndex(DbAdapter.COLUNA_ITEM_EMPRESTIMOS));
```

```
    String descri = aux.getString(aux
```

```
        .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_EMPRESTIM  
OS));
```

```

        Long millis = aux
            .getLong(aux

                .getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMP
RESTIMOS));
        Long idContato = aux.getLong(aux

            .getColumnIndex(DbAdapter.COLUNA_ID_CONTATO_EMPRESTI
MOS));
        Long idCategoria = aux.getLong(aux

            .getColumnIndex(DbAdapter.COLUNA_ID_CATEGORIA_EMPREST
IMOS));

        int status = aux.getInt(aux

            .getColumnIndex(DbAdapter.COLUNA_STATUS_EMPRESTIMOS));

        //stopCursor(aux);

        // Busca o nome da categoria no banco.
        aux = db.consultarCategoria(idCategoria);

        startManagingCursor(aux);

        String nomeCategoria = aux.getString(aux

            .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)
);

        text = "O item: " + item + " possui a descrição: " + descri
            + ".\n Ele foi adicionado na data "
            + (new Date(millis)).toString()
            + ", pertence a categoria de id: " + idCategoria + "("

```

```

        + nomeCategoria + "));

if (status == DbAdapter.STATUS_EMPRESTAR) {
    text = text + ", \n foi emprestado pelo contato de id: "
        + idContato + " ";
} else {
    text = text + ", \n foi emprestado para o contato de id: "
        + idContato + " ";
}

Toast.makeText(getApplicationContext(),                text,
Toast.LENGTH_LONG).show();
Log.w("EmprestaAiActivity", text);

stopCursor(aux);

}
};

```

O primeiro parâmetro para o método `onItemClickListener.onItemClick` é um `Adapter`, na verdade, nesse caso, será o `SimpleCursorAdapter` utilizado para preencher o `ListView` na declaração de `spListener`. Utilizado junto do segundo parâmetro, a posição na lista, podemos pegar do `Adapter` o item que está naquela posição. Como já disse, utilizaremos um `Cursor` para preencher um `SimpleCursorAdapter`, logo um item desse `Adapter` seria um item do `Cursor`.

```

// Jeito feio de pegar o Empréstimo.
//Cursor aux = db.consultarEmpréstimo(idEmpréstimo);

// Jeito Esperto.
Cursor aux = (Cursor) adapter.getItemAtPosition(position);

```

Então se podemos pegar as informações que precisamos diretamente de um Cursor que já em memória, não preciso consultar o banco novamente. O método `Adapter.getItemAtPosition` retorna um `Object`, fazemos um Casting para `Cursor` sabendo que ali realmente existe um `Cursor` e pegamos as informações que precisamos sobre aquele item. Note que o `Cursor` retornado já está na posição do item clicado.

A linha:

```
//Cursor aux = db.consultarEmprestimo(idEmprestimo);
```

Está comentada por que não precisamos fazer um novo acesso ao banco para pegar o item que foi clicado, as informações que queremos já estão disponíveis. Mais para frente nesse código existem outras linhas comentadas, se você fosse utilizar essa nova consulta ao banco, as próximas linhas também deveriam ser descomentadas, por exemplo:

```
// Permite que o Android gerencie o Cursor para melhor desempenho.
```

```
//startManagingCursor(aux);
```

O método `Activity.startManagingCursor` é responsável por ajudar na gerencia de quanta memória seu `Cursor` vai consumir é sempre bom permitir que o Android faça essa mágica para a gente.

Esta linha está comentada pelo seguinte motivo. O `Cursor` que estamos utilizando já está sendo gerenciado. Em um `fillData` eu já chamei `Activity.startManagingCursor` para ele. Lembre-se em Java objetos são passados como referencia, `aux` é uma variável de referencia, o objeto para o qual ela aponta, já está sendo gerenciado, então mesmo que eu tenha executado um `Adapter.getItemAtPosition` o `Cursor` retornado ainda é o mesmo que eu passei para o `Adapter`.

Em:

```
String text = "";
String item = aux.getString(aux
    .getColumnIndex(DbAdapter.COLUNA_ITEM_EMPRESTIMOS));
String descri = aux.getString(aux
    .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_EMPRESTIMOS));
Long milis = aux
    .getLong(aux
        .getColumnIndex(DbAdapter.COLUNA_DATA_DEVOLUCAO_EMPRESTIMOS
    ));
Long idContato = aux.getLong(aux
    .getColumnIndex(DbAdapter.COLUNA_ID_CONTATO_EMPRESTIMOS));
Long idCategoria = aux.getLong(aux
    .getColumnIndex(DbAdapter.COLUNA_ID_CATEGORIA_EMPRESTIMOS));
int status = aux.getInt(aux
    .getColumnIndex(DbAdapter.COLUNA_STATUS_EMPRESTIMOS));
```

Estamos simplesmente recuperando as informações sobre um Empréstimo a partir de um Cursor.

Se eu tivesse feito uma nova consulta para recuperar as informações sobre Empréstimo, eu deveria ter gerenciado essa nova instância de Cursor e claro também deveria ter parado de gerenciar através do nosso método utilitário stopCursor, como não fiz isso, posso simplesmente dizer para aux apontar para uma referencia de Cursor que acaba de ser criado.

```
//stopCursor(aux);

// Busca o nome da categoria no banco.
aux = db.consultarCategoria(idCategoria);

startManagingCursor(aux);
```

Esse novo Cursor tem que ser gerenciado, por isso usamos `Activity.startManagingCursor`.

Com `aux` apontando para uma nova instância recuperamos o nome da Categoria e montamos a String que será mostrada na tela e no LogCat. Como `aux` não será mais necessário depois dessa operação ele é fechado.

```
String nomeCategoria = aux.getString(aux

.getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)

);

text = "O item: " + item + " possui a descrição: " + descri
      + ".\n Ele foi adicionado na data "
      + (new Date(milis)).toString()
      + ", pertence a categoria de id: " + idCategoria + "("
      + nomeCategoria + ")";

if (status == DbAdapter.STATUS_EMPRESTAR) {
    text = text + ", \n foi emprestado pelo contato de id: "
          + idContato + " ";
} else {
    text = text + ", \n foi emprestado para o contato de id: "
          + idContato + " ";
}

Toast.makeText(getApplicationContext(),                text,
Toast.LENGTH_LONG)
        .show();
Log.w("EmprestaAiActivity", text);

stopCursor(aux);
```

Veja a linha abaixo retirada da declaração de `lvListener`:

```
public void onItemClickListener(AdapterView<?> adapter, View view,  
    int position, long idEmprestimo) {
```

Veja que o método `onItemClickListener.onItemClickListener` “magicamente” sabe qual o id do Empréstimo selecionado. Quando criamos a tabela `TABELA_EMPRESTIMOS` declaramos como `PRIMARY KEY` uma coluna de nome `COLUNA_ID_EMPRESTIMOS` e valor “_id”, ou seja, no SQLite o nome da coluna é “_id”.

// trecho de código da parte 1.

// <http://www.androidbrasilprojetos.org/android/introducao-a-banco-de-dados-android/>

```
public static final String COLUNA_ID_EMPRESTIMOS = "_id";
```

Depois quando foi feito o `SELECT` de quais dados seriam utilizados para preencher o `Cursor` que guarda o conteúdo do nosso `ListView` buscamos todas as colunas, incluindo a coluna “_id”.

O `SimpleCursorAdapter` e também o método `onItemClickListener.onItemClickListener` somente funcionam por causa desse “_id”. Quando o método `onItemClickListener.onItemClickListener` é executado, o `SimpleCursorAdapter` busca no `Cursor` uma coluna de nome “_id” e coloca seu valor no último parâmetro.

Veja que isso é muito vantajoso para os desenvolvedores, mas tem que saber utilizar, então como regra básica ao buscar um `Cursor`, sempre busque junto a coluna “_id”, sendo claro, que essa deve ser a `PRIMARY KEY` da sua tabela.

Nem toda tabela no banco é passível de ter somente uma chave simples e `INTEGER`, nesses casos você não deve utilizar o `SimpleCursorAdapter`. Utilize, por exemplo, um `ArrayList` e crie um `ArrayAdapter`.

O objeto spListener também é declarado e instanciado no início da classe.

O comportamento do objeto spListener é análogo ao comportamento de lvListener quanto ao gerenciamento do banco.

```
private OnItemSelectedListener spListener = new OnItemSelectedListener() {

@Override
public void onItemSelected(AdapterView<?> adapter, View view,
    int position, long idCategoria) {

    String descricao = null;

    // Verifica qual o item selecionado usando baseando-se nas
    // informações do Cursor.
    //Cursor aux = db.consultarCategoria(idCategoria);

    Cursor aux = (Cursor) adapter.getItemAtPosition(position);

    //startManagingCursor(aux);

    if (aux != null) {
        descricao = aux.getString(aux

.getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)
);
    }

    //stopCursor(aux);

    stopCursor(cursorEmprestimos);

    if ((descricao != null) && (descricao.equalsIgnoreCase("Todos"))) {
```



```

// Busca todos!
cursorEmprestimos = db.consultarTodosEmprestimosV2();

// Busca todos emprestimos entre Janeiro de 1970 e a data
// atual!
// cursorEmprestimos = db.consultarEmprestimosBetween(new
// Date(0), new Date());

// Busca somente os emprestimos entre essas datas.
// cursorEmprestimos = db.consultarEmprestimosBetween(
// new Date(10000), new Date());

// Busca todas datas de emprestimos entre Janeiro de 1970 e
// a data atual!
List<Date> datas = db.consultarQuantosEmprestimosBetween(
    new Date(0), new Date());

// Imprime no LogCat todas as datas de emprestimos.
for (Date d : datas) {
    Log.w("EmprestaAiActivity", d.toString());
}
} else {
    cursorEmprestimos = db
        .consultarEmprestimosPorCategoria(idCategoria);
}

startManagingCursor(cursorEmprestimos);

SimpleCursorAdapter lvAdapter = new SimpleCursorAdapter(
    getApplicationContext(), R.layout.linha, cursorEmprestimos,
    new String[] { DbAdapter.COLUNA_ITEM_EMPRESTIMOS,
        DbAdapter.COLUNA_CONTATO_EMPRESTIMOS }, new int[] {
        R.id.linha1, R.id.linha2 });

```

```

        setListAdapter(lvAdapter);

    }

    @Override
    public void onNothingSelected(AdapterView<?> arg0) {
        // TODO Auto-generated method stub

    }

};

```

Também consultamos o Adapter para recuperar um Cursor. Do Cursor recuperamos o nome da Categoria.

```

        // Verifica qual o item selecionado usando baseando-se nas
        // informações do Cursor.
        //Cursor aux = db.consultarCategoria(idCategoria);

        Cursor aux = (Cursor) adapter.getItemAtPosition(position);

        //startManagingCursor(aux);

        if (aux != null) {
            descricao = aux.getString(aux

            .getColumnIndex(DbAdapter.COLUNA_DESCRICAO_CATEGORIA)
        );

    }

    //stopCursor(aux);

    stopCursor(cursorEmprestimos);

```

Se a categoria selecionada for “Todos“, buscamos todos os Empréstimos no banco, caso contrário buscamos somente os Empréstimos de uma dada categoria.

```
if ((descricao != null) && (descricao.equalsIgnoreCase("Todos"))) {

    // Busca todos!
    cursorEmprestimos = db.consultarTodosEmprestimosV2();

    // Busca todos empréstimos entre Janeiro de 1970 e a data
    // atual!
    // cursorEmprestimos = db.consultarEmprestimosBetween(new
    // Date(0), new Date());

    // Busca somente os empréstimos entre essas datas.
    // cursorEmprestimos = db.consultarEmprestimosBetween(
    // new Date(10000), new Date());

    // Busca todas datas de empréstimos entre Janeiro de 1970 e
    // a data atual!
    List<Date> datas = db.consultarQuantosEmprestimosBetween(
        new Date(0), new Date());

    // Imprime no LogCat todas as datas de empréstimos.
    for (Date d : datas) {
        Log.w("EmprestaAiActivity", d.toString());
    }
} else {
    cursorEmprestimos = db
        .consultarEmprestimosPorCategoria(idCategoria);
}
```

Os comentários acima podem ser utilizados para testar o nosso emulador de BETWEEN para o campo data de devolução, buscando somente os Empréstimos que tem a data de devolução entre as datas passadas. Também imprimimos no LogCat a nossa lista de Date retornada pelo método DbAdapter.consultarQuantosEmprestimosBetween.

Com o Cursor devidamente alocado no if ou no else iniciamos o gerenciamento do Cursor e criamos um SimpleCursorAdapter para popular nosso ListView.

```
startManagingCursor(cursorEmprestimos);
```

```
SimpleCursorAdapter lvAdapter = new SimpleCursorAdapter(  
    getApplicationContext(), R.layout.linha, cursorEmprestimos,  
    new String[] { DbAdapter.COLUNA_ITEM_EMPRESTIMOS,  
        DbAdapter.COLUNA_CONTATO_EMPRESTIMOS }, new int[] {  
        R.id.linha1, R.id.linha2 });
```

```
setListAdapter(lvAdapter);
```

O layout R.id.linha na verdade contem dois TextView de identificadores R.id.linha1 e R.id.linha2 e serão utilizados para mostrar o conteúdo das colunas DbAdapter.COLUNA_ITEM_EMPRESTIMOS e DbAdapter.COLUNA_CONTATO_EMPRESTIMOS, respectivamente.

```
<?xml version="1.0" encoding="utf-8"?>
```

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

```
<TextView
```

```
    android:id="@+id/linha1"  
    android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content" />
```

```
<TextView
```

```
android:id="@+id/linha2"
```

```
android:padding="5sp"
```

```
android:layout_width="fill_parent"
```

```
android:layout_height="wrap_content" />
```

```
</LinearLayout>
```

Depois de instanciado o `IvAdapter` é utilizado como parâmetro para o método `setListAdapter` e assim o `ListView` é preenchido novamente toda vez que escolhemos um novo item no `Spinner`.

Explicação sobre `stopCursor` e `startManagingCursor`.

Antigamente, bastava que chamássemos `startManagingCursor` e o Android automaticamente cuidava de fazer exatamente o que as chamadas de `stopCursor` faz para nós, entretanto com a chegada do Android 3.0 o comportamento do sistema foi modificado e `startManagingCursor` e `stopManagingCursor` foram depreciados em favor da utilização de `Content Providers`. Como efeito colateral dessas mudanças todo código pre-HoneyComb que gerencia `Cursors` começou a dar pau e nos obrigar (até onde eu sei) a utilizar algo como o `stopCursor`.

Mídias em Android: Gráficos, áudio, vídeo e mapas

Animação no Android

O sistema de animação é propriedade de uma estrutura robusta que permite animar praticamente qualquer coisa. Pode-se definir uma animação para alterar qualquer propriedade do objeto ao longo do tempo, independentemente se ele chama a tela ou não.

O sistema de animação permite que se defina as seguintes características de uma animação:

Duração: pode-se especificar a duração de uma animação. O comprimento padrão é de 300 ms.

Interpolação tempo: pode-se especificar como os valores da propriedade são calculados em função do momento atual da animação decorrido.

Repetir contagem e comportamento: pode-se especificar se quer ou não ter uma repetição de animação quando se chega ao fim de uma duração e quantas vezes repetir a animação. Também pode-se especificar se deseja que a animação para reproduzir em sentido inverso. Defini-lo para reverter reproduz a animação para a frente depois para trás várias vezes, até que o número de repetições é atingido.

Animador : pode-se ter animações em conjuntos lógicos que jogam juntos ou sequencialmente ou após atrasos especificados.

Quadro de actualização: pode-se especificar a frequência para atualizar quadros de sua animação. O padrão é definido para atualizar a cada 10 ms, mas a velocidade em que seu aplicativo pode atualizar quadros, o que em última análise, depende de como o sistema está ocupado.

Visualizando uma animação

Podemos usar o sistema de animação vista para executar animação interpolada em exibições. Tween Animation calcula a animação com informações como o ponto de início, ponto final, tamanho, rotação, e outros aspectos comuns de uma animação.

A animação de interpolação pode executar uma série de transformações simples (posição, tamanho, rotação e transparência) sobre o conteúdo de um objeto View. Então, se temos um objeto TextView, pode-se mover, girar, aumentar ou reduzir o texto. Se ele tem uma imagem de fundo, a imagem de fundo será transformada junto com o texto. O pacote de animação fornece todas as classes usadas em uma animação de interpolação.

Uma sequência de instruções de animação define a animação, definido por XML ou código do Android. Tal como acontece com a definição de um layout, um arquivo XML é recomendado porque é mais legível, reutilizável e flexível que embutir a animação. No exemplo abaixo, usaremos XML.

As instruções de animação para definir as transformações que se deseja que ocorra, quando eles ocorrem, e quanto tempo eles devem esperar para aplicar. As transformações podem ser sequencial ou simultânea - por exemplo, podemos ter o conteúdo de um movimento TextView da esquerda para a direita, e depois girar 180 graus, ou podemos ter o movimento de texto e rodar simultaneamente. Cada transformação é um conjunto de parâmetros específicos para a transformação (tamanho inicial e final de tamanho para mudança de tamanho, o ângulo inicial e final para o ângulo de rotação, e assim por diante), e também um conjunto de parâmetros comuns (por exemplo, hora de início e duração) . Para fazer com que várias transformações ocorrem em simultâneo, dar-lhes a mesma hora de início, para torná-los sequencial, o cálculo do tempo de início mais o tempo de transformação anterior.

A animação arquivo XML pertence no diretório res/anim/ de seu projeto Android. O arquivo deve ter um único elemento raiz: esta será único <alpha>, <scale>, <translate>, <rotate>, interpolador de elemento, ou elemento <set>

que mantém grupos de esses elementos. Por padrão, todas as instruções de animação são aplicadas simultaneamente. Para ocorrerem em sequência, deve-se especificar o atributo `startOffset`.

O XML a seguir de um dos `ApiDemos` é usado para alongar, então, simultaneamente girar e girar um objeto `View`, exemplo 1.

Exibindo uma mídia

O Android framework multimídia inclui suporte para reprodução de diversos tipos de mídia comuns, de modo que podemos facilmente integrar vídeo, áudio e imagens em suas aplicações. Podemos reproduzir áudio ou vídeo a partir de arquivos de mídia armazenados em recursos do seu aplicativo (recursos-primos), a partir de arquivos autônomos no sistema de arquivos, ou a partir de um fluxo de dados que chega através de uma conexão de rede, todas as APIs usando `MediaPlayer`.

Mostraremos como escrever um aplicativo para exibir uma mídia que interage com o usuário e o sistema, a fim de obter um bom desempenho e uma experiência de usuário agradável.

Básico

As seguintes classes são utilizadas para reproduzir som e vídeo no sistema do Android:

`MediaPlayer`, esta classe é a API primária para reproduzir som e vídeo.

`AudioManager`, essa classe gerencia as fontes de áudio e saída de áudio em um dispositivo.

Declarações de manifesto

Antes de iniciar o desenvolvimento de um aplicativo usando MediaPlayer, verifique se o arquivo de manifesto tem as declarações apropriadas para permitir o uso de recursos relacionados.

Permissão Internet, se estivermos usando MediaPlayer para fluxo de rede baseada em conteúdo, o aplicativo deve solicitar acesso à rede.

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

Permissão de visualização, quando o aplicativo precisa apagar a tela ou o processador hibernar, usamos os métodos MediaPlayer.setScreenOnWhilePlaying() ou MediaPlayer.setWakeMode(), devemos solicitar essa permissão.

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

Usando MediaPlayer

Um dos componentes mais importantes do quadro de mídia é a classe MediaPlayer. Um objeto desta classe pode buscar, decodificar e reproduzir áudio e vídeo com configuração mínima. Ele suporta várias fontes diferentes de mídia, tais como: recursos locais, URLs internos, como um que se possa obter de um provedor de conteúdos e URLs externos (streaming).

Aqui está um exemplo de como reproduzir o áudio que está disponível como um recurso primo local (salvo na aplicação em res/raw/directory):

```
MediaPlayer mediaPlayer = MediaPlayer.create (contexto,  
R.raw.sound_file_1);  
mediaPlayer.start ();
```

E aqui está como executar uma mídia a partir de um URI disponível localmente no sistema:

```
Uri myUri = .... // inicializar Uri aqui
MediaPlayer mediaPlayer = new MediaPlayer ();
mediaPlayer.setAudioStreamType (AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource (getApplicationContext (), myUri);
mediaPlayer.prepare ();
MediaPlayer.start ();
```

Para se reproduzir a partir de uma URL remota via HTTP streaming:

```
String url = "http:// ....."; // sua URL aqui
MediaPlayer mediaPlayer = new MediaPlayer ();
mediaPlayer.setAudioStreamType (AudioManager.STREAM_MUSIC);
mediaPlayer.setDataSource (url);
mediaPlayer.prepare ()
MediaPlayer.start ();
```

Localização, Mapas e Mapas de Localização

Serviços de Localização

Aplicativos de localização baseado em mapas oferece uma experiência atraente em dispositivos móveis. Podemos construir esses recursos em um aplicativo usando as classes do pacote `android.location` e o Google Maps API Android.

Android permite o acesso de aplicações aos serviços de localização suportados pelo dispositivo através de classes do pacote `android.location`. O componente central da estrutura local é o serviço de sistema `LocationManager`, que oferece APIs para determinar a localização e o rumo do dispositivo subjacente (se disponível).

Tal como acontece com outros serviços do sistema, não é possível instanciar um `LocationManager` diretamente. Em vez disso, deve-se solicitar uma instância do sistema chamando `getSystemService`

(Context.LOCATION_SERVICE). O método retorna um identificador para uma instância LocationManager.

Uma vez que sua aplicação tem uma LocationManager, o aplicativo é capaz de fazer três coisas:

- Consultar a lista de todos os LocationProviders para o local último usuário conhecido.
- Registrar/cancelar o registro para atualizações periódicas de localização atual do usuário a partir de um provedor de localização (especificado quer por critérios ou nome).
- Registrar/cancelar o registro para um determinado localização se o dispositivo estiver nas proximidades (especificado por raio em metros) de uma dada latitude / longitude.

Com o Google Maps API Android, pode-se adicionar mapas no aplicativo que são baseadas em dados do Google Maps. A API lida automaticamente possui acesso aos servidores Google Maps, os dados de download, visualização do mapa, e gestos de toque no mapa.

A classe de chave no Google Maps API Android é MapView. A MapView exibe um mapa com os dados obtidos a partir do serviço do Google Maps. Quando o MapView tem foco, ele irá capturar teclas pressionadas e gestos de toque para deslocar e fazer zoom no mapa automaticamente, incluindo manipulação de solicitações de rede para mapas adicionais. Ele também fornece todos os elementos da interface do usuário necessárias para que os usuários controlem o mapa. O aplicativo também pode usar métodos de classe MapView para controlar o mapa de programação e desenhar uma série de sobreposições na parte superior do mapa.

O Google Maps Android APIs não estão incluídos na plataforma Android, mas estão disponíveis em qualquer dispositivo com a Loja Google Play.

Para integrar o Google Maps em um aplicativo, será preciso instalar os serviços do Google Play no Android SDK.

Compatibilidade Android com dispositivos e telas

Suporte a Múltiplas Telas

Permite especificar os tamanhos de tela que dão suporte à aplicação e ativar o modo de compatibilidade para telas maiores do que o que seu aplicativo suporta. É importante que se use sempre este elemento em um aplicativo para especificar os tamanhos de tela que o aplicativo suporta.

Uma aplicação "suporta" um determinado tamanho de tela que se redimensiona corretamente para preencher a tela inteira. Redimensionamento normal aplicado pelo sistema funciona bem para a maioria das aplicações e não temos que fazer qualquer trabalho extra para fazer a aplicação funcionar em telas maiores do que um aparelho celular. No entanto, muitas vezes é importante otimizar UI para diferentes tamanhos de tela, fornecendo recursos de formatos alternativos. Por exemplo, podemos querer modificar o layout de uma atividade quando está em um tablet comparado a quando rodando em um aparelho celular.

No entanto, se o aplicativo não funciona bem quando redimensionado para caber tamanhos de tela diferentes, podemos usar os atributos do elemento `<supports-screens>` para controlar se o aplicativo deve ser distribuído para telas menores ou ter sua interface ampliada ("zoom ") para caber telas maiores usando o modo de compatibilidade do sistema tela. Quando não projetamos o aplicativo para tamanhos de tela maior e o redimensionamento normal não alcançara os resultados adequados, o modo de compatibilidade de tela irá escalar sua interface emulando uma tela de tamanho normal e média densidade, em seguida, ampliando de forma a preencher toda a tela. Abaixo o formato do elemento `<supports-screens>`.

```
<supports-screens android:resizeable=["true" | "false"]  
                  android:smallScreens=["true" | "false"]  
                  android:normalScreens=["true" | "false"]
```

```
android:largeScreens=["true" | "false"]
android:xlargeScreens=["true" | "false"]
android:anyDensity=["true" | "false"]
android:requiresSmallestWidthDp="integer"
android:compatibleWidthLimitDp="integer"
android:largestWidthLimitDp="integer"/>
```

Android pode ser executado em uma variedade de dispositivos que oferecem diferentes tamanhos de tela e densidades. Para os aplicativos, o sistema Android fornece um ambiente de desenvolvimento consistente em todos os dispositivos e lida com a maior parte do trabalho para ajustar cada interface de usuário do aplicativo para a tela em que é exibido. Ao mesmo tempo, o sistema fornece APIs que permitem o controle da interface do usuário do aplicativo para tamanhos de tela e densidades específicos, a fim de aperfeiçoar o design da interface do usuário para configurações de telas diferentes. Por exemplo, podemos querer uma interface para tablets que é diferente da interface do usuário para telefones celulares. Conforme modelo abaixo.

Embora o sistema execute dimensionamento e redimensionamento para fazer o aplicativo de trabalho em telas diferentes, devemos fazer um esforço para otimizar o aplicativo para diferentes tamanhos e densidades de tela. Ao fazer isso, maximizamos a experiência do usuário para todos os dispositivos e os usuários acreditam que a sua escolha foi realmente feita para os seus dispositivos, em vez de simplesmente esticada para caber na tela em seus dispositivos.

Visualizando o suporte de telas

Agora daremos uma visão geral do suporte para telas múltiplas do Android, incluindo: uma introdução aos termos e conceitos usados na API, um resumo das configurações de tela que o sistema suporta, com uma visão geral da API e características de compatibilidade de tela subjacente.

Termos e conceitos

Tamanho da tela, tamanho físico real, medida diagonal da tela, e para simplificar, o agrupamento de todos os tamanhos de telas Android elas foram divididas em quatro tipos: pequeno, normal, grande e extra grande.

Densidade tela

A quantidade de pixels dentro de uma área física da tela, geralmente dada em dpi (pontos por polegada). Por exemplo, uma tela de densidade "baixa" tem menos pixels dentro de uma determinada área física, em comparação com uma tela de densidade "normal" ou "elevada", para simplificar, os grupos de todas as densidades de tela do Android, elas foram divididas em quatro: alta, baixa, média e extra.

Orientação

A orientação da tela a partir do ponto de vista do utilizador. Esta é paisagem ou retrato, o que significa que a proporção de tela de aspecto é mais larga ou alta, respectivamente.

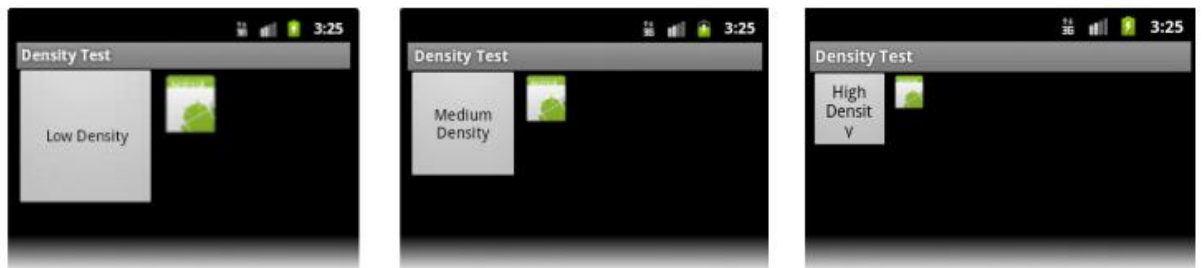
Resolução

O número total de pixels físicos sobre uma tela. Ao adicionar o suporte para múltiplas telas, as aplicações não trabalham diretamente com a resolução, aplicações deve estar preocupado apenas com o tamanho da tela e densidade, como especificado pelo tamanho generalizado e grupos de densidade.

Independência densidade

O aplicativo realiza a "independência densidade" quando se preserva o tamanho físico (do ponto de vista do usuário) de elementos de interface de usuário ao exibir telas com diferentes densidades.

Manter a independência da densidade é importante porque, sem ela, um elemento de UI (por exemplo, um botão) aparece fisicamente maior em uma tela de baixa densidade e menor numa tela de alta densidade. Tais mudanças de densidade relacionadas ao tamanho podem causar problemas no layout da aplicação e sua usabilidade. Figuras abaixo mostram as diferenças entre um aplicativo quando ele não fornece independência densidade e quando isso acontece, respectivamente.



Exemplo de aplicação sem suporte para diferentes densidades, como mostrado em baixo, médio e telas de alta densidade.



Exemplo de aplicação com bom suporte para diferentes densidades (é independente da densidade), como mostrado em baixo, telas de densidade média e alta.

Usando qualificadores de configuração

O android suporta diversas configurações que permitem controlar a forma como o sistema seleciona seus recursos alternativos com base nas características da tela do dispositivo atual. Um qualificador de configuração é uma string que podemos acrescentar a um diretório de recursos em seu projeto Android e especificar as configurações para a qual os recursos são projetados.

Para utilizar um qualificador de configuração:

- Crie um novo diretório no res do seu projeto/diretório usando o formato: <resources_name>-<qualifier>, onde <resources_name> é o nome do recurso padrão (como drawable ou layout) e <qualifier> é um qualificador de configuração conforme a tabela abaixo, especificando a configuração da tela para o qual estes recursos forem utilizados (tais como HDPI ou xlarge).

característica	Qualifier	Descrição
Size pequeno. normal. grande.	small	Recursos para telas de tamanho pequeno.
	normal	Recursos para telas de tamanho normal.
	large	Recursos para telas de tamanho grande.
	xlarge	Recursos para telas de tamanho extra grande.
Density dpi). (~160dpi). hdpi xhdpi (~320dpi). Estes densidade.	ldpi	Recursos de baixa densidade (~120 dpi).
	mdpi	Recursos de média densidade (~160dpi).
	hdpi	Recursos de alta densidade (~240 dpi).
	xhdpi	Recursos de densidade extra alta (~320dpi).
	nodpi	Recursos para todas as densidades. recursos são independentes da densidade.
	tvdpi	Recursos para telas com densidade em entre mdpi e hdpi; cerca de 213 dpi.

Orientation horizontal.	land	Orientação de telas no sentido (relação de aspecto de largura).
	port	Recursos de telas no sentido vertical (relação de aspecto de altura).

Aplicações para Web e Android

Criando aplicativos Web

Aplicativos Web

Se desejarmos utilizar uma aplicação web, como parte de um aplicativo cliente, podemos fazer isso usando WebView. A classe WebView é uma extensão da classe Android View que permite visualizar páginas da web como parte de um layout em um atividade. Ele não inclui quaisquer características de um navegador totalmente desenvolvido, como controles de navegação ou uma barra de endereços.

Um cenário comum em que o uso WebView é útil é quando desejamos fornecer informações em seu aplicativo que podemos precisar atualizar. Dentro de sua aplicação Android, podemos criar uma atividade que contém um WebView, em seguida, usar isso para exibir o documento que está hospedado online.

Outro cenário em que WebView pode ajudar é se o aplicativo fornece dados para o usuário que sempre requer uma ligação à Internet para obter dados, como e-mail. Neste caso, podemos achar que é mais fácil construir um WebView em sua aplicação Android que mostra uma página web com todos os dados do usuário, em vez de executar uma solicitação de rede, em seguida, analisar os dados e torná-lo em um layout Android. Em vez disso, podemos criar uma página web que é adaptado para dispositivos Android e depois implementar um WebView em sua aplicação Android que carrega a página web.

Adicionando um WebView ao aplicativo

Por exemplo, aqui está um arquivo de layout em que o WebView ocupa a tela:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

Para carregar uma página da Web no WebView, use `loadUrl()`. Por exemplo:

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.loadUrl("http://www.example.com");
```

Para que se possa ter acesso à Internet, solicitar a permissão `INTERNET` em seu arquivo de manifesto. Por exemplo:

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

Usando JavaScript no WebView

Se a página da Web que planejamos carregar necessita do uso de JavaScript, devemos habilitá-lo que funcione no WebView. Uma vez que o JavaScript esteja habilitado, também podemos criar interfaces entre o código do aplicativo e o código JavaScript.

Ativando JavaScript

JavaScript está desativado em um WebView por padrão. Podemos habilitá-lo através dos `WebSettings` anexados ao WebView. Podemos recuperar `WebSettings` com `getSettings()`, então habilitar o JavaScript com `setJavaScriptEnabled()`.

Por exemplo:

```
WebView myWebView = (WebView) findViewById(R.id.webview);  
WebSettings webSettings = myWebView.getSettings();  
webSettings.setJavaScriptEnabled(true);
```

WebSettings fornece acesso a uma variedade de outras configurações que podem ser úteis. Por exemplo, se estamos desenvolvendo uma aplicação web que é projetado especificamente para o WebView no seu aplicativo Android, então podemos definir uma sequência de agente de usuário personalizada com setUserAgentString(), em seguida, consultar o agente de usuário personalizada na página web para verificar se o cliente solicitando sua página web é realmente a sua aplicação Android.

Para vincular uma nova interface entre o JavaScript e o código do Android, chamando addJavascriptInterface(), passando uma instância de classe para ligar para o JavaScript e um nome de interface que com o JavaScript podemos chamar para acessar a classe.

Por exemplo, podemos incluir a seguinte classe em uma aplicação Android:

```
public class WebAppInterface {  
    Context mContext;  
  
    /** Instantiate the interface and set the context */  
    WebAppInterface(Context c) {  
        mContext = c;  
    }  
  
    /** Show a toast from the web page */  
    @JavascriptInterface  
    public void showToast(String toast) {
```

```

        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}

```

Isso cria uma interface Android para JavaScript executando o WebView. Neste ponto, a aplicação Web tem acesso à classe WebAppInterface. Por exemplo, aqui está um pouco de HTML e JavaScript que cria uma mensagem de brinde usando a nova interface quando o usuário clica em um botão:

```



```

```

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>

```

Manuseio de navegação de página

Quando o usuário clica em um link de uma página da web no WebView, o comportamento padrão do Android é mostrar uma aplicação que lida com URLs. Normalmente, o navegador padrão é aberto e carrega o URL de destino. No entanto, podemos substituir esse comportamento para o WebView, para abrir as solicitações dentro de sua WebView. sua história página web que é mantido por seu WebView.

Para abrir links clicados pelo usuário, basta fornecer um WebViewClient para o WebView, usando setWebViewClient(). Por exemplo:

```

WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());

```

Se queremos mais controle sobre onde a carga um link clicado, crie a sua própria `WebViewClient` que utilize o método `shouldOverrideUrlLoading()`. Por exemplo:

```
private class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        if (Uri.parse(url).getHost().equals("www.example.com")) {
            // This is my web site, so do not override; let my WebView load
            the page
            return false;
        }
        // Otherwise, the link is not for a page on my site, so launch another
        Activity that handles URLs
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        startActivity(intent);
        return true;
    }
}
```

Em seguida, criar uma instância desta `WebViewClient` novo para o `WebView`:

```
WebView myWebView = (WebView) findViewById (R.id.webview);
myWebView.setWebViewClient (novo MyWebViewClient ());
```

Histórico da página Navegando na web

Quando seu `WebView` substitui carregamento URL, ele automaticamente acumula um histórico de páginas visitadas. Podemos navegar para trás e para frente ao longo da história com o `GoBack()` e `GoForward()`.

Por exemplo, veja como a atividade pode usar o botão voltar para navegar para trás:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the Back button and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) &&
myWebView.canGoBack()) {
        myWebView.goBack();
        return true;
    }
    // If it wasn't the Back key or there's no web page history, bubble up to
the default
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```


Assinatura e publicação de aplicações Android

Controle de distribuição

Significa controlar as entregas dos seus aplicativos para os usuários que deseja, sobre os dispositivos que você deseja, em sua programação.

Publicação instantânea, atualizações instantâneas

No Google Play, você pode publicar seus produtos aos clientes instantaneamente. Basta fazer o upload e configurar o seu produto no Developer Console Google Play e pressione o botão Publish, seu aplicativo aparecerá na lista de loja dentro de horas.

Uma vez que seu aplicativo é publicado, você pode atualizá-lo tão frequentemente quanto você desejar. Você pode alterar os preços, configuração e opções de distribuição a qualquer momento através do Developer Console Google Play.

Mais tarde, quando você adicionar recursos ou resolver problemas de código, você pode publicar um binário atualizado a qualquer momento. Google Play faz a nova versão disponível imediatamente e notifica os clientes existentes que uma atualização está pronta para download. Para agilizar a implantação em toda a sua base de clientes, o Google permite que os usuários Play também passam aceitar atualizações automáticas de seu aplicativo, de modo que suas atualizações sejam entregues e instalados assim que publicá-las.

Alcançar os clientes que você deseja

Google Play faz mais do que conectar o seu aplicativo com os usuários, ajuda você a alcançar a distribuição mais ampla possível em todo o conjunto

de usuários Android, garantindo que a sua aplicação só está disponível para o público que você quer atingir.

Segmentação geográfica

Você pode usar controles no console do desenvolvedor Google Play para gerenciar facilmente a distribuição geográfica de seus aplicativos, sem qualquer alteração em seu binário do aplicativo. Você pode especificar que os países e territórios que você deseja distribuir para, e mesmo que as transportadoras (para alguns países).

Quando os usuários visitam a loja, Google Play garante que eles estão em um de seus países-alvo antes de baixar o aplicativo. Você pode mudar o seu país e designar um alvo a qualquer momento apenas por salvar as alterações no Developer Console Google Play.

Para ajudar você o mercado com os usuários ao redor do mundo, você pode localizar o seu anúncio em uma loja, incluindo detalhes de aplicativos e descrição, gráficos promocionais, screenshots e muito mais.

Capacidades de segmentação

Google Play também permite controlar a distribuição de acordo com as funções do dispositivo ou capacidades que a sua aplicação depende. Existem vários tipos de dependências que o aplicativo pode definir no seu manifesto, como recursos de hardware, formatos de compressão de textura OpenGL, bibliotecas, versões da plataforma Android, e outros.

Quando você faz o upload de seu aplicativo, o Google Play lê as dependências e estabelece as regras de distribuição necessárias.

O Google Play permite que você veja todos os dispositivos para os quais seu aplicativo está disponível e suas dependências (se houver). Desde o

Developer Console Google Play, você pode listar os dispositivos suportados e até mesmo excluir dispositivos específicos, se necessário.

Opções avançadas de entrega

Google Play oferece opções convenientes para gerir a forma como as suas aplicações são entregues aos usuários.

Na maioria dos casos, é fácil criar um aplicativo que suporta todos os seus tamanhos de tela alvo e versões da plataforma de um único APK. Distribuindo uma APK única para todos os seus usuários é uma abordagem altamente recomendável, porque é a maneira mais fácil de gerenciar e manter o aplicativo. Se você precisa entregar um APK diferente de dispositivos, o Google Play oferece uma maneira de fazer isso.

Uma opção chamada Multiple apoio APK permite criar pacotes APK múltiplos que usam o mesmo nome de pacote, mas diferem em seus formatos de compressão de textura OpenGL, tamanho de tela de suporte, ou versões da plataforma Android suportados. Você pode fazer o upload de todos os APKs ao Google Play sob uma listagem único produto e Google Play seleciona o melhor APK para entregar aos usuários, com base nas características de seus dispositivos.

A Expansão APK permite fazer upload de até duas transferências secundárias para cada APK publicado, incluindo APKs múltiplas. Cada um das duas pastas de expansão pode ser de até 2 GB cada e podem conter qualquer tipo de código. Quando você fizer o upload dos arquivos de expansão, o Google Play hospeda-os gratuitamente e manipula o download dos arquivos como parte da instalação do APK normal.

Proteger o seu Aplicativo

Para ajudá-lo a proteger seu aplicativo contra a pirataria, o Google Play oferece um serviço de licenciamento que você pode implementar em sua

aplicação. É um serviço baseado em rede que permite uma consulta de um aplicativo de servidor de licenciamento de confiança do Google Play para determinar se o aplicativo é licenciado para o usuário do dispositivo atual.