ContentProvider

Fagner Silva Martins

Objetivos

- O que são Provedores de conteúdo?
- Como ler dados de outras aplicações?
- Como identificar recursos de forma única?
- deixar dados disponíveis para outras aplicações?

Provedores de Conteúdo

Permite tornar dados de uma aplicação disponível para outras.

– Usa uma interface padronizada

- Declarar no manifest com <provider>
- Classe ContentProvider

– Baseado em URIs

- content://com.android.contacts/contacts/1
  - Prefixo – content://
  - Autoridade – com.android.contacts
  - Caminho - /contacts/
  - Id do registro (opcional) – 1
- Classe UriMatcher para validar URIs

Content Providers

Para construir é necessário implementar os métodos a seguir procedimento complexo, ver exemplo)

public String getType(Uri uri);	Retorna o tipo do URI
public boolean onCreate();	Chamado pelo Android quando provider é criado
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder);	Realiza uma consulta e retorna ao usuário
public Uri insert(Uri uri, ContentValues values);	Insere um novo registro e retorna seu URI
public int update(Uri uri, ContentValues values, String where, String[] whereArgs);	Atualiza um ou mais registros existentes.
public int delete(Uri uri, String where, String[] whereArgs);	Exclui um registro.

Content Provider

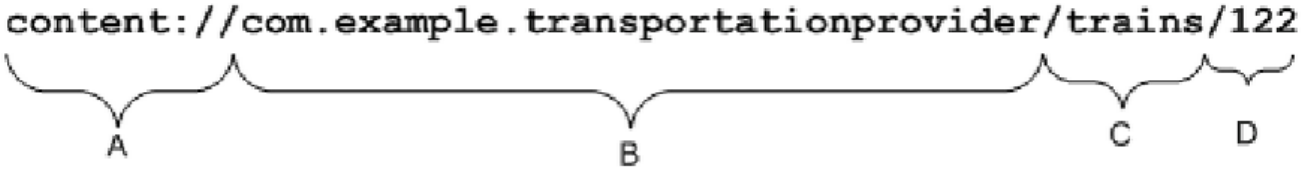
- Provedores de conteúdo são objetos que armazenam dados de forma permanente, e os tornam disponíveis para as outras aplicações.
- Android já fornece alguns provedores de conteúdo no pacote *android.provider.*
- Existem duas formas de tornar dados públicos:

extende-se a classe ContentProvider, ou insere-se os dados em algum provedor já existente.

Uniform Resource Identifier(URI)

- Provedoresdeconteúdosãolocalizadosvia URIs.
- Cada tipo de dado que o provedor disponibiliza é encontrado via uma URI diferente.
- O mesmo provedor pode disponibilizar mais de um tipo de dado, e portanto usar mais de uma URI.Exemplo:
  - android.provider.Contacts.Phones.CONTENT_URI
  - android.provider.Contacts.Photos.CONTENT_URI

URIs



- A) Segmento que indica que o dado é fornecido por um provedor de conteúdo.
- B)Identificador do provedor de conteúdo.
- C)Tipo(esubtipos)dodado.
- D)Identificador de um registro qualquer dos dados. Pode estar ausente.

Definindo Uma URI

```
import android.
import android.                                    umns;

public interface                                   ds BaseColumns {
  final String TAI                                 ents";
  final String AU
  final Uri CONTENT_URI = Uri.par
      + AUTHORITY + "/" + TABLE_N
  final String TIME = "time";
  final String TITLE = "title";
}
```

Faz sentido uma interface sem métodos?

Como usar essas constantes no código?

E outras constantes

```
import android.net.Uri;
import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
  final String TABLE_NAME = "events";
  final String AUTHORITY = "com.aula10";
  final Uri CONTENT_URI = Uri.parse("content://”
     + AUTHORITY + "/" + TABLE_NAME);
  final String TIME = "time";
  final String TITLE = "title";
}
```

Vale a pena definir também nomes para as colunas de dados como constantes.

Esses nomes tendem a ser usados o tempo todo!

Importação Estática

- Esse tipo de constante pode ser importada estaticamente.

```
import static com.aula12.Constants.AUTHORITY;
import static com.aula12.Constants.CONTENT_URI;
import static com.aula12.Constants.TABL
import static android.provider.BaseColum
```

Android já possui um nome predefinido para chaves primárias.

Criando um Provedor de Conteúdo

- Para criar um provedor de conteúdo, é necessário:

– Declarar o provedor de conteúdo no arquivo de manifesto.

– Definir uma forma de armazenar dados. Android provê o banco de dados SQLite, mas qualquer forma de armazenamento pode ser usada.

– Estender a classe ContentProvider

Declarando a URI no Manifesto

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.aula12"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon"
       android:label="@string/app_name">
       <provider android:name=".EventsProvider"
         android:authorities="com.aula12" />
     <activity android:name=".Events"
           android:label="@string/app_name">
       <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
     </activity>
  </application>
  <uses-sdk android:minSdkVersion="3" />
</manifest>
```

Estendendo ContentProvider

- ContentProvider é uma classe abstrata, contendo seis métodos abstratos:
  - query()
  - insert()
  - update()
  - delete()
  - getType()
  - onCreate()

```
public class EventsProvider extends ContentProvider {
  public int delete(Uri arg0, String arg1, String[] arg2) {}
  public String getType(Uri uri) {}
  public Uri insert(Uri uri, ContentValues values) {}
  public boolean onCreate() {}
  public Cursor query(Uri uri, String[] projection,
     String selection, String[] selectionArgs,
     String sortOrder) {}
  public int update(Uri uri, ContentValues values,
     String selection, String[] selectionArgs) {}
}
```

EventsProvider.java

MIME

- Se os dados fornecidos pelo provedor de conteúdo são novos, então é preciso definir um formato MIME (multipurpose Internetmail extensions) para eles. Nesse caso, é necessário um tipo para o registro individual, e um tipo para uma lista de registros:

EventsProvider.java

```
private static final String CONTENT_TYPE =
    "vnd.android.cursor.dir/vnd.example.event";

private static final String CONTENT_ITEM_TYPE =
    "vnd.android.cursor.item/vnd.example.event";
```

onCreate

EventsProvider.java

```
private static final int EVENTS = 1;
private static final int EVENTS_ID = 2;
private EventsData events;
private UriMatcher uriMatcher;

@Override
public boolean onCreate() {
   uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
   uriMatcher.addURI(AUTHORITY, "events", EVENTS);
   uriMatcher.addURI(AUTHORITY, "events/#", EVENTS_ID);
   events = new EventsData(getContext());
   return true;
}
```

O que seria um **UriMatcher**?

E essa classe **EventsData**?

query

EventsProvider.java

```
@Override
public Cursor query(Uri uri, String[] projection, String selection,
    String[] selectionArgs, String orderBy) {
  if (uriMatcher.match(uri) == EVENTS_ID) {
    long id = Long.parseLong(uri.getPathSegments().get(1));
    selection = appendRowId(selection, id);
  }
  SQLiteDatabase db = events.getReadableDatabase();
  Cursor cursor = db.query(TABLE_NAME, projection, selection, selectionArgs,
      null, null, orderBy);
  cursor.setNotificationUri(getContext().getContentResolver(), uri);
  return cursor;
}
```

getType

EventsProvider.java

```
@Override
public String getType(Uri uri) {
   switch (uriMatcher.match(uri)) {
   case EVENTS:
    return CONTENT_TYPE;
   case EVENTS_ID:
    return CONTENT_ITEM_TYPE;
   default:
    throw new IllegalArgumentException("Unknown URI " + uri);
   }
}
```

Onde foram definidas **quais URIs** esse provedor conhece?

insert

Qual o propósito **dessa** linha?

Por que disparamos **essa** exceção?

EventsProvider.java

```
@Override
public Uri insert(Uri uri, ContentValues values) {
    SQLiteDatabase db = events.getWritableDatabase();
    if (uriMatcher.match(uri) != EVENTS) {
      throw new IllegalArgumentException("Unknown URI " + uri);
    }
    long id = db.insertOrThrow(TABLE_NAME, null, values);
    Uri newUri = ContentUris.withAppendedId(CONTENT_URI, id);
    getContext().getContentResolver().notifyChange(newUri, null);
    return newUri;
}
```

delete

EventsProvider.java

```
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
   SQLiteDatabase db = events.getWritableDatabase();
   int count;
   switch (uriMatcher.match(uri)) {
   case EVENTS:
    count = db.delete(TABLE_NAME, selection, selectionArgs);
    break;
   case EVENTS_ID:
    long id = Long.parseLong(uri.getPathSegments().get(1));
    count = db.delete(TABLE_NAME, appendRowId(selection, id), selectionArgs);
    break;
   default:
    throw new IllegalArgumentException("Unknown URI " + uri);
   }
   getContext().getContentResolver().notifyChange(uri, null);
   return count;
}
```

update

EventsProvider.java

```
@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
  SQLiteDatabase db = events.getWritableDatabase();
  int count;
  switch (uriMatcher.match(uri)) {
  case EVENTS:
    count = db.update(TABLE_NAME, values, selection, selectionArgs);
    break;
  case EVENTS_ID:
    long id = Long.parseLong(uri.getPathSegments().get(1));
    count = db.update(TABLE_NAME, values, appendRowId(selection, id),
       selectionArgs);
    break;
  default:
    throw new IllegalArgumentException("Unknown URI " + uri);
  }
  getContext().getContentResolver().notifyChange(uri, null);
  return count;
}
```

Há que se implementar **esse** método.

EventsProvider.java

update

```
@Override
public int update(Uri uri, ContentValues values, String selection,
    String[] selectionArgs) {
  SQLiteDatabase db = events.getWritableDatabase();
  int count;
  switch (uriMatcher.match(uri)) {
  case EVENTS:
```

Mas, o que ele faz mesmo?

```
private String appendRowId(String selection, long id) {
   return _ID + "=" + id
      + (!TextUtils.isEmpty(selection) ? " AND (" + selection + ')' : "");
 }
```

```
        selectionArgs);
    break;
  default:
    throw new Illeg                    Unknown URI " + uri);
  }
  getContext().get                     yChange(uri, null);
  return count;
}
```

E como podemos ter acesso a um provedor de conteúdo?

Usando um provedor de conteúdo

- São necessárias três informações para que possamos ler um provedor de conteúdo:
  - A URI que identifica o provedor
  - Os nomes dos campos que queremos ler ou escrever
  - O tipo dos dados desses campos.

Quais passos precisam ser tomados por uma atividade que lê nosso provedor?

Lendo Dados

Events.java

```
private static String[] FROM = { _ID, TIME, TITLE, };
private static String ORDER_BY = TIME + " DESC";

private Cursor getEvents() {
   return managedQuery(CONTENT_URI, FROM, null, null, ORDER_BY);
}
```

O que são os argumentos **desse** construtor?

E quais informações seriam necessárias para escrevermos dados?

Inserindo dados no Provedor

Events.java

```
private void addEvent(String string) {
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
    getContentResolver().insert(CONTENT_URI, values);
}
```

Falta agora sermos capazes de exibir os eventos na tela.

ContentProvider vs SQLiteDatabase

ContentProvider

```
private Cursor getEvents() {
  return managedQuery(CONTENT_URI, FROM, null,
    null, ORDER_BY);
}
```

```
private void addEvent(String string) {
     ContentValues values = new ContentValues();
     values.put(TIME, System.currentTimeMillis());
     values.put(TITLE, string);
     getContentResolver().insert(CONTENT_URI, values);
}
```

SQLiteDatabase

```
private Cursor getEvents() {
  SQLiteDatabase db = events.getReadableDatabase();
  Cursor cursor = db
    .query(TABLE_NAME, FROM, null, null, null, null,
      ORDER_BY);
  startManagingCursor(cursor);
  return cursor;
}
```

```
private void addEvent(String string) {
  SQLiteDatabase db = events.getWritableDatabase();
  ContentValues values = new ContentValues();
  values.put(TIME, System.currentTimeMillis());
  values.put(TITLE, string);
  db.insertOrThrow(TABLE_NAME, null, values);
}
```

Mostrando os Eventos

Events.java

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.evtlist);
    addEvent("onCreate");
    Cursor cursor = getEvents();
    showEvents(cursor);
}
```

Precisamos implementar **esse** método.

Sera que ele vai ser diferente do método que já tínhamos feito para o banco de dados tradicional?

Mostrando os Eventos

Qual deve ser a superclasse dessa atividade?

Events.java

```
private static String[] FROM = { _ID, TIME, TITLE, };
private static int[] TO = { R.id.rowid, R.id.time, R.id.title, };

private void showEvents(Cursor cursor) {
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
        R.layout.item, cursor, FROM, TO);
    setListAdapter(adapter);
}
```

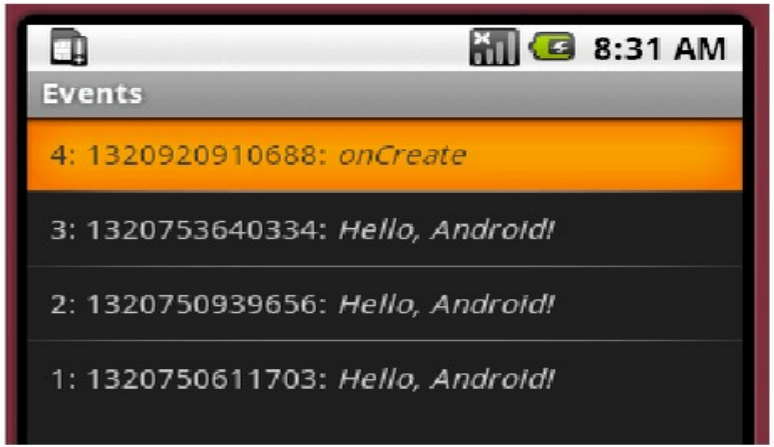
ListActivity

Events.java

```
public class Events extends ListActivity {
    ...
    public void onCreate(Bundle savedInstanceState) {...}
    private void addEvent(String string) {...}
    private Cursor getEvents() {...}
    private void showEvents(Cursor cursor) {...}
}
```

E como poderíamos remover eventos da lista?

Qual interface o usuário teria para remover eventos?



Removendo Itens do Provedor

Events.java

```
public void onItemClick(AdapterView<?> av, View v, int pos, long id) {
    View v0 = ((RelativeLayout)v).getChildAt(0);
    String strId = ((TextView)v0).getText().toString();
    Log.v("Events", "Deleting " + strId);
    getContentResolver().delete(CONTENT_URI, "_ID = " + strId, null);
    showEvents(getEvents());
}
```

O que **esse** código está fazendo?

E **essa** parte, o que faz ela?

E **esses** parâmetros, para que servem eles?

E que outras mudanças na classe são necessárias?

Apagando uma Entrada

```
public final boolean onContextItemSelected(final MenuItem item) {
  AdapterContextMenuInfo info =
      (AdapterContextMenuInfo) item.getMenuInfo();
  RelativeLayout rl = (RelativeLayout)info.targetView;
  TextView tv = (TextView)rl.getChildAt(0);
  String strId = tv.getText().toString();
  switch (item.getItemId()) {
  case del:
    getContentResolver().delete(CONTENT_URI, "_ID = " + strId, null);
    showEvents(getEvents());
    break;
  case show:
    ...
```

Mostrar o item selecionado é mais complicado

Cursor

- Um cursor é um iterador.
  - Iterator é um padrão de projetos.

O que faz o código abaixo?

```
Cursor cur = managedQuery(event, null, null, null, null);
Log.v("show", String.valueOf(cur.getColumnCount()));
Log.v("show", cur.getColumnName(0));
Log.v("show", cur.getColumnName(1));
Log.v("show", cur.getColumnName(2));
int timeColumn = cur.getColumnIndex(TIME);
int titleColumn = cur.getColumnIndex(TITLE);
cur.moveToFirst();
String eventStr = cur.getString(titleColumn);
eventStr += "(" + cur.getLong(timeColumn) + ")";
Log.v("show", eventStr);
```

Implemente o caso show de nosso menu.

```
case show:
    Uri event = Uri.withAppendedPath(CONTENT_URI, strId);
    Cursor cur = managedQuery(event, null, null, null, null);
    Log.v("show", String.valueOf(cur.getColumnCount()));
    Log.v("show", cur.getColumnName(0));
    Log.v("show", cur.getColumnName(1));
    Log.v("show", cur.getColumnName(2));
    int timeColumn = cur.getColumnIndex(TIME);
    int titleColumn = cur.getColumnIndex(TITLE);
    cur.moveToFirst();
    String eventStr = cur.getString(titleColumn);
    eventStr += "(" + cur.getLong(timeColumn) + ")";
    Log.v("show", eventStr);
    break;
```

Obrigado!

Contato: fagner.silva.martins@gmail.com