ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**

Edición 2020 - Materia: **Aplicaciones Android**

TEMA: Content Providers

Consideraciones

- Estas transparencias **no** tienen el objetivo de suplir las clases.
- Por tanto, serán **complementadas** con ejemplos, códigos, profundizaciones y comentarios por parte del docente.
- El **orden** de dictado de estos temas está sujeto a la consideración del docente.

Referencias

- Documentación para developers:
  - *http://developer.android.com/*
- Proveedores de contenido:
  - *https://developer.android.com/guide/topics/providers/content-providers.html*
- Conceptos básicos sobre el proveedor de contenido:
  - *https://developer.android.com/guide/topics/providers/content-provider-basics.html*
- Proveedor de calendario:
  - *https://developer.android.com/guide/topics/providers/calendar-provider.html*
- Proveedor de contactos:
  - *https://developer.android.com/guide/topics/providers/contacts-provider.html*
- Cargadores:
  - *https://developer.android.com/guide/components/loaders.html*
- Referencia: android.provider package
  - (*http://developer.android.com/reference/android/provider/package-summary.html*).

Agenda

- Content Providers
- Cursor Loaders

Content Providers (1)

- Proveen información estructurada de una forma genérica.
- Permiten especificar seguridad.
- La información puede ser utilizada por varias aplicaciones (por ej.: Contactos). En caso contrario, SQLite o similar puede ser una mejor solución.
- Interfaz única **ContentResolver** que se obtiene del Contexto de la aplicación.
  - *ContentResolver cr* = *getContentResolver*();
  - Clase **ContentValues** representa la información a almacenar / obtener.
    - Set de tipo clave / valor tipados.
    - *getBoolean(String key)* / *put(String* key, *Boolean value)* / *get(String key)*

Content Providers (2)

- Interfaz **ContentResolver**
  - Métodos principales
    - String getType (Uri uri)
    - *int delete(Uri url, String where, String[] selectionArgs)*
    - Uri insert(Uri url, ContentValues values)
    - *int update(Uri uri, ContentValues values, String where, String[] selectionArgs)*
    - Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
    - *InputStream openInputStream(Uri uri)*
    - OutputStream openOutputStream(Uri uri)

Content Providers (3)

- Luego, las invocaciones al *ContentResolver* son enviadas al *ContentProvider* particular.
- Android resuelve IPC (comunicación inter-procesos).
- Los métodos **insert**() y **update**() deberían ser "*thread-safe*".
- **Permisos**
  - Para interactuar con un proveedor es necesario pedir los permisos adecuados.
  - Los permisos son definidos por cada proveedor, en la aplicación Android que contiene la implementación especifica del *ContentProvider*.
  - Se piden agregando nodo <*uses-permission*> en Manifest XML

Content Providers (4)

- **URIs**
  - El proveedor utiliza la URI para acceder a la tabla con la información.
  - *Content URI* incluye:
    - Esquema (content://)
    - Nombre simbólico que identifica el proveedor
    - Identificador de tabla

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

  - Algunos proveedores permiten acceder a una fila especifica (ID) de una tabla de proveedor mediante URI especifica. Se debe utilizar el campo identificador _ID:

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

Content Providers (5)

- **Consulta**
  - Sintaxis similar para la construcción de consultas SQLite:
    - Proyección
    - Campos para Selección
    - Valores para Selección
    - Ordenación
    - Agrupación
  - Interfaz Cursor para el procesamiento de los resultados.

```
// 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
```

Content Providers (6)

- **Consulta (cont.)**
  - Se navega el Cursor y se obtienen columnas por índice y tipo de datos.
  - Al igual que con SQLite, hay que resolver el índice de columna a partir del nombre previamente.

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

```
while (mCursor.moveToNext()) {

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

Content Providers (7)

- **Insertar**

```
/*
 * 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
);
```

Content Providers (8)

Actualizar

```
// 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_%"};
```

```
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
);
```

Content Providers (9)

- **Eliminar**

```
// Defines selection criteria for the rows you want to delete
String mSelectionClause = UserDictionary.Words.APP_ID + " LIKE ?";
String[] mSelectionArgs = {"user"};
```

```
// 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
);
```

Content Providers (10)

- **AlarmClock**: Provee un Intent para iniciar una actividad de Alarma o Temporizador nuevo.
- **CallLog**: Contiene información sobre las llamadas recibidas y realizadas.
- **Contacts**: Este proveedor fue discontinuado en API Level 5. Reemplazado por *ContactsContract*. Solo retorna la información del contacto asociado a la primera cuenta Google configurada en el teléfono.
- **ContactsContract**: Contrato entre el proveedor de Contactos y las Aplicaciones. Esta API permite acceso a múltiples cuentas, soporte de agregación de contactos similares, etc.

Content Providers (11)

- **CalendarContract**: Contrato entre el proveedor de Calendario y las Aplicaciones.
  - **CalendarContract.Calendars** Tabla que contiene información específica del calendario. Cada fila representa un calendario (nombre, color, configuración, etc.).
  - **CalendarContract.Events**: Tabla que contiene información específica de los eventos. Cada fila representa un evento, y contiene el título, ubicación, hora de inicio/fin, recurrencia. Los participantes, recordatorios, y propiedades extendidas residen en su propia tabla, referenciada por el campo _ID.
  - **CalendarContract.Instances**: Tabla que contiene hora de inicio y fin de las ocurrencias de un evento (relación 1:N con Events).
  - **CalendarContract.Attendees**: Tabla que contiene los participantes o invitados de un evento.
  - **CalendarContract.Reminders**: Tabla que contiene las alertas / notificaciones de un evento.

Content Providers (12)

- **DocumentsProvider**: Provee de documentos almacenados en el dispositivo.
- **MediaStore**: Provee de meta-información para todos los archivos multimedia disponibles en el dispositivo (almacenamiento interno y externo).
- **Settings**: Contiene la configuración global del dispositivo.
- **UserDictionary**: Contiene todas las palabras definidas por el usuario para utilizar como texto predictivo durante la entrada de texto por parte del usuario.

Content Providers (13)

- **Telephony**: Contiene información relacionada a la operativa del teléfono, SMSs, MMSs, lista de APNs, etc.
  - **Telephony.Sms**: Contiene todos los mensajes de texto SMS.
  - **Telephony.Sms.Conversations**: Contiene todos los mensajes de texto SMS enviados.
  - **Telephony.Sms.Draft**: Contiene todos los mensajes de texto SMS en borrador.
  - **Telephony.Sms.Inbox**: Contiene todos los mensajes de texto SMS recibidos.

Cursor Loaders (1)

- Permiten cargar información de manera asíncrona desde una Activity o Fragment.
- Monitorean la fuente de datos por cambios.
  - Actualización ante cambios en origen de datos.
  - Uso de Callbacks.
- (Re)Conexión automática luego de cualquier cambio.

- **LoaderManager**: clase abstracta asociada a Actividad o Fragmento para manejar múltiples instancias de *Loaders*. Hay una única instancia por Actividad / Fragmento. Soporte para manejo de ciclo de vida de los mismos. Típicamente utilizado con *CursorLoader*.

Cursor Loaders (2)

- **LoaderManager.LoaderCallbacks**: interface para que cliente maneje los eventos del *Loader*. Por ejemplo, se debe utilizar el callback ***onCreateLoader***() para crear un nuevo *Loader* con la información que queremos mostrar.
  - **onCreateLoader**(): Instancia y retorna un nuevo Loader para el ID dado.
  - **onLoadFinished**(): Invocado cuando un Loader creado previamente ha finalizado la carga.
  - **onLoaderReset**(): Invocado cuando un Loader creado previamente esta siendo reseteado.

Cursor Loaders (3)

- **Loader**: clase abstracta que realiza la carga asincrónica de la información. Es la clase base. Típicamente se debe utilizar *CursorLoader*, pero se pueden implementar *Loaders* específicos (recordar implementar mecanismos de monitoreo de la fuente de datos para actualización).
- **AsyncTaskLoader**: *Loader* abstracto que provee una ***AsyncTask*** para obtener la información.
- **CursorLoader**: Subclase de ***AsyncTaskLoader*** que realiza consultas no bloqueantes (nuevo *Thread*) sobre un ***ContentResolver*** y devuelve un Cursor como resultado. Este método es el preferido para cargar información de un ***ContentProvider*** en lugar de utilizar las APIs de Activity o Fragment para realizar las consultas.

Cursor Loaders (4)

- **Start** / **Restart** loaders
  - Típicamente, inicialización en **onCreate**() de Actividad (idem Fragments) con ***getLoaderManager().initLoader()***.
  - Inicializa un Loader si no existe aún. Se llama a ***onCreateLoader***() de ***LoaderManager.LoaderCallbacks***.
  - Reinicia un loader si existe uno con mismo ID.