Memorizzazione di dati

Introduzione

In questa lezione vedremo quali sono i metodi principali messi a disposizione da una applicazione Android per memorizzare dei dati in maniera permanente e successivamente recuperarli.

Memorizzazione di dati

- Un tipico sistema operativo per PC fornisce un file-system comune che ogni applicazione può usare per memorizzare e rileggere file che a seconda dei permessi possono essere anche letti da altre applicazioni.
- Android usa un approccio diverso: tutti i dati delle applicazioni, inclusi i file, sono privati di quella applicazione.
- Esistono dei meccanismi mediante i quali una applicazione può condividere i propri dati con le altre.

Meccanismi di memorizzazione

Android fornisce 6 meccanismi per memorizzare e recuperare dati.
Il loro utilizzo dipende da tre da 2 aspetti fondamentali:

- I dati devono essere privati oppure possono essere accessibili da altre applicazioni
- I dati richiedono molto oppure poco spazio

Meccanismi di memorizzazione

Questi meccanismi sono:

- Shared Preferences (dati pubblici/privati)
- Internal Storage (dati privati)
- External Storage (dati pubblici)
- SQLite Database (dati strutturati privati)
- Rete (web service)
- Content Provider (dati privati accessibili)

Preferences ...

- Una applicazione può scrivere e leggere valori (detti ”Preferences”) condivisi con altre applicazioni dello stesso package (”shared preferences”) oppure privati dell'Activity.
- Per Preferences si intendono le informazioni di personalizzazione di una applicazione come per es. il colore di sfondo, i tipi di carattere, la suoneria predefinita...
- Tali Preferences possono essere definite tramite codice oppure mediante uno specifico file xml da salvare nella cartella res\xml

Preferences ...

- Per scrivere e leggere le shared preferences si usa il metodo `getSharedPreferences()`.
- Per scrivere e leggere le preferences private si usa il metodo `getPreferences()`.

Preferences ...

E' possibile definire le seguenti preferences:

- CheckBoxPreferences
- DialogPreferences
- EditTextPreferences
- ListPreferences
- RingtonePreferences

Preferences

```
<PreferenceScreen
      xmlns:android="http://schemas.android.com/apk/res/android"
       android:key="first_preferencescreen">
   <CheckBoxPreference
         android:key="wifi enabled"
         android:title="WiFi" />
   <PreferenceScreen
         android:key="second_preferencescreen"
         android:title="WiFi settings">
      <CheckBoxPreference
            android:key="prefer wifi"
            android:title="Prefer WiFi" />
      ... altre preferences ...
   </PreferenceScreen>
</PreferenceScreen>
```

Internal Storage 1/3

- I file creati da un'applicazione sono salvati all'interno della directory

data/data/<package>/

- Soltanto l'app può scrivere all'interno di questa cartella.
- E' possibile creare sottocartelle all'interno di essa nelle quali le altre applicazioni possono leggere e scrivere.
- Un'applicazione può scrivere e leggere file locali usando `openFileOutput()` e `openFileInput()` con nome locale e percorso.

Internal Storage 2/3

- Se ci sono file statici inseriti nel progetto sotto **`res/raw/`**, bisogna accedervi usando il metodo **`openRawResource()`.**

Internal Storage 3/3

I parametri utilizzabili sono:

- MODE_PRIVATE – Nessun accesso dalle altre app
- MODE_WORLD_READABLE – Sola lettura pe le altre app
- MODE_WORLD_WRITABLE – Accesso in scrittura per le altre app
- MODE_WORLD_READABLE | MODE_WORLD_WRITABLE – Accesso in lettura / scrittura per le altre app

External Storage 1/4

- Un dispositivo Android dispone di un External Storage, tipicamente una SD card, rimovibile o meno
- Tutti i file e le directory salvate all'interno dalla SD sono accessibili da parte di tutte le app
- Per poter accedere in lettura non è necessario specificare alcun permission, mentre in scrittura è necessario specificare il permesso:

  android.permission.WRITE_EXTERNAL_STORAGE
- Per ottenere la root della memoria esterna:

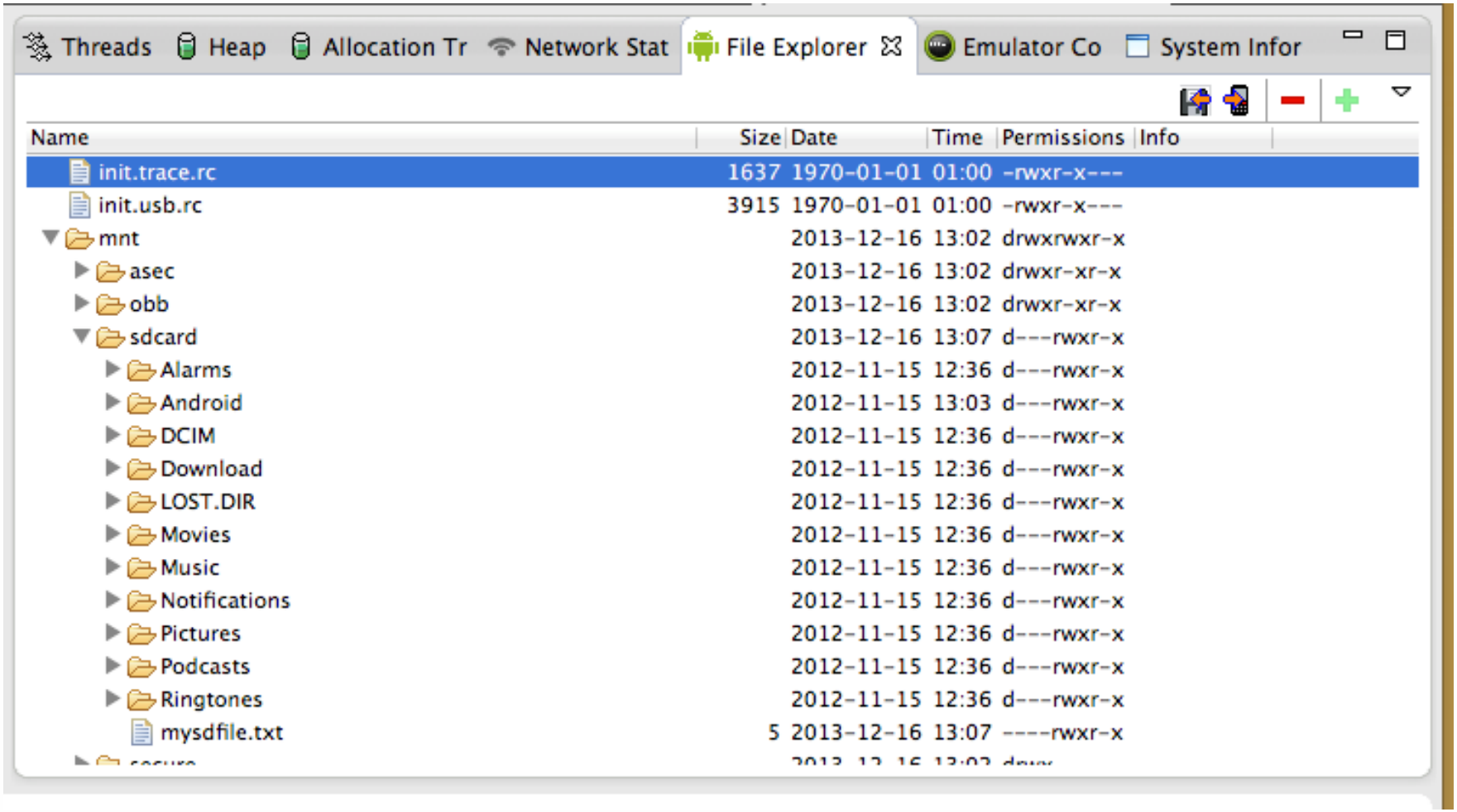
```
File sdcardDir =
Environment.getExternalStorageDirectory();
```

External Storage 2/4

Prima di procedere nelle operazioni di lettura/scrittura sulla SD card è necessario verificarne l'accessibilità mediante un apposito controllo:

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)) {
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    mExternalStorageAvailable = mExternalStorageWriteable = false;
```

SD Card nell'emulatore



External Storage 3/4

- Per accedere a file privati sulla SD card si utilizza:

getExternalFilesDir() per API level ≥8

getExternalStorageDirectory() per API level < 8

- Nel primo caso i file saranno creati all'interno della SD card nella directory:

/Android/data/<package_name>/files/

- I file così creati sono file privati dell'app
- ”Tipicamente” non dovrebbero essere visibili alle altre app.
- Non esiste alcuna garanzia di sicurezza per questi file
- Se l'app viene rimossa questi file saranno cancellati.

External Storage 4/4

- Sulla SD card possono essere disponibili delle aree condivise all'interno delle qualsi salvare file pubblici: DIRECTORY_MUSIC, DIRECTORY_PICTURES, DIRECTORY_RINGTONES...
- Per accedere a file condivisi sulla SD card si utilizza:

getExternalStoragePublicDirectory() per API level ≥8

getExternalStorageDirectory() per API level < 8

- Nel primo caso i file saranno creati all'interno della SD card nella directory predefinita per il tipo di file

```
File path = Environment.getExternalStoragePublicDirectory(
Environment.DIRECTORY_PICTURES);
```

- Nel secondo caso si dovrà esplicitare il path da usare.

- Android supporta il sistema di database SQLite.
- SQLite è open source, supporta le funzionalità standard dei database relazionali e richiede pochissima memoria a runtime (circa 250 KBytes)
- Supporta i tipi di dato TEXT, INTEGER e REAL
- Per utilizzare SQLite in Android non è necessario alcun tipo di installazione o configurazione.
- Un database SQLite è privato: pertanto se una app vuole esporne il contenuto dovrà farlo mediante i Content Provider

- Il package android.database contiene tutte le classi generali per operare con il database
- Il package android.database.sqlite contiene le classi specifiche di sqlite.
- Android include nell'SDK il tool sqlite3, che permette di visualizzare il contenuto delle tabelle, eseguire comandi SQL ed effettuare sui database SQL altri tipi di operazioni.
- Tutti i database sul dispositivo sono memorizzati nella directory

```
/data/data/<NOME_PACKAGE>/databases
```

SQLiteOpenHelper

- La classe che dovrà gestire il nostro database è ereditata da SQLiteOpenHelper
- Normalmente si sovrascrivono i metodi onCreate() per creare il database e onUpgrade() per aggiornare il database in caso di modifiche nel suo schema.
- Come *primary key* delle tabelle si utilizza ”_id”.
- Una buona pratica è quella di definire classe per ciascuna Tabella del database con almeno i due metodi onCreate() e onUpdate().

Cursor

- Il Cursor è l'oggetto restituito dall'esecuzione di una query e punta all'insieme dei risultati della query
- Per conoscere il numero degli elementi si utilizza il metodo count()
- Per navigare tra i risultati si usano i metodi moveToFirst(), moveToNext() e isAfterLast()
- Per accedere ai dati del Cursor si utilizzano i metodi getLong(indiceColonna) e getString(indiceColonna)

SQLiteDatabase 1/2

- SQLiteDatabase è la classe base mediante la quale opereremo con il nostro database
- Fornisce i metodi insert(), update() e delete()
- Il metodo execSQL() ci permentte di eseguire direttamente del codice SQL (con dei limiti)

```
db.execSQL("CREATE TABLE customer (_id INTEGER PRIMARY KEY NOT NULL, name TEXT NOT NULL, address TEXT);");
```

- Il metodo rowQuery() esegue una query SQL e restituisce un oggetto Cursor

```
Cursor mCursor = db.rawQuery(”SELECT * FROM customer WHERE _id = 1;”, null);
```

SQLiteDatabase 2/2

- Il metodo query() restituisce un oggetto Cursor sull'insieme dei risultati

Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

Content Provider

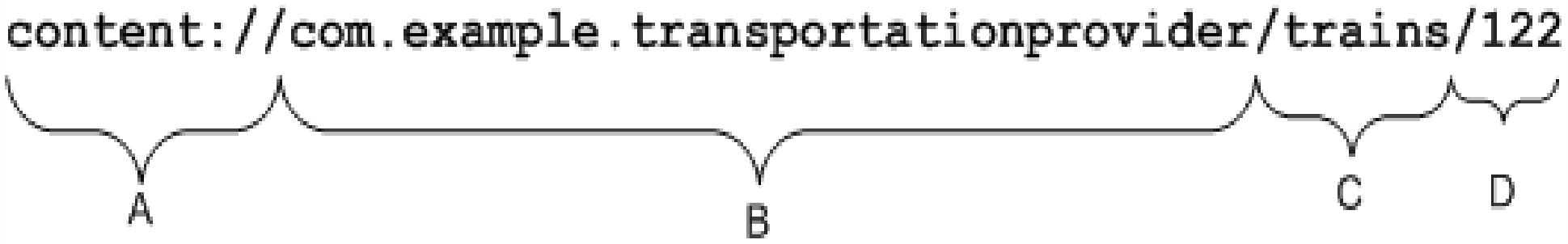
- Se si vogliono rendere pubblici dei dati bisogna creare un Content Provider.
- Questo oggetto permette di memorizzare dati accessibili da tutte le applicazioni.
- E' l'unico modo attraverso il quale package diversi possono condividere dati.
- Android di serie fornisce Content Provider per i tipi di dati più comuni (es. contatti o file multimediali).

Interrogare un Content Provider

- Il modo nel quale un Content Provider memorizza i dati è specifico dell'implementazione, ma tutti i Content Provider devono rispettare una convenzione comune per effettuare una query e restituire i risultati.
- Ogni Content Provider espone una stringa (URI, Uniform Resource Identifier) univoco.
- Un URI può indicare tutti gli oggetti di un certo tipo oppure uno specifico record.

Sintassi dell'URI

- A: prefisso standard (sempre uguale)
- B: percorso completo della classe
- C: percorso completo del tipo di dato
- D: se presente è l'ID dello specifico record



Esempi di URI

- `content://contacts/people/` è l'URI che restituisce la lista di tutti i contatti sul dispositivo.
- `content://contacts/people/23` è l'URI che restituisce il contatto con ID=23
- `content://media/internal/images/` restituisce la lista delle immagini memorizzate nel dispositivo
- `content://media/external/images/` restituisce la lista delle immagini contenute nella scheda di memoria (es. SD-Card)

Content Provider - Permission

- Il modello di sicurezza di Android richiede che un'applicazione non possa eseguire una data azione ”sensibile” senza aver richiesto ed ottenuto la relativa Permission
- Tale richiesta deve essere dichiarata all'interno dell' AndroidManifest.xml e viene gestita a runtime, ma è durante l'installazione che l'utente concede o non concede la Permission richiesta.

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

Content Provider - Esempio

L'esempio seguente accede al Content ”Contatti” elencando il contenuto

```
ContentResolver cr = getContentResolver();
Uri uri =
Uri.parse("content://com.android.contacts/contacts"););
 Cursor cur = cr.query(uri, null, null, null, null);
 if (cur.getCount() > 0) {
       while (cur.moveToNext())
       {
           String id = cur.getString(

cur.getColumnIndex(ContactsContract.Contacts._ID));
           String name = cur.getString(

cur.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
      }
}

cur.close();
```

Rete

- Infine non bisogna dimenticare che ovviamente i dati oltre che sul dispositivo possono essere memorizzati sulla rete.

- I package utili a questo scopo sono:

`java.net.*`

e

`android.net.*`

Conclusioni

In questa lezione abbiamo visto gli strumenti messi a disposizione da Android per la memorizzazione permanente dei dati: le Preferences, i File, i Database, i Content Provider e ovviamente la rete.