Android: data persistence

Romain Rochegude

2016.09.30

Introduction

Introduction

- POO basics: modeling the **domain**
- Modeling domain objects and their interactions
- Data bound with a remote API
- Need of a local database

The native way

The "raw" way

```java
private static final String
  SQL_CREATE_ENTRIES =
    "CREATE TABLE REPO (" +
    "_ID INTEGER PRIMARY KEY," +
    "NAME TEXT)";

private static final String
  SQL_DELETE_ENTRIES =
    "DROP TABLE IF EXISTS REPO ";
```

- Subclass SQLiteOpenHelper

```java
public class ReposDbHelper extends
  SQLiteOpenHelper {
    public static final int
      DATABASE_VERSION = 1;
    public static final String
      DATABASE_NAME = "repos.db";

    public ReposDbHelper(Context
      context) {
        super(context, DATABASE_NAME,
          null, DATABASE_VERSION);
    }
```

```java
    //...
    public void onCreate(SQLiteDatabase
      db) {
        db.execSQL(SQL_CREATE_ENTRIES);
    }

    public void onUpgrade(SQLiteDatabase
      db, int oldVersion, int
      newVersion) {
        db.execSQL(SQL_DELETE_ENTRIES);
        onCreate(db);
    }
}
```

- Get an instance of `SQLiteOpenHelper`

```
ReposDbHelper dbHelper =
new ReposDbHelper(getContext());
```

- Put Information into a Database

```
SQLiteDatabase db =
  dbHelper.getWritableDatabase();


ContentValues values = new
  ContentValues();
values.put("name", "a sample name");


long newRowId = db.insert("REPO", null,
  values);
```

- Read Information from a Database

```java
SQLiteDatabase db =
    dbHelper.getReadableDatabase();

String[] projection = { "_id", "name" };

String selection = "NAME = ?";
String[] selectionArgs = { "a sample
    name" };

String sortOrder = "NAME DESC";
```

```java
Cursor cursor = db.query(
    "REPO",
    projection,
    selection,
    selectionArgs,
    null,
    null,
    sortOrder);
cursor.moveToFirst();

long itemId = cursor.getLong(
    cursor.getColumnIndexOrThrow("_ID")
);
```

The ContentProvider way

- Provide a `ContentProvider` subclass dedicated to the application

```
public class RepoProvider extends
  ContentProvider {
}
```

- Define a specific `UriMatcher` and configure available URI

```java
public class RepoProvider extends
  ContentProvider {
    private static final UriMatcher
      sUriMatcher =
        new
          UriMatcher(UriMatcher.NO_MATCH);
}
```

```java
static {
  sUriMatcher.addURI("fr.test.app.provider",
      "repo", 1);
  sUriMatcher.addURI("fr.test.app.provider",
      "repo/#", 2);
}
```

- Override various CRUD methods

```java
public Cursor query(
    Uri uri,
    String[] projection,
    String selection,
    String[] selectionArgs,
    String sortOrder) {
    //...
```

```java
//...
switch (sUriMatcher.match(uri)) {
    case 2:
        selection = selection + "_ID = "
            + uri.getLastPathSegment();
        break;
    default:
        //...
}
```

- Use it through a `ContentResolver` instance

```
mCursor = getContentResolver().query(
    "fr.test.app.provider/repo/2",
    mProjection,
    mSelectionClause,
    mSelectionArgs,
    mSortOrder);
```

- Refer to the **open-sourced Google's iosched application**
- Helpful libraries to simplify the deal with a ContentProvider:
  - ProviGen
  - AutoValue: Cursor Extension

- Perform CRUD operations outside of the main thread

Query data using Loader

- To query the ContentProvider in an
  Activity, make it implementing
  LoaderManager.LoaderCallbacks<Cursor>

```java
public class ReposListActivity
    extends FragmentActivity
    implements
      LoaderManager.LoaderCallbacks<Cursor>
      {
}
```

- Start loading data with a loader identifier

```
getLoaderManager()
    .initLoader(LOADER_REPOS, null,
        this);
```

- Implement LoaderCallbacks...

- ...to create the CursorLoader

```java
@Override
public Loader<Cursor> onCreateLoader(
  int id, Bundle bundle) {
  if(id == LOADER_REPOS) {
    return new CursorLoader(
      this,
      "fr.test.app.provider/repo",
      mProjection,
      null, null, null);
  }
  //...
}
```

- . . . and deal with result

```java
@Override
public void
  onLoadFinished(Loader<Cursor> loader,
  Cursor cursor) {
    if(loader.getId() == LOADER_REPOS){
        // ...
    }
}
```

- See also: AsyncQueryHandler

The ORM way

The well-known: ORMLite

- Declare your model using ORMLite annotations

```java
@DatabaseTable(tableName = "REPO")
public class RepoEntity {
    @DatabaseField(columnName = "_ID",
        generatedId = true)
    public long _id;

    @DatabaseField
    public String name;
}
```

- declare the corresponding DAO

```java
public class DAORepo extends
   BaseDaoImpl<RepoEntity, Long> {

   public DAORepo(ConnectionSource cs)
       throws SQLException {
       this(cs, RepoEntity.class);
   }

   //...
}
```

- subclass `OrmLiteSqliteOpenHelper`

```java
public class DatabaseHelperTest
  extends OrmLiteSqliteOpenHelper {

  private static final String
    DATABASE_NAME = "test.db";

  private static final int
    DATABASE_VERSION = 1;
  //...
```

```java
//...
public DatabaseHelperTest(
  Context context) {

  super(context,
    DATABASE_NAME,
    null,
    DATABASE_VERSION,
    R.raw.ormlite_config);
}
//...
```

```java
//...
@Override
public void onCreate(SQLiteDatabase db,
  ConnectionSource cs) {
  TableUtils.createTable(cs,
    RepoEntity.class);
}

@Override
public void onUpgrade(SQLiteDatabase db,
  ConnectionSource cs, int oldVersion,
  int newVersion) {
  //...
}
```

- get the requested DAO

```
DatabaseHelperTest helper = //...

ConnectionSource cs =
  helper.getConnectionSource();

DatabaseTableConfig<RepoEntity>
  tableConfig =
  DatabaseTableConfigUtil.fromClass(cs,
    RepoEntity.class);

DAORepo dao = new DAORepo(cs,
  tableConfig);
```

- perform CRUD operations

```
// create

RepoEntity repo =
  new RepoEntity("a sample name");

dao.create(repo);
```

```java
// read

List<RepoEntity> repos =
    dao.queryBuilder()
        .where()
        .eq("name", "a sample name")
        .query();
```

- Performance: orm-gap gradle plugin

```
buildscript {
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath
        'com.github.stephanenicolas.ormgap'
      + ':ormgap-plugin:1.0.0-SNAPSHOT'
  }
}

apply plugin: 'ormgap'
```

- generate an ORMLite configuration file that boosts DAOs creations
- to use this file

```java
public RepoDatabaseHelper(Context
  context) {
    super(context,
      DATABASE_NAME,
      null,
      DATABASE_VERSION,
      R.raw.ormlite_config);
}
```

The attractive way: requery

- Object mapping
- SQL generator
- RxJava and Java 8 support
- No reflection, compile-time processing and generation
- Relationships support
- Callback method (@PostLoad)
- Custom type converters

- Define object mapping

```
@Entity
abstract class Repo {
    @Key @Generated
    int id;

    String name;
}
```

- Easy to perform SQL queries

```
Result<Repo> repos = data
  .select(Repo.class)
  .where(Repo.NAME.lower().like("%sample%"))
  .orderBy(Repo.ID.desc())
  .get();
```

Async management: RxJava

- Get a specific instance of `SingleEntityStore`

```java
DatabaseSource dbSource =
  new DatabaseSource(context,
    Models.DEFAULT, "test.db", 1);


Configuration conf =
  dbSource.getConfiguration();


SingleEntityStore<Persistable> data =
  RxSupport.toReactiveStore(new
    EntityDataStore<>(conf));
```

- and use it the RX way

```
data.select(RepoEntity.class)
    .get()
    .subscribeOn(Schedulers.newThread())
    .subscribe(/*...*/)
```

Conclusion

Conclusion

- Personal assessments of each way

	ContentProvider	ORMLite	requery
setup	-	+	+
performance	+	-	+
readability	-	+	+
maintainability	-	+	+