Android ROOM library

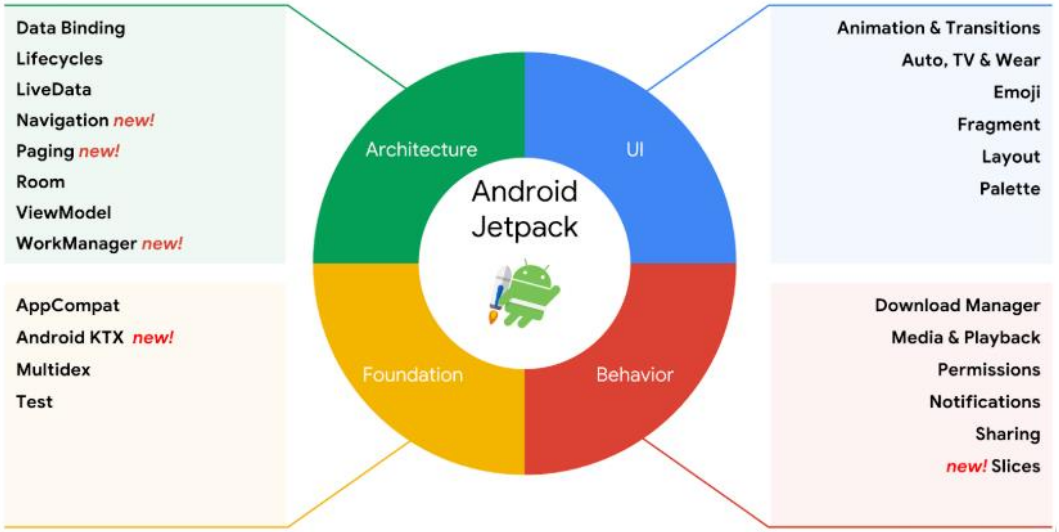
JEON YONGTAE
https://github.com/yongtaii/yongapps

1

ROOM

Room Persistence Library

JetPack component란?



JetPack 이란 ?

- 구글 IO에서 62개정도의 작은 세션들을 공개했습니다 그 세션들의 집합
- Android앱을 손쉽게 개발하도록 지원하는 android 소프트웨어 구성요소 컬렉션
- JetPack 컴포넌트로 상용구코드를 작성하지않고, 복잡한 작업을 간소화 시킵니다.

ROOM (Object Relational Mapping)

ROOM 이란 ?

- ORM(Object Relational Mapping) 라이브러리
- ROOM은 데이터베이스의 객체를 자바 or코틀린 객체로 매핑해주는것
- SQLite의 추상레이어 위에 제공하고 있으며 SQLite의 모든 기능을 제공하면서 편한 데이터베이스의 접근을 편하게 도와주는 라이브러리

ROOM (Object Relational Mapping)

– 기존 안드로이드의 SQLite는 사용이 매우 불편하여 ORMLite, greenDAO와 같은 ORM을 이용하여 컨트롤하는게 일반적이었다.

– ROOM은 구글에서 만든 공식 ORM이며, 여러가지 강력한 기능을 갖는다.

ROOM vs SQLite

```
openDatabase();

if (virusName.contains("\'")){
    virusName = virusName.replace( target: "\'",  replacement: "");
}

mDatabase.execSQL("insert into MVC_VLIST VALUES(null, '"+scanType
        +"', '"+packageName
        +"', '"+virusName
        +"', '"+check
        +"');");

closeDatabase();
```

1. SQLite 경우 쿼리에 대한 에러를 컴파일에 verification 없지만 ROOM에서는 컴파일 도중 SQL에 대한 유효성 검사가 가능합니다. 따라서 runtime crash를 막을 수 있다.

ROOM vs SQLite

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

    if (oldVersion < 2){
        db.execSQL("CREATE TABLE MVC_VLIST( _id INTEGER PRIMARY KEY AUTOINCREMENT," +
                "SCANTYPE TEXT, PKGNAME TEXT, MALNAME TEXT, VCHECK TEXT);");

    }
    if (oldVersion < 3){
        db.execSQL("CREATE TABLE MVC_VLIST2( _id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "SCANTYPE TEXT, PKGNAME TEXT, MALNAME TEXT, VCHECK TEXT);");

    }
    if (oldVersion < 4){
        db.execSQL("CREATE TABLE MVC_VLIST3( _id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                "SCANTYPE TEXT, PKGNAME TEXT, MALNAME TEXT, VCHECK TEXT);");

    }

}
```

2. Schema가 변경이 될경우 SQL쿼리를 수동으로 업데이트 해야하지만 ROOM의 경우는 쉽게 해결이 가능합니다.

ROOM vs SQLite

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

// Define a projection that specifies which columns from the database
// you will actually use after this query.
String[] projection = {
    BaseColumns._ID,
    FeedEntry.COLUMN_NAME_TITLE,
    FeedEntry.COLUMN_NAME_SUBTITLE
    };

// Filter results WHERE "title" = 'My Title'
String selection = FeedEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "My Title" };

// How you want the results sorted in the resulting Cursor
String sortOrder =
    FeedEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor cursor = db.query(
    FeedEntry.TABLE_NAME,     // The table to query
    projection,               // The array of columns to return (pass null to get all)
    selection,                // The columns for the WHERE clause
    selectionArgs,            // The values for the WHERE clause
    null,                     // don't group the rows
    null,                     // don't filter by row groups
    sortOrder                 // The sort order
    );
```

3. SQLite 경우 Java데이터 객체를 변경하기 위해 많은 상용구 코드(Boiler Plate code)를 사용해야하지만 ROOM의 경우 ORM라이브러리가 상용구 코드(Boiler Plate code) 없이 매핑 가능합니다. 대신 annotatinos을 사용한다.

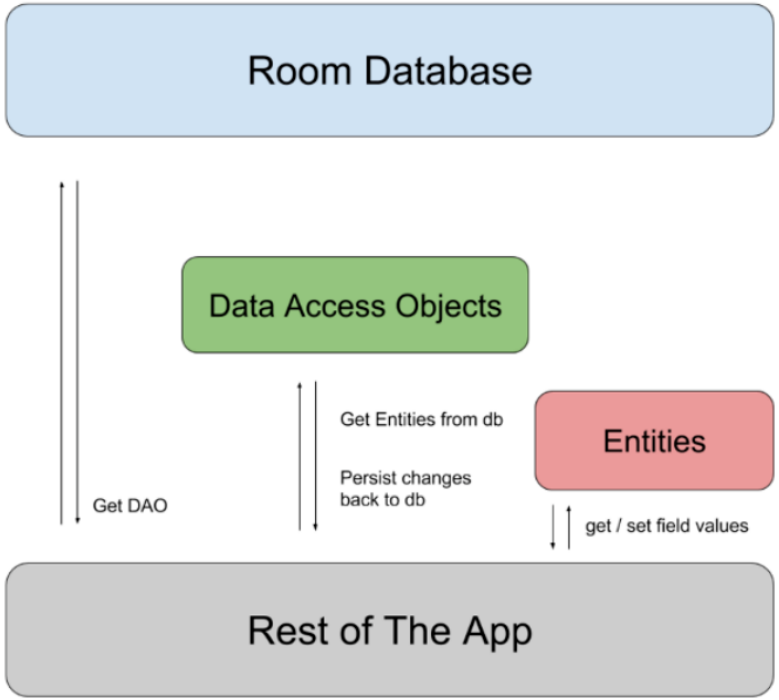
ROOM vs SQLite

```
@Query("SELECT * FROM Users WHERE id = :userId")
Single<User> getUserById(String userId);
```

```
@Query("SELECT * FROM Users WHERE id = :userId")
Maybe<User> getUserById(String userId);
```

4. ROOM의 경우 LiveData와 RxJava를 위한 Observation 으로 생성하여 동작할 수 있지만 SQLite는 그렇지 않습니다.

ROOM 구성요소



Room Structure : 크게 3가지로 구성됨

Database

Entity

Dao (Data Access Object)

ROOM 구현 - import ROOM

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.0'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support:cardview-v7:27.1.1'

    implementation 'android.arch.persistence.room:runtime:1.0.0'
    annotationProcessor 'android.arch.persistence.room:compiler:1.0.0'
}
```

Room Dependency를 Build.gradle(app)에 inport 한다

ROOM 구현 - Entity 생성

```
@Entity(tableName = MyDatabase.TABLE_NAME_TODO)
public class Todo implements Serializable {

    @PrimaryKey(autoGenerate = true)
    public int todo_id;

    public String name;

    public String description;

    @ColumnInfo(name = "category")
    public String category;

    @Ignore
    public String priority;

}
```

- @Entity(tableName="테이블네임")으로 테이블 이름을 사용자가 지정할 수 있다. (default 값은 Class명과 동일)
- PrimaryKey로 PK값을 지정한다. (중복 시 오류 발생) autoGenerate=true 값을 통해 자동으로 Key값을 생성할 수 있다.
- @ColumInfo(name="name") : 칼럼 명을 지정할 수있다

ROOM - Entity

- This is Model Class (column filed 처럼 행동하는 property 를 정의할 수 있는)
- Database 내의 테이블을 java나 kotlin으로 나타낸 것.
- @ColumInfo : column name을 set 할 수 있다.
- @PrimaryKey : 최소 한 개의 property는 반드시 @PrimarayKey를 가져야 한다
- @Embedded : 다른 filed name으로부터 column name을 set하기 위해서 사용

Entity Annotations

@Entity

테이블에 설정하는 조건으로써 컬럼들의 설정값을 한번에 정의할 수 있고, 테이블에 대한 설정또한 조절할 수 있다.

tableName() - database 내 테이블 이름 지정

indices() - 데이터베이스 쿼리 속도를 높이기 위한 컬럼들을 인덱스로 지정

inheritSuperIndices() - true로 설정할 경우 부모클래스에 선언된 모든 인덱스가 현재의 Entity클래스로 옮겨짐

primaryKeys() - 기본키로 지정하고 싶은 칼럼 값들을 한번에 설정

foreignKeys() - 외래키로 지정하고 싶은 칼럼 값들을 한번에 설정

ignoredColumns() - DB에 생성되기를 원하지 않는 컬럼을 한번에 설정

Entity Annotations

@PrimaryKey

Room Entity는 반드시 1개 이상의 primaryKey를 가져야 한다. autoGenerate = true 설정이 바람직하다

autoGenerate = true로 설정할 경우 유니크한 아이디값을 자동으로 생성

@ForignKey

객체간의 관계를 정의할 때 사용하며 여러 제약조건을 설정할 수 있다

예를들어 부모객체가 삭제될 때 참조하고있는 자식객체를 모두 삭제하는등의 기능등을 수행할 수 있다 (onDelete=CASCADE)

Entity Annotations

@ColumnInfo

Entity의 필드값에 컬럼 속성을 변경

name - DB에서의 컬럼명 지정. 기본값으로 Entity 내의 필드명이 사용된다

typeAffinity - 컬럼의 타입 지정(defalt : UNDEFINED, TEXT, INTEGER, REAL, BLOB)

index - 컬럼들의 인덱싱 생성

@Ignore

별도로 전달받는 인자값이 없으며 해당 Annotation을 설정할 경우 데이터베이스에 생성되지 않는다

ROOM 구현 - Dao interface

```java
@Dao
public interface DaoAccess {

    @Insert
    long insertTodo(Todo todo);

    @Insert
    void insertTodoList(List<Todo> todoList);

    @Query("SELECT * FROM " + MyDatabase.TABLE_NAME_TODO)
    List<Todo> fetchAllTodos();

    @Query("SELECT * FROM " + MyDatabase.TABLE_NAME_TODO + " WHERE category = :category")
    List<Todo> fetchTodoListByCategory(String category);

    @Query("SELECT * FROM " + MyDatabase.TABLE_NAME_TODO + " WHERE todo_id = :todoId")
    Todo fetchTodoListById(int todoId);

    @Update
    int updateTodo(Todo todo);

    @Delete
    int deleteTodo(Todo todo);
}
```

- Annotation을 활용하여 interface,abstract class로 작성한다

ROOM - Dao

- Database에 엑세스 하는데 사용되는 메서드들 (select, inesrt, delete, join …).

  데이터를 읽거나 쓸때 사용한다

- Data Access Object
- SQL query 문과 interface 역할을 한다
- @Insert , @Query, @Update, @Delete 등이 사용됩니다.

  @Update, @Delete는 변화되거나 삭제된 열을 나타내는 번호(int)를 return 한다

ROOM 구현 - Dao interface

- Room에서는 SQL을 활용한 직접적인 쿼리접근 대신에 DAO(Data Access Object)를 이용하여 데이터베이스에 접근해야 한다
- Query 구문 또한 Annotation으로 작성되어야 하며, 문자열 안에 있는 qeury의 자동완성 기능을 지원한다. 따라서 쿼리 작성시 오탈자가 날 확률이 매우 줄어들었다
- 기존 쿼리의 가장 큰 문제는 컴파일 시 쿼리상의 오류를 발견하지 못한다는 점이었다. Room DAO에서는 작성된 쿼리의 오류를 컴파일시에 발견하여 조기에 오류를 수정 할 수 있는 큰 장점이 있다.

Dao Annotation

```
@Insert
long insertTodo(Todo todo);

@Insert
void insertTodoList(List<Todo> todoList);
```

@Insert

- @Entity로 정의된 클래스만 인자로 받거나, 그 클래스의 Collection 또는 Array만 인자로 받을 수 있다
- 인자가 하나인 경우, long type의 return값 (insert 된 값의 rowId)을 받는다
- 인자가 다수인 경우, long[],List<Long> type의 return값 (insert 된 값의 rowId)을 받는다

ABORT(default) - 충돌 발생 시 트랜잭션 롤백

REPLACE - 충돌 발생 시 기존 데이터와 입력데이터 교체

IGNORE - 충돌 발생 시 기존데이터 유지, 입력데이터 버림

Dao Annotation

@Update

데이터를 갱신할 때 사용합니다. 전달받은 매개변수의 PK값에 매칭되는 entity를 찾아 갱신한다

@Delete

데이터를 삭제할 때 사용합니다. Insert, Update, Delete 모두 비슷한 파라미터를
받습니다. 전달받은 매개변수의 PK값에 매칭되는 entity를 찾아 삭제한다

@Query

주로 데이터를 선택할 때 사용한다. (다른 구문은 Annotation을 사용하는게 일반적 이다.)
Query는 DAO클래스중 가장 핵심 부분이며 데이터를 읽고 쓸 수 있게 한다. 컴파일시에
query검사가 이루어 지기때문에 런타임 오류를 최소화 할 수 있다.

ROOM 구현 - Database class

```java
@Database(entities = {Todo.class}, version = 1)
public abstract class MyDatabase extends RoomDatabase {

    private static MyDatabase INSTANCE;

    public static final String DB_NAME = "app_db";
    public static final String TABLE_NAME_TODO = "todo";

    public abstract DaoAccess daoAccess();

    public static MyDatabase getAppDatabase(Context context) {
        if (INSTANCE == null) {
            INSTANCE =
                    Room.databaseBuilder(context.getApplicationContext(), MyDatabase.class, MyDatabase.DB_NAME)
                            // allow queries on the main thread.
                            // Don't do this on a real app! See PersistenceBasicSample for an example.
                            .fallbackToDestructiveMigration()
                            .build();
        }
        return INSTANCE;
    }


    public static void destroyInstance() {
        INSTANCE = null;
    }

}
```

- @Database annotation을 이용하여 사용할 Entity를 배열로 입력한다.
- RoomDatabase를 상속받아 abstract 클래스를 작성한다

Database class

> *If your app runs in a single process, you should follow the singleton design pattern when instantiating an* `AppDatabase` *object.* ***Each*** `RoomDatabase` ***instance is fairly expensive, and you rarely need access to multiple instances within a single process.***

- RoomDatabase 인스턴스를 만드는 과정은 매우 비싼 작업이다. 하지만 접근은 자주하기 때문에 공식문서에서는 singleton패턴을 이용하여 만드는 것을 권장한다

ROOM - Database

- RoomDatabse를 반드시 상속해야 하는 abstract class
- 사용할 Entity 목록을 를 반드시 이곳에 작성해야 한다
- schema가 변경될 때마다 이곳에 version number를 업데이트 해야 한다.
- Database 접근 지점을 제공하며, DAO를 관리한다

사용시 주의점

MainThread 보다는 BackgroundThread에서 사용

ROOM 쿼리 호출을 MainThread에서 할 경우 아래와 같은 에러가 발생한다

```
java.lang.IllegalStateException: Cannot access database on the main thread since it may potentially lock the UI for a long period of time
```

많은 양의 쿼리를 하다보면 오랜기간동안 UI가 동작하지 않는 문제가 발생할 수 있다

DATABSE를 세팅할 때 allowMainThreadQueries() 설정값을 통해 mainThread 구동을 허용할 수 있으나, 샘플코드에서는 위와 같은 이유로 절대 쓰지말라고 알린다

사용시 주의점

비동기 처리를 위한 방법들

1. AsyncTask
2. RxJava
3. Java Thread

Thanks!

Any questions ?

You can find me at

- jeonyt89@gmail.com