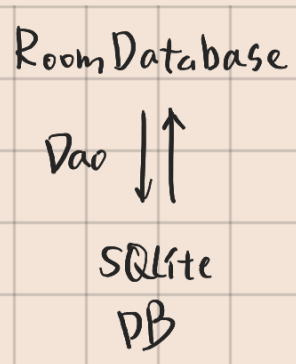
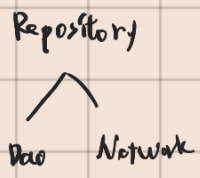




1. Add ROOM to Gradle

   /Gradle Scripts/ build.gradle (Module) → paste code

   /Gradle Scripts/ build.gradle (Project) → paste code

   → sync

2. Create /model/ Contact.java

   CRUD
   Data Access Object

3. Create interface : /data/ ContactDao.java

4. Create /util/ ContactRoomDatabase.java

5. Create /data/ ContactRepository.java

數據庫操作不可放在主線程(UI線程)

∵耗時會導致延遲

Asc : Ascending 升序

DESC : Descending 降序. 新的資料在最前面 →

查看DB文件：右下 Device File Explorer

→ data/data/com.logan.roombasic/database/

→ 選取3文件 → save as

→ 用 db Browser 看

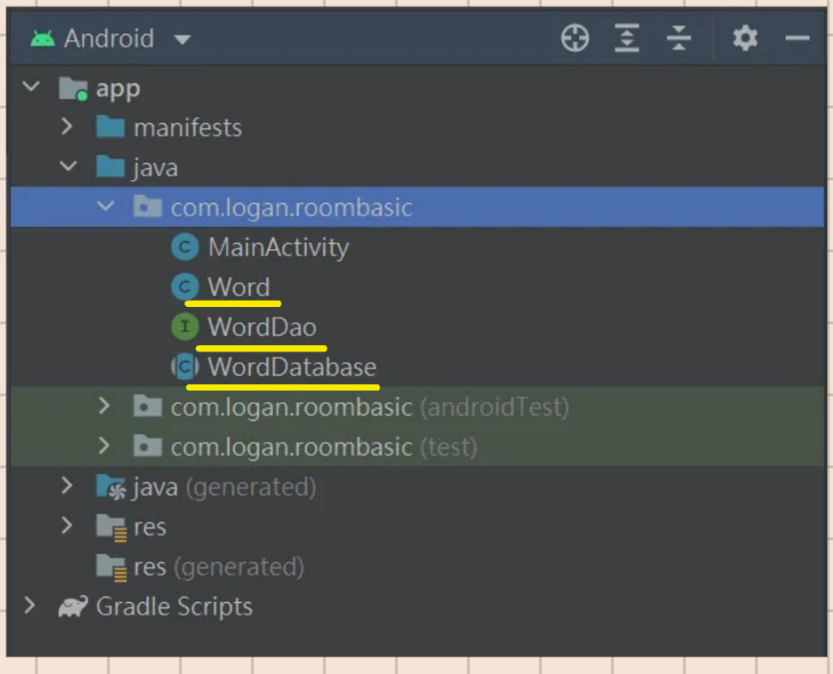
基本必備

- Entity ⟶ Word.java
- Dao ⟶ WordDao.java
- Database ⟶ WordDatabase.java

***.最簡易 Room 實作

1° New Project ⟶ 將 Room 加到 Gradle 中 (Module & Project)

2° App 架構



3° Word.java (Entity)

activity_main.xml | MainActivity.java | Word.java | WordDao.java | WordDatabase.java

```
@Entity ⟶ 告知這是一個 entity 的 class
public class Word {
    @PrimaryKey(autoGenerate = true)
    private int id;                                  ⟵ 自動產生 id

    @ColumnInfo(name = "english_word")
    private String word;                             ⟵ 設定行名稱
    @ColumnInfo(name = "chinese_meaning")
    private String chineseMeaning;

    public Word(String word, String chineseMeaning) {
        this.word = word;
        this.chineseMeaning = chineseMeaning;
    }                                                ⟵ constructor

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getWord() { return word; }         ⟵ getter & setter
```

```
34      public void setWord(String word) { this.word = word; }
37
38      public String getChineseMeaning() { return chineseMeaning; }
41
42      public void setChineseMeaning(String chineseMeaning) { this.chineseMeaning = chineseMeaning; }
45  }
46
```

setter

4° wordDao.java (Dao)

activity_main.xml × MainActivity.java × Word.java × WordDao.java × WordDatabase.java ×

```
1   package com.logan.roombasic;
2
3   import ...
10
11  @Dao
12  public interface WordDao {
13      @Insert → function内容会自动完成,只需命名&决定input
14      void insertWords(Word... words); // multi input
15
16      @Update
17      int updateWords(Word... words);
18                                  表示可以input多个Word obj
19      @Delete
20      void deleteWords(Word... words);
21
22      @Query("DELETE FROM WORD")
23      void deleteAllWords();
24
25      @Query("SELECT * FROM WORD ORDER BY ID DESC") // Descending DESC
26      List<Word> getAllWords();
27  }
```

5° WordDatabase.java (Database)

activity_main.xml × MainActivity.java × Word.java × WordDao.java × WordDatabase.java ×

```
1   package com.logan.roombasic;
2
3   import ...
5                                   一个DB可有多种不同的entity(Table)
6   @Database(entities = {Word.class}, version = 1, exportSchema = false)
7   public abstract class WordDatabase extends RoomDatabase {
8       public abstract  WordDao getWordDao();
9   }
10
```

可以写成{a,b,c}

6° MainActivity.java

activity_main.xml × MainActivity.java × Word.java × WordDao.java × WordDatabase.java ×

```
14  public class MainActivity extends AppCompatActivity {
15      WordDatabase wordDatabase;
16      WordDao wordDao;
17      Button buttonInsert, buttonUpdate, buttonClear, buttonDelete;
18      TextView textView;
19
20      @Override
21      protected void onCreate(Bundle savedInstanceState) {
22          super.onCreate(savedInstanceState);
23          setContentView(R.layout.activity_main);
```

```
        // Create database
1°      wordDatabase = Room.databaseBuilder( context: MainActivity.this,
創db            WordDatabase.class,  ← input Database 檔案
                name: "word_database")  ← 決定 Room存檔名稱
                .allowMainThreadQueries()
                .build();
                ↖ ***. Room的操作預設不能在 Main thread (UI)進行
                  !.要加入這一行!

        // Connect database to Dao
2°      wordDao = wordDatabase.getWordDao();
給db    textView = findViewById(R.id.textView);
        updateView();  ← 記得 update 到 textView中
Dao
        // Insert Button
        buttonInsert = findViewById(R.id.button_insert);
        buttonInsert.setOnClickListener(view -> {
            Word word1 = new Word( word: "Hello",  chineseMeaning: "你好");
            Word word2 = new Word( word: "World",  chineseMeaning: "世界");
3°          wordDao.insertWords(word1, word2);
            updateView();
利用            ↳ 使用Dao method
        });
Dao method
操作    // Update
        buttonUpdate = findViewById(R.id.button_update);
        buttonUpdate.setOnClickListener(view -> {
            Word word = new Word( word: "Hehe",  chineseMeaning: "哭阿");
            word.setId(46);  ← 利用id 更改數值
            wordDao.updateWords(word);
            updateView();
        });

        // Clear Button
        buttonClear = findViewById(R.id.button_clear);
        buttonClear.setOnClickListener(view -> {...});

        // Delete
        buttonDelete = findViewById(R.id.button_delete);
        buttonDelete.setOnClickListener(view -> {...});
    }

    void updateView(){
        List<Word> list = wordDao.getAllWords();
        String text = "";
        for (int i =0; i<list.size(); i++){
            Word word = list.get(i);
            text += word.getId() + ":" + word.getWord() + ":" + word.getChineseMeaning() + "\n";
        }
        textView.setText(text);
    }
}
```

米、統整

1) Entity：設定 table、行列名稱、id、constructor、getter setter

2) Dao：宣告 CRUD method 名稱和 input

3) Database：宣告 abstract getDao() method

4) 运用：1° 创 db

2° connect db 和 Dao

3° 使用 Dao CRUD method 操作

1° 改成 LiveData ⟶ 用 onChange method 取代原本 updateView()

2° 改成 singleton (Database class)

3° 创 AsyncTask ⟶ 取代原的 Dao method，改为 Async method (目的：移到背景)

4° 移到 ViewModel

5° 移到 Repository

ViewModel：管理 UI 数据

CRUD：应由 Repository 负责

***. 進階版 RooM

1° 繼續使用原 Entity

2° 將 Dao 中的 List 变 LiveData

目的：自动更新 UI

*. WordDao.java

activity_main.xml × MainActivity.java × WordRepository.java × WordViewModel.java × Word.java × WordDao.ja

```
1   package com.logan.roombasic;
2
3   import ...
11
12  @Dao
13  public interface WordDao {
14      @Insert
15      void insertWords(Word... words); // multi input
16
17      @Update
18      int updateWords(Word... words);
19
20      @Delete
21      void deleteWords(Word... words);
22
23      @Query("DELETE FROM WORD")
24      void deleteAllWords();
25
26      @Query("SELECT * FROM WORD ORDER BY ID DESC") // Descending DESC
27      LiveData<List <Word>> getAllWordsLive(); // Don't need update view anymore
```

```
29
```
改成 LiveData 類別

3° 將 Database 改為 singleton　　App 只使用 1 个 db

∵ 耗資源 ∴ 希望只產生一个 database obj，因此用 singleton

*. WordDatabase.java

WordDatabase.java | MainActivity.java | WordRepository.java | WordViewModel.java | WordDao.java

```java
package com.logan.roombasic;

import ...

@Database(entities = {Word.class}, version = 1, exportSchema = false)
public abstract class WordDatabase extends RoomDatabase {

    // singleton obj INSTANCE
    private  static WordDatabase INSTANCE;

    // Method for creating WordDatabase obj
    // singleton concept : if null, create one, otherwise, use the old one
    static synchronized WordDatabase getDatabase(Context context){
        if(INSTANCE == null){
            INSTANCE = Room.databaseBuilder(context.getApplicationContext()
                    , WordDatabase.class
                    , name: "word_database")
//                    .allowMainThreadQueries() // 新增AsyncTack後, 可以移除這行
                    .build();
        }
        return INSTANCE;
    }

    public abstract  WordDao getWordDao();
}
```

singleton

不同 thread 同時有 request，会排隊，保證只有一个 database obj

← 原本只有這行

***. 統整 (進階 ROOM 架構)

架構：MainActivity → ViewModel → Repository { Database, Dao } → Entity

interface Dao

Data：佈
Method：

- CRUDData()
- getAllDataLive()

Class Repository

Data: Database

Dao

LiveData

Method:

- Repository(); ← 初始化 database, dao
  1° init Database using singleton
  2° connect Dao to Database
  3° assign LiveData using Dao.getAllDataLive()
- getAllDataLive(); ← Getter, 供 ViewModel 取得 LiveData
  1° return LiveData 為了在 Main 用 LiveData 設定 observe 而存在
- Class CRUDAsyncTask ← 將 Dao 中的 CRUD method 背景化
  Data: Dao
  Method: CRUDAsyncTask(); (constructor)
  1° override doInBackground method
  2° put Dao CRUD method in doInBackground
- CRUDData(); ← 為了要在 "ViewModel" 中呼叫而建立
  1° call CRUDAsyncTask class & execute

Class ViewModel

Data: Repository

Method:

- ViewModel(); ← 初始化 Repository obj
  1° init Repository obj

這之于function和Repo中的名稱一樣，只是為了在Main中用viewModel obj呼叫，才再宣告一次

- getAllDataLive(); ← 用Repository的getter取得LiveData 內容也只是再call Repo的method
  1° return Repository.getAllDataLive()
- CRUDData(); ← 名稱和Repository的一樣，只是為了在Main中使用 ，再宣告一次
  1° Repository.CRUDData()

MainActivity

Data: ViewModel

Method:

1° viewModel = new ViewModelProvider(....)

初始化 ViewModel

2° viewModel.getAllDataLive().observe(override onchange ...)

利用 ViewModel → Repository的getter取得 Repo 中的 LiveData

再用 observe 設定即時更新 UI

3° viewModel.CRUDData()

call ViewModel method 操作 ROOM Database

4° 創 Repository

*. WordRepository.java

MainActivity.java × WordViewModel.java × WordRepository.java ×

```java
public class WordRepository {

    private WordDatabase wordDatabase;
    private WordDao wordDao;
    private LiveData<List<Word>> allWordsLive;

    // Constructor
    public WordRepository(Context context){

        // Create database using singleton
        wordDatabase = WordDatabase.getDatabase(context.getApplicationContext());

        // Connect database to Dao
        wordDao = wordDatabase.getWordDao();

        // Get LiveData in database
        allWordsLive = wordDao.getAllWordsLive();
    }

    // Getter
    public LiveData<List<Word>> getAllWordsLive() { return allWordsLive; }

    void insertWords(Word... words) { new InsertAsyncTask(wordDao).execute(words); }
    void updateWords(Word... words) { new UpdateAsyncTask(wordDao).execute(words); }
    void deleteWords(Word... words) { new DeleteAsyncTask(wordDao).execute(words); }
    void deleteAllWords(Word... words) { new DeleteAllAsyncTask(wordDao).execute(); }

    static class InsertAsyncTask extends AsyncTask<Word, Void, Void> { // 報告進度, 報告結果
        private WordDao wordDao;

        // Constructor
        public InsertAsyncTask(WordDao wordDao) { this.wordDao = wordDao; }

        @Override
        protected Void doInBackground(Word... words){
            wordDao.insertWords(words);
            return null;
        }

        //@Override
        //protected void onPostExecute(Void unused) {}        // 任務完成時呼叫,將資料帶回主線程
        //protected void onProgressUpdate(Void... values) {} // 回報進度,做進度條用
        //protected void onPreExecute() {}                    // 執行前呼叫
    }

    static class UpdateAsyncTask extends AsyncTask<Word, Void, Void>{
        private WordDao wordDao;

        // Constructor
```

為3在ViewModel使用

利用 AsyncTask class 將 Dao 的 CRUD method 移到背景

```java
        public UpdateAsyncTask(WordDao wordDao) { this.wordDao = wordDao; }

        @Override
        protected Void doInBackground(Word... words){
            wordDao.updateWords(words);
            return null;
        }
    }

    static class DeleteAsyncTask extends AsyncTask<Word, Void, Void>{
        private WordDao wordDao;

        // Constructor
        public DeleteAsyncTask(WordDao wordDao) { this.wordDao = wordDao; }

        @Override
        protected Void doInBackground(Word... words){
            wordDao.deleteWords(words);
            return null;
        }
    }

    static class DeleteAllAsyncTask extends AsyncTask<Void, Void, Void>{ // 不須輸入參數,改成void
        private WordDao wordDao;

        // Constructor
        public DeleteAllAsyncTask(WordDao wordDao) { this.wordDao = wordDao; }

        @Override
        protected Void doInBackground(Void... voids){
            wordDao.deleteAllWords();
            return null;
        }
    }
}
```

5° 創 ViewModel

*. WordViewModel.java

MainActivity.java | WordViewModel.java | WordRepository.java

```java
package com.logan.roombasic;

import ...

public class WordViewModel extends AndroidViewModel {

    private WordRepository wordRepository;

    Constructor
    public WordViewModel(@NonNull Application application) {
        super(application);

        wordRepository = new WordRepository(application);
    }

    Getter
    public LiveData<List<Word>> getAllWordsLive() { return wordRepository.getAllWordsLive(); }

    void insertWords(Word... words) { wordRepository.insertWords(words); }
    void updateWords(Word... words) { wordRepository.updateWords(words); }
```

```
    在Main中使用
    void deleteWords(Word... words) { wordRepository.deleteWords(words); }
    void deleteAllWords(Word... words) { wordRepository.deleteAllWords(); }

}
```

6° MainActivity.java

MainActivity.java | WordViewModel.java | WordRepository.java

```
public class MainActivity extends AppCompatActivity {
    Button buttonInsert, buttonUpdate, buttonClear, buttonDelete;
    TextView textView;
    WordViewModel wordViewModel;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        wordViewModel = new ViewModelProvider( owner: this).get(WordViewModel.class);
        textView = findViewById(R.id.textView);
        buttonInsert = findViewById(R.id.button_insert);
        buttonUpdate = findViewById(R.id.button_update);
        buttonClear = findViewById(R.id.button_clear);
        buttonDelete = findViewById(R.id.button_delete);

        //當資料有變,更新 textview,利用 LiveData 的 observe 達成
        wordViewModel.getAllWordsLive().observe( owner: MainActivity.this, new Observer<List<Word>>(){
            @Override
            public void onChanged(List<Word> words) {  // Invoke when data change
                String text = "";
                for (int i =0; i<words.size(); i++){
                    Word word = words.get(i);
                    text += word.getId() + ":" + word.getWord() + ":" + word.getChineseMeaning()
                            + "\n";
                }
                textView.setText(text);
            }
        });

        // Insert Button
        buttonInsert.setOnClickListener(view -> {
            Word word1 = new Word( word: "Hello",  chineseMeaning: "你好");
            Word word2 = new Word( word: "World",  chineseMeaning: "世界");
            wordViewModel.insertWords(word1, word2);
        });

        // Update Button
        buttonUpdate.setOnClickListener(view -> {
            Word word = new Word( word: "Hehe",  chineseMeaning: "哭阿");
            word.setId(46);
            wordViewModel.updateWords(word);
        });

        // Clear Button
        buttonClear.setOnClickListener(view -> {
            wordViewModel.deleteAllWords();
        });

        // Delete Button
        buttonDelete.setOnClickListener(view -> {
            Word word = new Word( word: "Doesn't matter",  chineseMeaning: "無關緊要");
```

```
72              word.setId(45); // delete row base on id
73              wordViewModel.deleteWords(word);
74          });
75      }
76
```