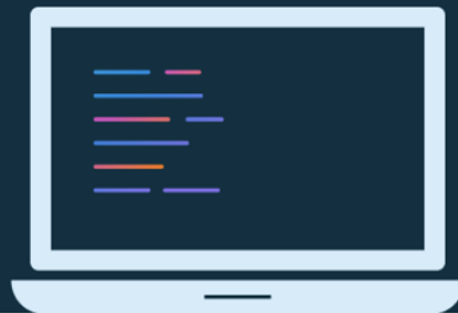




Bài học 9: Cấu trúc của ứng dụng (khả năng lưu trữ cố định)



Giới thiệu về bài học này

Bài học 9: Cấu trúc của ứng dụng (khả năng lưu trữ cố định)

- [Lưu trữ dữ liệu](#)
- [Thư viện lưu trữ Phòng](#)
- [Lập trình không đồng bộ](#)
- [Coroutine](#)
- [Kiểm tra cơ sở dữ liệu](#)
- [Tóm tắt](#)

Lưu trữ dữ liệu

Những cách lưu trữ dữ liệu trong ứng dụng Android

- Bộ nhớ dành riêng cho ứng dụng
- Bộ nhớ dùng chung (tệp được chia sẻ với các ứng dụng khác)
- Lựa chọn ưu tiên
- Cơ sở dữ liệu

Cơ sở dữ liệu là gì?

Tập hợp dữ liệu có cấu trúc có thể dễ dàng truy cập, tìm kiếm và sắp xếp, bao gồm:

- Bảng
- Hàng
- Cột

Ví dụ về cơ sở dữ liệu

person
_id
name
age
email

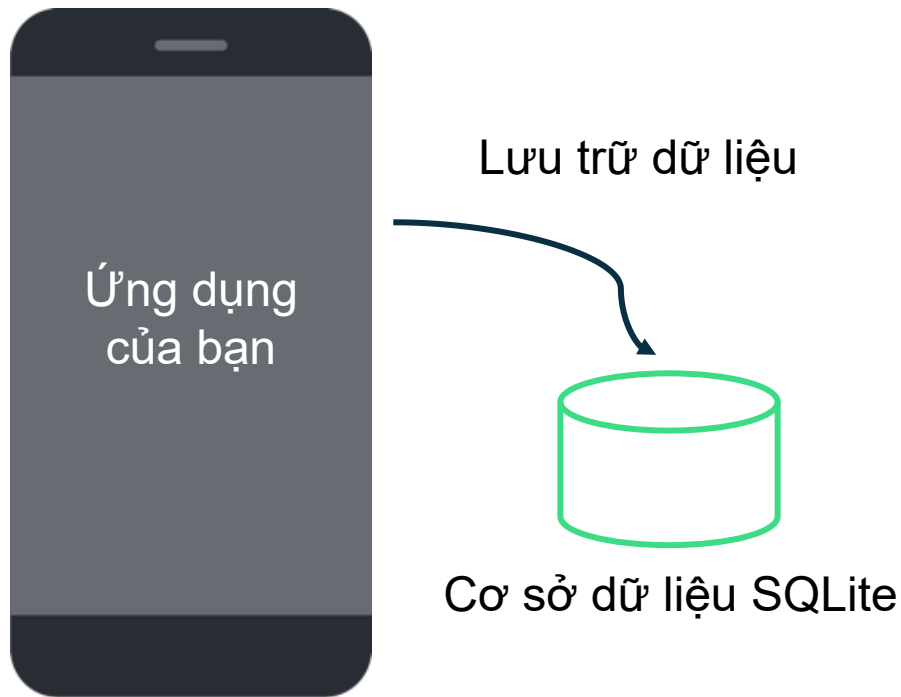
car
_id
make
model
year

Ngôn ngữ truy vấn có cấu trúc (SQL)

Dùng SQL để truy cập và sửa đổi cơ sở dữ liệu quan hệ.

- Tạo bảng mới
- Truy vấn dữ liệu
- Chèn dữ liệu mới
- Cập nhật dữ liệu
- Xóa dữ liệu

SQLite trong Android



Ví dụ về các lệnh SQLite

Create

```
INSERT INTO colors VALUES ("red", "#FF0000");
```

Read

```
SELECT * from colors;
```

Update

```
UPDATE colors SET hex="#DD0000" WHERE name="red";
```

Delete

```
DELETE FROM colors WHERE name = "red";
```


Tương tác trực tiếp với cơ sở dữ liệu

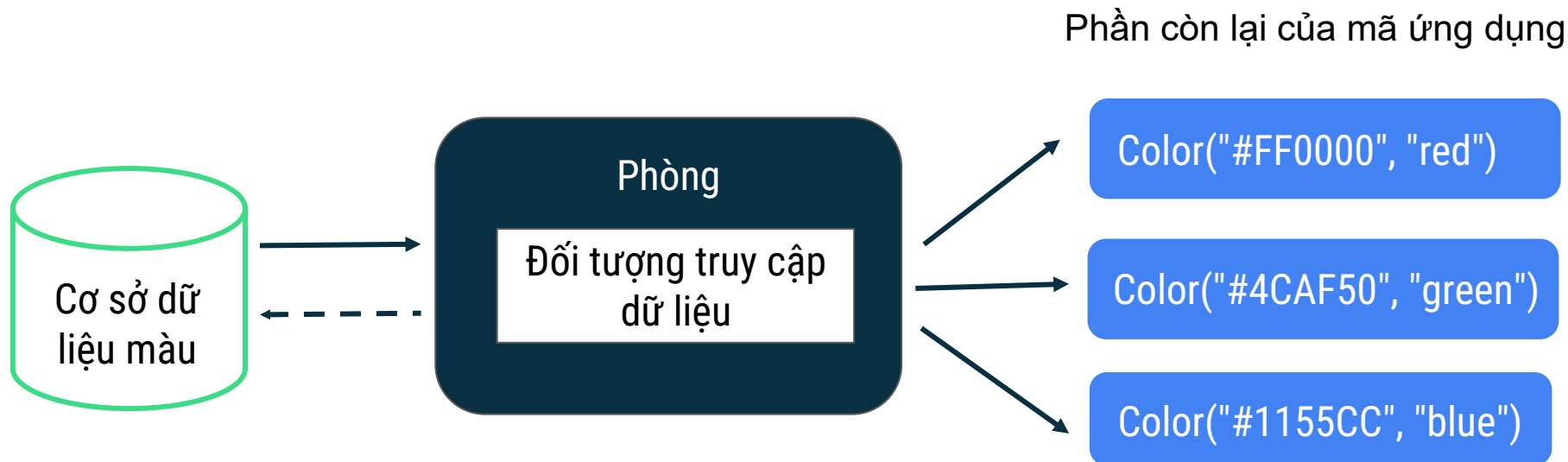
- Không có phương thức xác minh thời gian biên dịch đối với các truy vấn SQL thô
- Cần có nhiều mã nguyên mẫu để chuyển đổi giữa các đối tượng dữ liệu và \longleftrightarrow truy vấn SQL

Thư viện lưu trữ Phòng

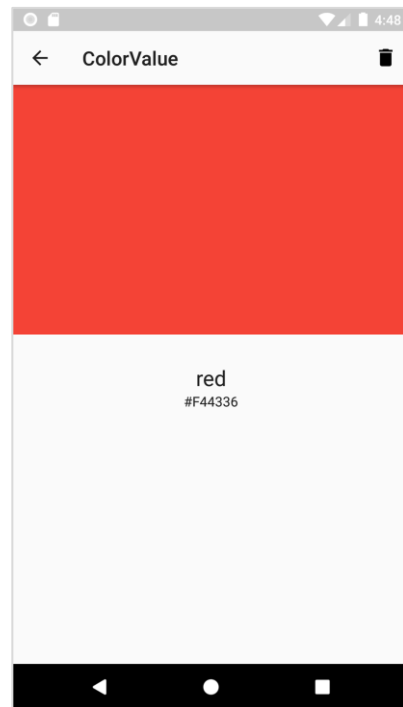
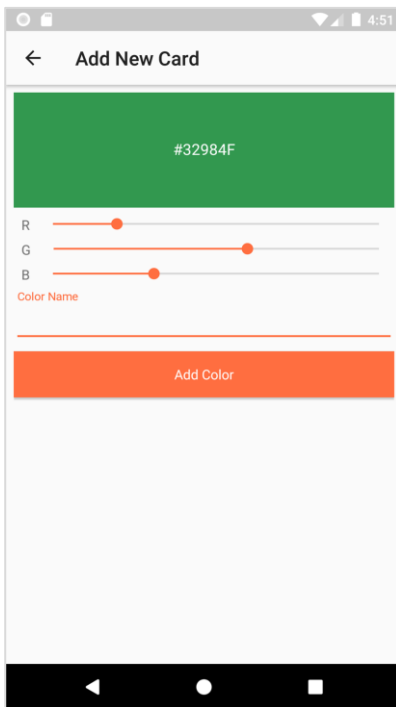
Thêm phần phụ thuộc vào Gradle

```
dependencies {  
    implementation "androidx.room:room-runtime:$room_version"  
    kapt "androidx.room:room-compiler:$room_version"  
  
    // Kotlin Extensions and Coroutines support for Room  
    implementation "androidx.room:room-ktx:$room_version"  
  
    // Test helpers  
    testImplementation "androidx.room:room-testing:$room_version"  
}
```

Phòng



Ứng dụng ColorValue



Phòng

- Thực thể

`Color`

- Đối tượng truy cập dữ liệu (DAO)

`ColorDao`

- Cơ sở dữ liệu

`ColorDatabase`

Lớp Color

```
data class Color {  
    val hex: String,  
    val name: String  
}
```

Chú thích

- Cung cấp thêm thông tin cho trình biên dịch

`@Entity` biểu thị lớp thực thể, `@Dao` biểu thị đối tượng truy cập dữ liệu (DAO), `@Database` biểu thị cơ sở dữ liệu

- Có thể nhận các tham số

```
@Entity(tableName = "colors")
```

- Có thể tự động tạo mã cho bạn

Thực thể

Lớp liên kết với bảng cơ sở dữ liệu SQLite

- `@Entity`
- `@PrimaryKey`
- `@ColumnInfo`

Ví dụ về thực thể

```
@Entity(tableName = "colors")
```

```
data class Color {
```

```
@PrimaryKey(autoGenerate = true) val _id: Int,
```

```
@ColumnInfo(name = "hex_color") val hex: String,
```

```
val name: String
```

}

colors

_id
hex_color
name

Đối tượng truy cập dữ liệu (DAO)

Làm việc với các lớp DAO thay vì truy cập trực tiếp vào cơ sở dữ liệu:

- Xác định các hoạt động tương tác với cơ sở dữ liệu trong DAO.
- Khai báo DAO ở dạng giao diện hoặc lớp trừu tượng.
- Phòng sẽ tạo phương thức triển khai DAO vào thời gian biên dịch.
- Phòng sẽ xác minh tất cả các truy vấn DAO của bạn vào thời gian biên dịch.

Ví dụ về DAO

@Dao

```
interface ColorDao {
```

```
    @Query("SELECT * FROM colors")
```

```
    fun getAll(): Array<Color>
```

```
    @Insert
```

```
    fun insert(vararg color: Color)
```

```
    @Update
```

```
    fun update(color: Color)
```

```
    @Delete
```

```
    fun delete(color: Color)
```

Query

@Dao

```
interface ColorDao {
```

```
    @Query("SELECT * FROM colors")
```

```
    fun getAll(): Array<Color>
```

```
    @Query("SELECT * FROM colors WHERE name = :name")
```

```
    fun getColorByName(name: String): LiveData<Color>
```

```
    @Query("SELECT * FROM colors WHERE hex_color = :hex")
```

```
    fun getColorByHex(hex: String): LiveData<Color>
```

Insert

```
@Dao
interface ColorDao {
    ...

    @Insert
    fun insert(vararg color: Color)

    ...
}
```

Update

```
@Dao
interface ColorDao {
    ...

    @Update
    fun update(color: Color)

    ...
}
```

Delete

```
@Dao
interface ColorDao {
    ...

    @Delete
    fun delete(color: Color)

    ...
}
```


Tạo cơ sở dữ liệu Phòng

- Chú thích lớp bằng `@Database` và thêm danh sách các thực thể:
`@Database(entities = [Color::class], version = 1)`
- Khai báo lớp trừu tượng mở rộng `RoomDatabase`:
`abstract class ColorDatabase : RoomDatabase() {`
 - Khai báo phương thức trừu tượng không có đối số trả về DAO:
`abstract fun colorDao(): ColorDao`

Ví dụ về cơ sở dữ liệu Phòng

```
@Database(entities = [Color::class], version = 1)
abstract class ColorDatabase : RoomDatabase() {
    abstract fun colorDao(): ColorDao
    companion object {
        @Volatile
        private var INSTANCE: ColorDatabase? = null
        fun getInstance(context: Context): ColorDatabase {
            ...
        }
    }
    ...
}
```

Tạo thực thể của cơ sở dữ liệu

```
fun getInstance(context: Context): ColorDatabase {  
    return INSTANCE ?: synchronized(this) {  
        INSTANCE ?: Room.databaseBuilder(  
            context.applicationContext,  
            ColorDatabase::class.java, "color_database"  
        )  
        .fallbackToDestructiveMigration()  
        .build()  
        .also { INSTANCE = it }  
    }  
}
```

Lấy và sử dụng DAO

Lấy DAO từ cơ sở dữ liệu:

```
val colorDao = ColorDatabase.getInstance(application).colorDao()
```

Tạo Màu mới và dùng DAO để chèn màu đó vào cơ sở dữ liệu:

```
val newColor = Color(hex = "#6200EE", name = "purple")  
colorDao.insert(newColor)
```

Lập trình không đồng bộ

Tác vụ chạy trong thời gian dài

- Tải thông tin xuống
- Đồng bộ hóa với máy chủ
- Ghi vào một tệp
- Tác vụ tính toán nặng
- Đọc hoặc ghi vào cơ sở dữ liệu

Nhu cầu lập trình không đồng bộ

- Bị giới hạn thời gian thực hiện tác vụ mà vẫn phải phản hồi nhanh
- Cân bằng với nhu cầu thực thi các tác vụ chạy trong thời gian dài
- Kiểm soát cách thức và vị trí thực thi các tác vụ

Lập trình không đồng bộ trên Android

- Luồng
- Lệnh gọi lại
- Cùng với nhiều lựa chọn khác

Bạn nên dùng cách nào?

Coroutine

Coroutine

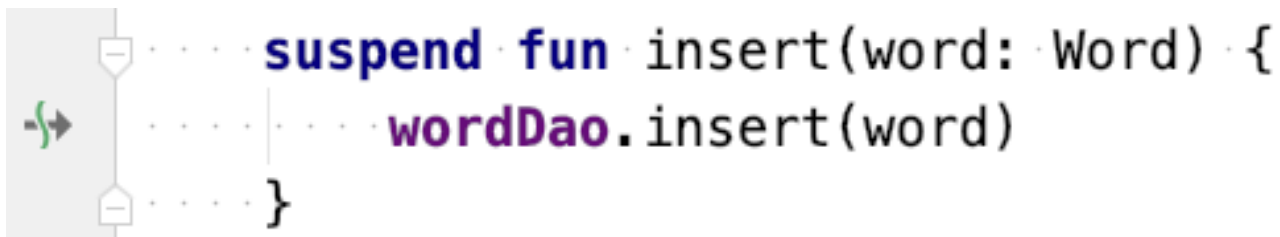
- Duy trì khả năng phản hồi của ứng dụng, đồng thời quản lý các tác vụ chạy trong thời gian dài.
- Đơn giản hóa mã không đồng bộ trong ứng dụng Android của bạn.
- Viết mã theo cách tuần tự
- Xử lý các trường hợp ngoại lệ bằng khối `try/catch`

Lợi ích của coroutine

- Dung lượng nhẹ
- Ít rò rỉ bộ nhớ
- Tích hợp sẵn tính năng hỗ trợ hủy
- Tích hợp Jetpack

Hàm tạm ngưng

- Thêm từ khóa xác định `suspend`
- Phải được gọi bằng các coroutine hoặc hàm tạm ngưng khác

A screenshot of a code editor showing a Kotlin function definition. The function is `suspend fun insert(word: Word) { wordDao.insert(word) }`. The word `suspend` is highlighted in blue. The word `wordDao` is highlighted in purple. On the left side of the code, there are three annotations: a green arrow pointing right, a white box with a minus sign, and a white box with a plus sign.

```
suspend fun insert(word: Word) {  
    wordDao.insert(word)  
}
```

Suspend và resume

- `suspend`

Tạm dừng quá trình thực thi coroutine hiện tại và lưu các biến cục bộ

- `resume`

Tự động tải trạng thái đã lưu và tiếp tục thực thi từ thời điểm mà bị tạm ngưng

Ví dụ

```
suspend fun fetchDocs() {  
    val docs = get("...")  
    show(docs)  
}
```

Main Thread
[stack]

suspend

resume



Thêm từ khóa xác định suspend vào các phương thức DAO

```
@Dao
interface ColorDao {

    @Query("SELECT * FROM colors")
    suspend fun getAll(): Array<Color>

    @Insert
    suspend fun insert(vararg color: Color)

    @Update
    suspend fun update(color: Color)

    @Delete
    suspend fun delete(color: Color)
```

Kiểm soát vị trí chạy coroutine

Phương thức gửi	Nội dung mô tả tác vụ	Ví dụ về tác vụ
<code>Dispatchers.Main</code>	Tác vụ giao diện người dùng và tác vụ không chặn (ngắn)	Cập nhật LiveData, gọi các hàm tạm ngưng
<code>Dispatchers.IO</code>	Tác vụ mạng và ổ đĩa	Cơ sở dữ liệu, tệp IO
<code>Dispatchers.Default</code>	Dùng nhiều CPU	Phân tích cú pháp JSON

withContext

```
suspend fun get(url: String) {  
  
    // Start on Dispatchers.Main  
  
    withContext(Dispatchers.IO) {  
        // Switches to Dispatchers.IO  
        // Perform blocking network IO here  
    }  
  
    // Returns to Dispatchers.Main  
}
```

CoroutineScope

Coroutine phải chạy trong `CoroutineScope`:

- Theo dõi mọi coroutine đã bắt đầu trong phạm vi đó (ngay cả những coroutine bị tạm ngưng)
- Đưa ra cách hủy các coroutine trong một phạm vi
- Là cầu nối giữa các hàm thông thường và coroutine

Ví dụ:

`GlobalScope`

`ViewModel` **có** `viewModelScope`

`Lifecycle` **có** `lifecycleScope`

Bắt đầu coroutine mới

- `launch` - no result needed

```
fun loadUI() {  
    launch {  
        fetchDocs()  
    }  
}
```

- `async` - can return a result

ViewModelScope

```
class MyViewModel: ViewModel() {  
  
    init {  
        viewModelScope.launch {  
            // Coroutine that will be canceled  
            // when the ViewModel is cleared  
        }  
    }  
    ...  
}
```

Ví dụ về viewModelScope

```
class ColorViewModel(val dao: ColorDao, application: Application)
    : AndroidViewModel(application) {

    fun save(color: Color) {
        viewModelScope.launch {
            colorDao.insert(color)
        }
    }

    ...
}
```

Kiểm tra cơ sở dữ liệu

Thêm phần phụ thuộc vào Gradle

```
android {  
    defaultConfig {  
        ...  
        testInstrumentationRunner "androidx.test.runner  
            .AndroidJUnitRunner"  
        testInstrumentationRunnerArguments clearPackageData: 'true'  
    }  
}  
  
dependencies {  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.0'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.1'  
}
```

Kiểm tra mã Android

- `@RunWith (AndroidJUnit4::class)`
- `@Before`
- `@After`
- `@Test`

Tạo lớp kiểm tra

```
@RunWith(AndroidJUnit4::class)
```

```
class DatabaseTest {
```

```
    private lateinit val colorDao: ColorDao
```

```
    private lateinit val db: ColorDatabase
```

```
    private val red = Color(hex = "#FF0000", name = "red")
```

```
    private val green = Color(hex = "#00FF00", name = "green")
```

```
    private val blue = Color(hex = "#0000FF", name = "blue")
```

```
    ...
```

Tạo và đóng cơ sở dữ liệu cho mỗi hoạt động kiểm tra

Trong tệp `DatabaseTest.kt`:

`@Before`

```
fun createDb() {  
    val context: Context = ApplicationProvider.getApplicationContext()  
    db = Room.inMemoryDatabaseBuilder(context, ColorDatabase::class.java)  
        .allowMainThreadQueries()  
        .build()  
    colorDao = db.colorDao()  
}
```

`@After`

```
@Throws(IOException::class)  
fun closeDb() = db.close()
```

Kiểm tra thao tác chèn và truy xuất từ cơ sở dữ liệu

Trong tệp `DatabaseTest.kt`:

```
@Test
@Throws(Exception::class)
fun insertAndRetrieve() {
    colorDao.insert(red, green, blue)
    val colors = colorDao.getAll()
    assert(colors.size == 3)
}
```

Tóm tắt

Tóm tắt

Trong Bài học 9, bạn đã tìm hiểu cách:

- Thiết lập và định cấu hình cơ sở dữ liệu bằng thư viện Phòng
- Dùng coroutine để lập trình không đồng bộ
- Dùng coroutine với Phòng
- Kiểm tra cơ sở dữ liệu

Tìm hiểu thêm

- [7 mẹo hay dành cho Phòng](#)
- [Thư viện lưu trữ Phòng](#)
- [Trang chủ SQLite](#)
- [Lưu dữ liệu bằng SQLite](#)
- [Hướng dẫn về coroutine](#)
- [Phương thức gửi – kotlinx-coroutines-core](#)
- [Coroutine trên Android \(phần I\): Tìm hiểu khái niệm cơ bản](#)
- [Coroutine trên Android \(phần II\): Bắt đầu](#)
- [Coroutine dễ dàng trong Android: viewModelScope](#)
- [Kiến thức cơ bản về coroutine của Kotlin](#)

Lộ trình

Thực hành những gì bạn đã học được bằng cách hoàn thành lộ trình này:

[Bài học 9: Cấu trúc của ứng dụng \(khả năng lưu trữ cố định\)](#)

