


Android 架构组件的最新进展 (下篇)



谷歌开发者 
已认证的官方帐号

已关注

6 人赞同了该文章

根据我们曾经做的调查，开发者们希望 Android 官方可以维护一些实用的组件库和架构实践，以降低中大型应用的开发门槛，这样开发团队就可以集中更多精力在实际业务的优化和改进上。

Jetpack 项目正是为了解决这些问题而诞生的，Jetpack 是一系列助力您更容易打造优秀 Android 应用的工具和组件，这些组件能帮助您遵循最佳实践、免除编写繁复的样板代码并简化复杂任务，从而使您可以专注于最核心的代码逻辑。其中 `androidx.*` 库与 Framework API 解耦，这能够提供向后兼容的同时，也能更频繁地更新。

Android Jetpack 中的**架构组件**可帮助您设计稳健、可测试且易维护的应用。从最初发布的管理 Activity 和 Fragment 生命周期的 **Lifecycle 库**和访问 SQLite 数据库的 **Room 库**，后来推出了**分页 (Paging)**、**导航 (Navigation)** 和管理后台任务的 **WorkManager 库**。根据 2019 年最新的开发者调查中，70% 以上的专业开发者用过这五个库当中的至少一个库进行应用开发。

这里我们分上下两篇介绍**架构组件**的最新更新，如果您还没有阅读本文的上篇，请点击[这里查看《Android 架构组件的最新进展 \(上篇\)》](#)。本篇将会继续为大家介绍分页库、Room 持久性库和 WorkManager。希望大家能在其中发现对自己的应用有帮助的全新功能以及改进：

分页库

Paging (分页)使得开发者可以逐步、高效地加载大量数据，从而节省用户的电池和流量。而且它和架构组件中的其他部分或者其他技术都能配合使用，比如 Room, Realm, Retrofit 等等。

为了让分页的使用更加便捷，在不久未来的版本里我们将提供：

- 内置的网络支持，而且提供错误处理机制
- Header 和 Footer 支持
- 更好的 RxJava 支持以及协程的集成

Room 持久性库

Room 是一个在 SQLite 上提供抽象层的持久存储库，您可以回顾[我们之前的介绍文章](#)了解更多 Room 的详细信息。

协程处理

在 Room 2.1 中，开发者可以通过 Kotlin 语言的 `suspend` 关键字让 Room 生成正确的协程代码，包括使用后台 dispatcher，这大大降低了开发者处理协程的工作量：

```
// Room 2.1
```

```
@Query("SELECT * FROM song WHERE songId = :songId")  
suspend fun getSongs(songId: String): List<Song>
```

```
@Insert  
suspend fun insertSongs(songs: List<Song>)
```

```
@Transaction
```

▲ 赞同 6 ▼

● 添加评论

➦ 分享

♥ 喜欢

★ 收藏

📄 申请转载

...



```

        deleteSongsWithIds(songs.map { it.songId })
        return songs
    }
}

```

另外，在 Room 2.1 中也提供了扩展函数让开发者方便地启动事务。它还会提供一个协程上下文 (CoroutineContext)，这样开发者可以更方便地执行多个数据库操作：

```

database.withTransaction {
    val songs = getSongsWithElapsedTimeLessThan(1000)
    deleteSongsWithIds(songs.map { it.songId })
    return songs
}

```

全文搜索

全文搜索功能是对 SQLite 的一个扩展，让其创建一个数据表从而更高效地检索数据。

在 Room 2.0 中，一个 Dao 的检索方法看起来可能是这样：

```

// Room 2.0

@Dao
interface SongDao {
    @Query("""
        SELECT *
        FROM Song
        WHERE songName LIKE '%' || :query || '%'
        OR albumName LIKE '%' || :query || '%'
        OR artistName LIKE '%' || :query || '%'
        """)
    fun searchSongs(query: String): List<Song>
}

```

△ 注意 WHERE 和 OR 语句的长度

而在 2.1 中，只需要加入一个 @Fts4 注解，就可以通过 MATCH 语句让一切都轻松很多：

```

// Room 2.1

@Entity
@Fts4
data class Song(
    @PrimaryKey
    @ColumnInfo(name = "rowid")
    val id: Long,
    val url: String,
    val songName: String,
    val albumName: String,
    val artistName: String
)

@Dao
interface SongDao {
    @Query("""
        SELECT *
        FROM Song
        WHERE Song MATCH :query
        """)
}

```



数据库视图

很像数据表，但又不完全一样。基本上，您可以像检索数据表一样检索数据库视图，但不能在其中插入数据。

在 2.1 中，您可以用 `@DatabaseView` 注解您的数据类，但这时您不需要创建一个数据表，而是直接将 `BigQuery` 放在注解部分，让其成为一个能快速检索的视图：

```

@DatabaseView("""
    SELECT
        Album.*,
        count(song_id) AS num_of_songs,
        sum(song_elapsed_time) AS total_time
    FROM Album
    JOIN AlbumSongRef ON (album_id = ref_album_id)
    JOIN Song ON (ref_song_id = song_id)
    GROUP BY album_id
""")
data class AlbumItem(
    @Embedded
    val album: Album,
    @ColumnName("num_of_songs")
    val numOfSongs: Int,
    @ColumnName("total_time")
    val totalTime: Long
)

```

而这个视图 (如上面的 `AlbumItem`) 可以像其他数据表一样使用：

```

@Query("""
    SELECT * FROM AlbumItem
    ORDER BY num_of_songs DESC
""")
fun getAlbumItemsByNumOfSongs(): List<AlbumItem>

```

扩展的 Rx 支持

在 Room 2.1 中，您使用的 `insert`, `update`, `delete` 方法能返回 `Completable`, `Maybe` 和 `Single`。而且在 `Query` 注解的方法里可以使用 Rx 作为返回类型，并处理 `update`, `insert` 或者 `delete` 这样的写入操作：

```

@Insert
fun addSong(song: Song): Completable

@update
fun updateSong(song: Song): Single<Int>

@Query("""
    UPDATE Song SET
    lastPlayedTime = :time
    WHERE id = :id
""")
fun updateSongPlayTime(Long: time, String: id): Completable

```

Room 接下来会实现增量注解处理，提高构建速度，另外也会进一步提升关系数据的处理效率，并提升数据迁移的便利性。在协程方面，则会加入 `Channel` 和 `Flow` 的支持。

WorkManager 是一个后台进程库，用于处理那些不需要即时处理的任务，而且可以在应用甚至设备重启后依然确保任务正确触发。另外，WorkManager 也支持按条件启动，比如根据网络连接状况的变化启动特定的任务。

性能和兼容性

按需配置

以往 WorkManager 需要在应用启动时就初始化，而按需配置 (On-demand Configuration) 可以让开发者仅在需要时才启动 WorkManager。在 WorkManager 2.1 中，您可以通过重载 Configuration.Provider 中的方法来获得一个 WorkManager 的配置对象。

```
// WorkManager 2.1.0

class MyApp : Application(), Configuration.Provider {

    override fun getWorkManagerConfiguration() : Configuration {
        return Configuration.Builder()
            // set your options here
            .build()
    }
}
```

在目前 WorkManager 2.0 中 WorkManager.getInstance() 方法并不需要开发者提供参数，而在 2.1 中开发者传入 context 参数后，WorkManager 如果没有初始化，它会基于参数访问 application 对象并获取到配置：

```
WorkManager.getInstance().enqueue(...)
```

Google Play services 集成

此功能即将到来，而且提升了在 Marshmallow 之前版本的设备上的运行性能。而且**这个集成是可选的**，开发者可以根据需要自行选择是否集成。

兼容性改进

兼容性方面，我们主要是在做 "幕后工作"。比如和 OEM 沟通，确保不同的设备能拥有一致的应用退出操作。

测试

第一点，也是开发者们一直有提到的: Robolectric 支持。Robolectric 是一个高效可靠的 Android 单元测试框架，现在已被全面支持。

第二点，Worker 已提供单元测试的支持。

您可以使用 TestWorkerBuilder:

```
// WorkManager 2.1.0

// Create a test worker
val request = OneTimeWorkRequestBuilder<MyWorker>.build()
val worker = TestWorkerBuilder
    .from(context, request, executor)
    .build()
```

```
val result = worker.doWork()
assertThat(result, `is`(Result.success()))
assertThat(...)
```

也可以使用 TestListenableWorkerBuilder:

```
// WorkManager 2.1.0

// Or create a listenable worker
val request = OneTimeWorkRequestBuilder<MyWorker>.build()
val listenableWorker = TestListenableWorkerBuilder
    .from(context, request)
    .build()

// Test its behavior
val result = listenableWorker.startWork().get()
assertThat(result, `is`(Result.success()))
assertThat(...)
```

WorkManager 的下一步

我们正在努力实现前台服务的支持，让您可以在前台也能使用 WorkManager API。

感谢大家对本次连载的关注，希望在了解完架构组件的最新进展后，大家能在其中找到适合自己应用的功能。您也可以观看 📺 [下面的视频](#) 📺 重温我们对架构组件进展的介绍。

- 腾讯视频链接

Android 架构组件的最新进展_腾讯视频
[v.qq.com](#)



- Bilibili 视频链接

<https://www.bilibili.com/video/av71179658/>
[www.bilibili.com](#)



如果对架构组件有疑问或者建议，欢迎在评论区和我们分享。

[点击这里](#)进一步了解 Android Jetpack

发布于 2019-10-18

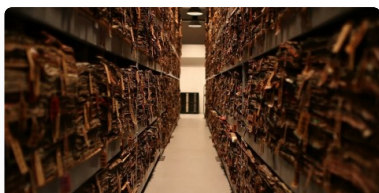
[Android](#) [Android 开发](#)



更完整的 Android 拍照实现教程

tevin

发表于极光日报



[译] 构建 Android APP 一定要绕过的 30 个坑

Glowi...

发表于稀土



Android SDK开发与使用的那些事儿

brucevanfdm



Android 源码分析 —— 从 Toast 出发

mzlogin

还没有评论

写下你的评论...

